

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	79
9.1	<code>\footnote</code>	79
9.2	<code>\appendix</code>	80
9.3	appendix package	81
9.4	amsmath package	82
9.5	mathtools package	85
9.6	breqn package	86
9.7	listings package	86
9.8	enumitem 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). Finally, a fix came to the new hook management system (`ltxcmdhooks`) with the 2021-11-15 kernel, with implications to the hook we add to `\appendix` (see <https://tex.stackexchange.com/q/617905> and <https://github.com/latex3/latex2e/pull/699>, thanks Phelype Oleinik). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.

TODO Bump this to 2021-11-15 when the release comes.

```
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 ‘`ltcounts.dtx`’ in ‘`source2e`’). Besides, there may be a chain of resetting counters, which must be taken into account: if ‘counterC’ gets reset by ‘counterB’, and ‘counterB’ gets reset by ‘counterA’, stepping the latter affects all three of them.

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

`__zrefclever_get_enclosing_counters_value:n`

Recursively generate a *sequence* of “enclosing counters” values, for a given `<counter>` and leave it in the input stream. These functions must be expandable, since they get called from `\zref@newprop` and are the ones responsible for generating the desired information when the label is being set. Note that the order in which we are getting this information is reversed, since we are navigating the counter reset chain bottom-up. But it is very hard to do otherwise here where we need expandable functions, and easy to handle at the reading side.

```

\__zrefclever_get_enclosing_counters_value:n {<counter>}

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

```

```

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

```

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

```

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

```

(End definition for `_zrefclever_get_enclosing_counters_value:n`.)

`_zrefclever_counter_reset_by:n`

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

```

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

```

(End definition for `_zrefclever_counter_reset_by:n`.)

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

```

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

```

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

```

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

```

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

4 Plumbing

4.1 Messages

```

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

```

```

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   }

```

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:`.

```

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

```

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

```

299 \seq_const_from_clist:Nn
300 \c__zrefclever_ref_options_font_seq
301 {
302   namefont ,
303   reffont ,
304 }

```

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

```

305 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
306 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
307 \c__zrefclever_ref_options_possibly_type_specific_seq
308 \c__zrefclever_ref_options_type_names_seq
309 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
310 \c__zrefclever_ref_options_typesetup_seq
311 \c__zrefclever_ref_options_font_seq
312 \seq_new:N \c__zrefclever_ref_options_reference_seq
313 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
314 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
315 \c__zrefclever_ref_options_possibly_type_specific_seq
316 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
317 \c__zrefclever_ref_options_reference_seq
318 \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”.

```

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

320 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
321 {
322   \group_begin:
323   \tl_if_empty:nF {#2}
324   {
325     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
326     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
327     {
328       \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
329       \prop_new:c { g__zrefclever_dict_ #2 _prop }
330       \tl_set:Nn \l__zrefclever_dict_language_tl {#2}
331       \keys_set:nn { zref-clever / declarelang } {#1}
332     }
333   }
334   \group_end:
335 }
336 \@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 { $\langle language alias \rangle$ } { $\langle aliased language \rangle$ }

337 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
338 {
339   \tl_if_empty:nF {#1}
340   {
341     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
342     {
343       \exp_args:NNnx
344       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
345       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
346     }
347     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
348   }
349 }
350 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

351 \keys_define:nn { zref-clever / declarelang }
352 {
353   declension .code:n =
354   {
355     \prop_gput:cnn
356     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
357     { declension } {#1}
358   } ,
359   declension .value_required:n = true ,
360   gender .code:n =
361   {
362     \prop_gput:cnn
363     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
364     { gender } {#1}
365   } ,
366   gender .value_required:n = true ,
367   allcaps .code:n =
368   {
369     \prop_gput:cnn
370     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
371     { allcaps } { true }
372   } ,
373   allcaps .value_forbidden:n = true ,
374 }

```

`_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.

```

375 \cs_new_protected:Npn \__zrefclever_process_language_options:
376 {
377   \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
378   { \l__zrefclever_ref_language_tl }
379   \l__zrefclever_dict_language_tl
380   {

```

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.

```

381   \exp_args:NNx \seq_set_from_clist:Nn
382   \l__zrefclever_dict_declension_seq
383   {
384     \prop_item:cn
385     {
386       g__zrefclever_dict_
387       \l__zrefclever_dict_language_tl _prop

```

```

388     }
389     { declension }
390 }
391 \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
392 {
393     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
394     {
395         \msg_warning:nxxx { zref-clever }
396         { language-no-decl-ref }
397         { \l__zrefclever_ref_language_tl }
398         { \l__zrefclever_ref_decl_case_tl }
399         \tl_clear:N \l__zrefclever_ref_decl_case_tl
400     }
401 }
402 {
403     \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
404     {
405         \seq_get_left:NN \l__zrefclever_dict_declension_seq
406         \l__zrefclever_ref_decl_case_tl
407     }
408     {
409         \seq_if_in:NVF \l__zrefclever_dict_declension_seq
410         \l__zrefclever_ref_decl_case_tl
411         {
412             \msg_warning:nxxx { zref-clever }
413             { unknown-decl-case }
414             { \l__zrefclever_ref_decl_case_tl }
415             { \l__zrefclever_ref_language_tl }
416             \seq_get_left:NN \l__zrefclever_dict_declension_seq
417             \l__zrefclever_ref_decl_case_tl
418         }
419     }
420 }

```

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.

```

421 \exp_args:NNx \seq_set_from_clist:Nn
422 \l__zrefclever_dict_gender_seq
423 {
424     \prop_item:cn
425     {
426         g__zrefclever_dict_
427         \l__zrefclever_dict_language_tl _prop
428     }
429     { gender }
430 }
431 \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
432 {
433     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
434     {
435         \msg_warning:nxxxx { zref-clever }
436         { language-no-gender }
437         { \l__zrefclever_ref_language_tl }

```

```

438         { g }
439         { \l__zrefclever_ref_gender_tl }
440         \tl_clear:N \l__zrefclever_ref_gender_tl
441     }
442 }
443 {
444     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
445     {
446         \seq_if_in:NVF \l__zrefclever_dict_gender_seq
447         \l__zrefclever_ref_gender_tl
448         {
449             \msg_warning:nxxx { zref-clever }
450             { gender-not-declared }
451             { \l__zrefclever_ref_language_tl }
452             { \l__zrefclever_ref_gender_tl }
453             \tl_clear:N \l__zrefclever_ref_gender_tl
454         }
455     }
456 }

```

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

```

457     \str_if_eq:eeT
458     {
459         \prop_item:cn
460         {
461             g__zrefclever_dict_
462             \l__zrefclever_dict_language_tl _prop
463         }
464         { allcaps }
465     }
466     { true }
467     { \bool_set_true:N \l__zrefclever_capitalize_bool }
468 }
469 {

```

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

```

470     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
471     {
472         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
473         { \l__zrefclever_ref_decl_case_tl }
474         { \l__zrefclever_ref_language_tl }
475         \tl_clear:N \l__zrefclever_ref_decl_case_tl
476     }
477     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
478     {
479         \msg_warning:nxxxx { zref-clever }
480         { language-no-gender }
481         { \l__zrefclever_ref_language_tl }
482         { g }
483         { \l__zrefclever_ref_gender_tl }
484         \tl_clear:N \l__zrefclever_ref_gender_tl
485     }
486 }

```


(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

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

```

488 \seq_new:N \g_zrefclever_loaded_dictionaries_seq

(End definition for \g_zrefclever_loaded_dictionaries_seq.)

```

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

```

489 \bool_new:N \l_zrefclever_load_dict_verbose_bool

(End definition for \l_zrefclever_load_dict_verbose_bool.)

```

`__zrefclever_provide_dictionary:n` Load dictionary for known *<language>* if it is available and if it has not already been loaded.

```

\__zrefclever_provide_dictionary:n {<language>}

490 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
491 {
492   \group_begin:
493   \@bsphack
494   \prop_get:NnNTF \g_zrefclever_languages_prop {#1}
495   \l_zrefclever_dict_language_tl
496   {
497     \seq_if_in:NVF
498     \g_zrefclever_loaded_dictionaries_seq
499     \l_zrefclever_dict_language_tl
500     {
501       \exp_args:Nx \file_get:nnNTF
502       { zref-clever- \l_zrefclever_dict_language_tl .dict }
503       { \ExplSyntaxOn }
504       \l_tmpa_tl
505       {
506         \tl_clear:N \l_zrefclever_setup_type_tl
507         \exp_args:NNx \seq_set_from_clist:Nn
508         \l_zrefclever_dict_declension_seq
509         {
510           \prop_item:cn
511           {
512             g_zrefclever_dict_
513             \l_zrefclever_dict_language_tl _prop
514           }
515           { declension }
516         }
517         \seq_if_empty:NTF \l_zrefclever_dict_declension_seq
518         { \tl_clear:N \l_zrefclever_dict_decl_case_tl }
519         {
520           \seq_get_left:NN \l_zrefclever_dict_declension_seq
521           \l_zrefclever_dict_decl_case_tl
522         }
523         \exp_args:NNx \seq_set_from_clist:Nn
524         \l_zrefclever_dict_gender_seq
525         {
526           \prop_item:cn

```

```

527         {
528             g__zrefclever_dict_
529             \l__zrefclever_dict_language_tl _prop
530         }
531         { gender }
532     }
533     \keys_set:nV { zref-clever / dictionary } \l_tmpa_tl
534     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
535     \l__zrefclever_dict_language_tl
536     \msg_note:nnx { zref-clever } { dict-loaded }
537     { \l__zrefclever_dict_language_tl }
538 }
539 {
540     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
541     {
542         \msg_warning:nnx { zref-clever } { dict-not-available }
543         { \l__zrefclever_dict_language_tl }
544     }

```

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.

```

545         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
546         \l__zrefclever_dict_language_tl
547     }
548 }
549 }
550 {
551     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
552     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
553 }
554 \@esphack
555 \group_end:
556 }
557 \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>}

558 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
559 {
560     \group_begin:
561     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
562     \__zrefclever_provide_dictionary:n {#1}
563     \group_end:
564 }
565 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for `_zrefclever_provide_dictionary_verbose: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>}

566 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
567 {
568   \exp_args:Nnx \prop_gput_if_new:cnn
569   { g__zrefclever_dict_ \l_zrefclever_dict_language_tl _prop }
570   { type- \l_zrefclever_setup_type_tl - #1 } {#2}
571 }
572 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
573 {
574   \prop_gput_if_new:cnn
575   { g__zrefclever_dict_ \l_zrefclever_dict_language_tl _prop }
576   { default- #1 } {#2}
577 }

```

(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.

```

578 \keys_define:nn { zref-clever / dictionary }
579 {
580   type .code:n =
581   {
582     \tl_if_empty:nTF {#1}
583     { \tl_clear:N \l_zrefclever_setup_type_tl }
584     { \tl_set:Nn \l_zrefclever_setup_type_tl {#1} }
585   } ,
586   case .code:n =
587   {
588     \seq_if_empty:NTF \l_zrefclever_dict_declension_seq
589     {
590       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
591       { \l_zrefclever_dict_language_tl } {#1}
592     }
593     {
594       \seq_if_in:NnTF \l_zrefclever_dict_declension_seq {#1}
595       { \tl_set:Nn \l_zrefclever_dict_decl_case_tl {#1} }
596       {
597         \msg_info:nxxx { zref-clever } { unknown-decl-case }
598         {#1} { \l_zrefclever_dict_language_tl }
599         \seq_get_left:NN \l_zrefclever_dict_declension_seq

```

```

600         \l__zrefclever_dict_decl_case_tl
601     }
602 }
603 },
604 case .value_required:n = true ,
605 gender .code:n =
606 {
607     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
608     {
609         \msg_info:nnxxx { zref-clever } { language-no-gender }
610         { \l__zrefclever_dict_language_tl } { gender } {#1}
611     }
612     {
613         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
614         {
615             \msg_info:nnn { zref-clever }
616             { option-only-type-specific } { gender }
617         }
618         {
619             \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
620             { \__zrefclever_provide_dict_type_transl:nn { gender } {#1} }
621             {
622                 \msg_info:nnxx { zref-clever } { gender-not-declared }
623                 { \l__zrefclever_dict_language_tl } {#1}
624             }
625         }
626     }
627 },
628 gender .value_required:n = true ,
629 }
630 \seq_map_inline:Nn
631 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
632 {
633     \keys_define:nn { zref-clever / dictionary }
634     {
635         #1 .value_required:n = true ,
636         #1 .code:n =
637         {
638             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
639             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
640             {
641                 \msg_info:nnn { zref-clever }
642                 { option-not-type-specific } {#1}
643             }
644         } ,
645     }
646 }
647 \seq_map_inline:Nn
648 \c__zrefclever_ref_options_possibly_type_specific_seq
649 {
650     \keys_define:nn { zref-clever / dictionary }
651     {
652         #1 .value_required:n = true ,
653         #1 .code:n =

```

```

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

```

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.

```

685 \prop_new:N \g__zrefclever_fallback_dict_prop
686 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
687 {
688     tpairsep = {,~} ,
689     tlistsep = {,~} ,
690     tlastsep = {,~} ,
691     notesep = {~} ,

```

```

692     namesep   = {\nobreakspace} ,
693     pairsep   = {,~} ,
694     listsep   = {,~} ,
695     lastsep   = {,~} ,
696     rangesep  = {\textendash} ,
697     refpre    = {} ,
698     refpos    = {} ,
699 }

```

Get translations

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

```

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

700 \prg_new_protected_conditional:Npnn
701   \_zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
702 {
703   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
704   \l__zrefclever_dict_language_tl
705   {
706     \prop_get:cnNTF
707     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
708     { type- #2 - #3 } #4
709     { \prg_return_true: }
710     { \prg_return_false: }
711   }
712   { \prg_return_false: }
713 }
714 \prg_generate_conditional_variant:Nnn
715   \_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 {\langle language \rangle} {\langle key \rangle}
        \langle tl variable \rangle {\langle false code \rangle}

716 \prg_new_protected_conditional:Npnn
717   \_zrefclever_get_default_transl:nnN #1#2#3 { F }
718 {
719   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
720   \l__zrefclever_dict_language_tl
721   {
722     \prop_get:cnNTF
723     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
724     { default- #2 } #3
725     { \prg_return_true: }
726     { \prg_return_false: }

```


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_t1`, check if first here with `\zref@ifpropundefined`: close it at the door.

```

747 \tl_new:N \l__zrefclever_ref_property_t1
748 \keys_define:nn { zref-clever / reference }
749 {
750   ref .choice: ,
751   ref / default .code:n =
752     { \tl_set:Nn \l__zrefclever_ref_property_t1 { default } } ,
753   ref / zc@thecnt .code:n =
754     { \tl_set:Nn \l__zrefclever_ref_property_t1 { zc@thecnt } } ,
755   ref / page .code:n =
756     { \tl_set:Nn \l__zrefclever_ref_property_t1 { page } } ,
757   ref / title .code:n =
758     {
759       \AddToHook { begindocument }
760       {
761         \@ifpackageloaded { zref-titleref }
762         { \tl_set:Nn \l__zrefclever_ref_property_t1 { title } }
763         {
764           \msg_warning:nn { zref-clever } { missing-zref-titleref }
765           \tl_set:Nn \l__zrefclever_ref_property_t1 { default }
766         }
767       }
768     } ,
769   ref .initial:n = default ,
770   ref .default:n = default ,
771   page .meta:n = { ref = page } ,
772   page .value_forbidden:n = true ,
773 }
774 \AddToHook { begindocument }
775 {
776   \@ifpackageloaded { zref-titleref }
777   {
778     \keys_define:nn { zref-clever / reference }
779     {
780       ref / title .code:n =
781       { \tl_set:Nn \l__zrefclever_ref_property_t1 { title } }
782     }
783   }
784   {
785     \keys_define:nn { zref-clever / reference }
786     {
787       ref / title .code:n =
788       {
789         \msg_warning:nn { zref-clever } { missing-zref-titleref }
790         \tl_set:Nn \l__zrefclever_ref_property_t1 { default }
791       }
792     }
793   }
794 }
```

typeset option

```
795 \bool_new:N \l__zrefclever_typeset_ref_bool
796 \bool_new:N \l__zrefclever_typeset_name_bool
797 \keys_define:nn { zref-clever / reference }
798 {
799   typeset .choice: ,
800   typeset / both .code:n =
801   {
802     \bool_set_true:N \l__zrefclever_typeset_ref_bool
803     \bool_set_true:N \l__zrefclever_typeset_name_bool
804   } ,
805   typeset / ref .code:n =
806   {
807     \bool_set_true:N \l__zrefclever_typeset_ref_bool
808     \bool_set_false:N \l__zrefclever_typeset_name_bool
809   } ,
810   typeset / name .code:n =
811   {
812     \bool_set_false:N \l__zrefclever_typeset_ref_bool
813     \bool_set_true:N \l__zrefclever_typeset_name_bool
814   } ,
815   typeset .initial:n = both ,
816   typeset .value_required:n = true ,
817
818   noname .meta:n = { typeset = ref },
819   noname .value_forbidden:n = true ,
820 }
```

sort option

```
821 \bool_new:N \l__zrefclever_typeset_sort_bool
822 \keys_define:nn { zref-clever / reference }
823 {
824   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
825   sort .initial:n = true ,
826   sort .default:n = true ,
827   nosort .meta:n = { sort = false },
828   nosort .value_forbidden:n = true ,
829 }
```

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`.

```
830 \seq_new:N \l__zrefclever_typesort_seq
831 \keys_define:nn { zref-clever / reference }
832 {
833   typesort .code:n =
834   {
835     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
836     \seq_reverse:N \l__zrefclever_typesort_seq
837   } ,
838   typesort .initial:n =
```

```

839     { part , chapter , section , paragraph },
840     typesort .value_required:n = true ,
841     notypesort .code:n =
842     { \seq_clear:N \l__zrefclever_typesort_seq } ,
843     notypesort .value_forbidden:n = true ,
844 }

```

comp option

```

845 \bool_new:N \l__zrefclever_typeset_compress_bool
846 \keys_define:nn { zref-clever / reference }
847 {
848     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
849     comp .initial:n = true ,
850     comp .default:n = true ,
851     nocomp .meta:n = { comp = false },
852     nocomp .value_forbidden:n = true ,
853 }

```

range option

```

854 \bool_new:N \l__zrefclever_typeset_range_bool
855 \keys_define:nn { zref-clever / reference }
856 {
857     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
858     range .initial:n = false ,
859     range .default:n = true ,
860 }

```

cap and capfirst options

```

861 \bool_new:N \l__zrefclever_capitalize_bool
862 \bool_new:N \l__zrefclever_capitalize_first_bool
863 \keys_define:nn { zref-clever / reference }
864 {
865     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
866     cap .initial:n = false ,
867     cap .default:n = true ,
868     nocap .meta:n = { cap = false },
869     nocap .value_forbidden:n = true ,
870
871     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
872     capfirst .initial:n = false ,
873     capfirst .default:n = true ,
874 }

```

abbrev and noabbrevfirst options

```

875 \bool_new:N \l__zrefclever_abbrev_bool
876 \bool_new:N \l__zrefclever_noabbrev_first_bool
877 \keys_define:nn { zref-clever / reference }
878 {
879     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
880     abbrev .initial:n = false ,
881     abbrev .default:n = true ,
882     noabbrev .meta:n = { abbrev = false },
883     noabbrev .value_forbidden:n = true ,

```

```

884
885     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
886     noabbrevfirst .initial:n = false ,
887     noabbrevfirst .default:n = true ,
888 }

```

S option

```

889 \keys_define:nn { zref-clever / reference }
890 {
891     S .meta:n =
892     { capfirst = true , noabbrevfirst = true },
893     S .value_forbidden:n = true ,
894 }

```

hyperref option

```

895 \bool_new:N \l__zrefclever_use_hyperref_bool
896 \bool_new:N \l__zrefclever_warn_hyperref_bool
897 \keys_define:nn { zref-clever / reference }
898 {
899     hyperref .choice: ,
900     hyperref / auto .code:n =
901     {
902         \bool_set_true:N \l__zrefclever_use_hyperref_bool
903         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
904     } ,
905     hyperref / true .code:n =
906     {
907         \bool_set_true:N \l__zrefclever_use_hyperref_bool
908         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
909     } ,
910     hyperref / false .code:n =
911     {
912         \bool_set_false:N \l__zrefclever_use_hyperref_bool
913         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
914     } ,
915     hyperref .initial:n = auto ,
916     hyperref .default:n = auto
917 }
918 \AddToHook { begindocument }
919 {
920     \@ifpackageloaded { hyperref }
921     {
922         \bool_if:NT \l__zrefclever_use_hyperref_bool
923         { \RequirePackage { zref-hyperref } }
924     }
925     {
926         \bool_if:NT \l__zrefclever_warn_hyperref_bool
927         { \msg_warning:nn { zref-clever } { missing-hyperref } }
928         \bool_set_false:N \l__zrefclever_use_hyperref_bool
929     }
930     \keys_define:nn { zref-clever / reference }
931     {
932         hyperref .code:n =
933         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }

```

```

934     }
935 }

```

nameinlink option

```

936 \str_new:N \l__zrefclever_nameinlink_str
937 \keys_define:nn { zref-clever / reference }
938 {
939     nameinlink .choice: ,
940     nameinlink / true .code:n =
941     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
942     nameinlink / false .code:n =
943     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
944     nameinlink / single .code:n =
945     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
946     nameinlink / tsingle .code:n =
947     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
948     nameinlink .initial:n = tsingle ,
949     nameinlink .default:n = true ,
950 }

```

preposinlink option

```

951 \bool_new:N \l__zrefclever_preposinlink_bool
952 \keys_define:nn { zref-clever / reference }
953 {
954     preposinlink .bool_set:N = \l__zrefclever_preposinlink_bool ,
955     preposinlink .initial:n = false ,
956     preposinlink .default:n = true ,
957 }

```

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`.

```

958 \tl_new:N \l__zrefclever_ref_language_tl
959 \tl_new:N \l__zrefclever_main_language_tl
960 \tl_new:N \l__zrefclever_current_language_tl
961 \AddToHook { begindocument }
962 {
963   \@ifpackageloaded { babel }
964   {
965     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
966     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
967   }
968   {
969     \@ifpackageloaded { polyglossia }
970     {
971       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
972       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
973     }
974     {
975       \tl_set:Nn \l__zrefclever_current_language_tl { english }
976       \tl_set:Nn \l__zrefclever_main_language_tl { english }
977     }
978   }

```

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.

```

979   \tl_set:Nn \l__zrefclever_ref_language_tl
980   { \l__zrefclever_main_language_tl }
981 }
982 \keys_define:nn { zref-clever / reference }
983 {
984   lang .code:n =
985   {
986     \AddToHook { begindocument }
987     {
988       \str_case:nnF {#1}
989       {
990         { main }
991         {
992           \tl_set:Nn \l__zrefclever_ref_language_tl
993           { \l__zrefclever_main_language_tl }
994           \__zrefclever_provide_dictionary_verbosely:x
995           { \l__zrefclever_ref_language_tl }
996         }
997
998         { current }
999         {
1000          \tl_set:Nn \l__zrefclever_ref_language_tl

```

```

1001         { \l__zrefclever_current_language_tl }
1002         \__zrefclever_provide_dictionary_verbose:x
1003         { \l__zrefclever_ref_language_tl }
1004     }
1005 }
1006 {
1007     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1008     {
1009         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1010     }
1011     {
1012         \msg_warning:nnn { zref-clever }
1013         { unknown-language-opt } {#1}
1014         \tl_set:Nn \l__zrefclever_ref_language_tl
1015         { \l__zrefclever_main_language_tl }
1016     }
1017     \__zrefclever_provide_dictionary_verbose:x
1018     { \l__zrefclever_ref_language_tl }
1019 }
1020 }
1021 } ,
1022 lang .value_required:n = true ,
1023 }
1024 \AddToHook { begindocument / before }
1025 {
1026     \AddToHook { begindocument }
1027     {

```

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.

```

1028     \__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.

```

1029     \keys_define:nn { zref-clever / reference }
1030     {
1031         lang .code:n =
1032         {
1033             \str_case:nnF {#1}
1034             {
1035                 { main }
1036                 {
1037                     \tl_set:Nn \l__zrefclever_ref_language_tl
1038                     { \l__zrefclever_main_language_tl }
1039                     \__zrefclever_provide_dictionary:x
1040                     { \l__zrefclever_ref_language_tl }
1041                 }
1042
1043                 { current }
1044                 {
1045                     \tl_set:Nn \l__zrefclever_ref_language_tl
1046                     { \l__zrefclever_current_language_tl }

```

```

1047         \__zrefclever_provide_dictionary:x
1048         { \l__zrefclever_ref_language_tl }
1049     }
1050 }
1051 {
1052     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1053     {
1054         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1055     }
1056     {
1057         \msg_warning:nnn { zref-clever }
1058         { unknown-language-opt } {#1}
1059         \tl_set:Nn \l__zrefclever_ref_language_tl
1060         { \l__zrefclever_main_language_tl }
1061     }
1062     \__zrefclever_provide_dictionary:x
1063     { \l__zrefclever_ref_language_tl }
1064 }
1065 } ,
1066 lang .value_required:n = true ,
1067 }
1068 }
1069 }

```

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.

```

1070 \tl_new:N \l__zrefclever_ref_decl_case_tl
1071 \keys_define:nn { zref-clever / reference }
1072 {
1073     d .code:n =
1074     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1075 }
1076 \AddToHook { begindocument }
1077 {
1078     \keys_define:nn { zref-clever / reference }
1079     {

```

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

```

1080         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1081         d .value_required:n = true ,
1082     }
1083 }

```

nudge & Co. options

```

1084 \bool_new:N \l__zrefclever_nudge_enabled_bool

```



```

1085 \bool_new:N \l__zrefclever_nudge_multitype_bool
1086 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1087 \bool_new:N \l__zrefclever_nudge_singular_bool
1088 \bool_new:N \l__zrefclever_nudge_gender_bool
1089 \tl_new:N \l__zrefclever_ref_gender_tl
1090 \keys_define:nn { zref-clever / reference }
1091 {
1092   nudge .choice: ,
1093   nudge / true .code:n =
1094     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1095   nudge / false .code:n =
1096     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1097   nudge / obeydraft .code:n =
1098     {
1099       \ifdraft
1100         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1101         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1102       } ,
1103   nudge / obeyfinal .code:n =
1104     {
1105       \ifoptionfinal
1106         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1107         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1108       } ,
1109   nudge .initial:n = false ,
1110   nudge .default:n = true ,
1111   nonudge .meta:n = { nudge = false } ,
1112   nonudge .value_forbidden:n = true ,
1113   nudgeif .code:n =
1114     {
1115       \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1116       \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1117       \bool_set_false:N \l__zrefclever_nudge_gender_bool
1118       \clist_map_inline:nn {#1}
1119       {
1120         \str_case:nnF {##1}
1121         {
1122           { multitype }
1123           { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1124           { comptosing }
1125           { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1126           { gender }
1127           { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1128           { all }
1129         }
1130         {
1131           \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1132           \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1133           \bool_set_true:N \l__zrefclever_nudge_gender_bool
1134         }
1135       }
1136       {
1137         \msg_warning:nnn { zref-clever }
1138         { nudgeif-unknown-value } {##1}
1139       }
1140     }

```

```

1139     }
1140   } ,
1141   nudgeif .value_required:n = true ,
1142   nudgeif .initial:n = all ,
1143   sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1144   sg .initial:n = false ,
1145   sg .default:n = true ,
1146   g .code:n =
1147     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1148   }
1149 \AddToHook { begindocument }
1150 {
1151   \keys_define:nn { zref-clever / reference }
1152   {

```

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

```

1153     g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1154     g .value_required:n = true ,
1155   }
1156 }

```

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 `\zceref` 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).

```

1157 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1158 \keys_define:nn { zref-clever / reference }
1159 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

1160 \keys_define:nn { zref-clever / reference }
1161 {
1162   titleref .code:n = { \RequirePackage { zref-titleref } } ,
1163   titleref .value_forbidden:n = true ,
1164 }
1165 \AddToHook { begindocument }
1166 {
1167   \keys_define:nn { zref-clever / reference }
1168   {
1169     titleref .code:n =
1170       { \msg_warning:nnn { zref-clever } { titleref-preamble-only } }
1171   }
1172 }

```

note option

```

1173 \tl_new:N \l__zrefclever_zceref_note_tl
1174 \keys_define:nn { zref-clever / reference }
1175 {
1176   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
1177   note .value_required:n = true ,

```

```
1178 }
```

check option

Integration with zref-check.

```
1179 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1180 \bool_new:N \l__zrefclever_zcref_with_check_bool
1181 \keys_define:nn { zref-clever / reference }
1182 {
1183   check .code:n = { \RequirePackage { zref-check } } ,
1184   check .value_forbidden:n = true ,
1185 }
1186 \AddToHook { begindocument }
1187 {
1188   \@ifpackageloaded { zref-check }
1189   {
1190     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1191     \keys_define:nn { zref-clever / reference }
1192     {
1193       check .code:n =
1194       {
1195         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
1196         \keys_set:nn { zref-check / zcheck } {#1}
1197       } ,
1198       check .value_required:n = true ,
1199     }
1200   }
1201   {
1202     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1203     \keys_define:nn { zref-clever / reference }
1204     {
1205       check .value_forbidden:n = false ,
1206       check .code:n =
1207       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1208     }
1209   }
1210 }
```

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`.

```
1211 \prop_new:N \l__zrefclever_counter_type_prop
1212 \keys_define:nn { zref-clever / label }
1213 {
1214   countertype .code:n =
1215   {
1216     \keyval_parse:nnn
1217     {
1218       \msg_warning:nnnn { zref-clever }
1219       { key-requires-value } { countertype }
1220     }
1221   }
```

```

1221     {
1222         \__zrefclever_prop_put_non_empty:Nnn
1223         \l__zrefclever_counter_type_prop
1224     }
1225     {#1}
1226 },
1227 countertype .value_required:n = true ,
1228 countertype .initial:n =
1229 {
1230     subsection      = section ,
1231     subsubsection    = section ,
1232     subparagraph     = paragraph ,
1233     enumi            = item ,
1234     enumii           = item ,
1235     enumiii          = item ,
1236     enumiv           = item ,
1237     mpfootnote       = footnote ,
1238 } ,
1239 }

```

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.

```

1240 \seq_new:N \l__zrefclever_counter_resetters_seq
1241 \keys_define:nn { zref-clever / label }
1242 {
1243     counterresetters .code:n =
1244     {
1245         \clist_map_inline:nn {#1}
1246         {
1247             \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1248             {
1249                 \seq_put_right:Nn
1250                 \l__zrefclever_counter_resetters_seq {##1}
1251             }
1252         }
1253     } ,
1254     counterresetters .initial:n =
1255     {
1256         part ,
1257         chapter ,
1258         section ,
1259         subsection ,
1260         subsubsection ,
1261         paragraph ,
1262         subparagraph ,
1263     },

```

```

1264     counterresetters .value_required:n = true ,
1265 }

```

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`.

```

1266 \prop_new:N \l__zrefclever_counter_resetby_prop
1267 \keys_define:nn { zref-clever / label }
1268 {
1269     counterresetby .code:n =
1270     {
1271         \keyval_parse:nnn
1272         {
1273             \msg_warning:nnn { zref-clever }
1274             { key-requires-value } { counterresetby }
1275         }
1276         {
1277             \__zrefclever_prop_put_non_empty:Nnn
1278             \l__zrefclever_counter_resetby_prop
1279         }
1280         {#1}
1281     } ,
1282     counterresetby .value_required:n = true ,
1283     counterresetby .initial:n =
1284     {

```

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.

```

1285     enumii = enumi ,
1286     enumiii = enumii ,
1287     enumiv = enumiii ,
1288 } ,
1289 }

```

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.

```

1290 \tl_new:N \l__zrefclever_current_counter_tl
1291 \keys_define:nn { zref-clever / label }
1292 {
1293     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1294     currentcounter .value_required:n = true ,
1295     currentcounter .initial:n = \@currentcounter ,
1296 }

```

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.

```
1297 \prop_new:N \l__zrefclever_ref_options_prop
1298 \seq_map_inline:Nn
1299   \c__zrefclever_ref_options_reference_seq
1300   {
1301     \keys_define:nn { zref-clever / reference }
1302     {
1303       #1 .default:V = \c_novalue_tl ,
1304       #1 .code:n =
1305       {
1306         \tl_if_novalue:nTF {##1}
1307         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1308         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1309       } ,
1310     }
1311   }
```

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.

```
1312 \keys_define:nn { }
1313   {
1314     zref-clever / zcsetup .inherit:n =
1315     {
1316       zref-clever / label ,
1317       zref-clever / reference ,
1318     }
1319   }
```

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

```
1320 \ProcessKeysOptions { zref-clever / zcsetup }
```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

1321 \NewDocumentCommand \zcsetup { m }
1322 { \_zrefclever_zcsetup:n {#1} }

(End definition for \zcsetup.)

```

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

```

\_zrefclever_zcsetup:n{<options>}

1323 \cs_new_protected:Npn \_zrefclever_zcsetup:n #1
1324 { \keys_set:nn { zref-clever / zcsetup } {#1} }
1325 \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>}

1326 \NewDocumentCommand \zcRefTypeSetup { m m }
1327 {
1328   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1329   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1330   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1331   \keys_set:nn { zref-clever / typesetup } {#2}
1332 }

(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>.

```

1333 \seq_map_inline:Nn
1334   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1335   {
1336     \keys_define:nn { zref-clever / typesetup }
1337     {
1338       #1 .code:n =
1339       {
1340         \msg_warning:nnn { zref-clever }
1341         { option-not-type-specific } {#1}
1342       } ,
1343     }
1344   }
1345 \seq_map_inline:Nn
1346   \c__zrefclever_ref_options_typesetup_seq
1347   {
1348     \keys_define:nn { zref-clever / typesetup }
1349     {
1350       #1 .default:V = \c_novalue_tl ,
1351       #1 .code:n =
1352       {
1353         \tl_if_novalue:nTF {##1}
1354         {
1355           \prop_remove:cn
1356           {
1357             l__zrefclever_type_
1358             \l__zrefclever_setup_type_tl _options_prop
1359           }
1360           {#1}
1361         }
1362         {
1363           \prop_put:cnn
1364           {
1365             l__zrefclever_type_
1366             \l__zrefclever_setup_type_tl _options_prop
1367           }
1368           {#1} {##1}
1369         }
1370       } ,
1371     }
1372   }

```

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>}
1373 \NewDocumentCommand \zcLanguageSetup { m m }
1374 {
1375   \group_begin:
1376   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1377   \l__zrefclever_dict_language_tl
1378   {
1379     \tl_clear:N \l__zrefclever_setup_type_tl
1380     \exp_args:NNx \seq_set_from_clist:Nn
1381     \l__zrefclever_dict_declension_seq
1382     {
1383       \prop_item:cn
1384       {
1385         g__zrefclever_dict_
1386         \l__zrefclever_dict_language_tl _prop
1387       }
1388       { declension }
1389     }
1390     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1391     { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
1392     {
1393       \seq_get_left:NN \l__zrefclever_dict_declension_seq
1394       \l__zrefclever_dict_decl_case_tl
1395     }
1396     \exp_args:NNx \seq_set_from_clist:Nn
1397     \l__zrefclever_dict_gender_seq
1398     {
1399       \prop_item:cn
1400       {
1401         g__zrefclever_dict_
1402         \l__zrefclever_dict_language_tl _prop
1403       }
1404       { gender }
1405     }
1406     \keys_set:nn { zref-clever / langsetup } {#2}
1407   }
1408   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1409   \group_end:
1410 }
1411 \@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>}
1412 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1413 {
1414   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }

```

```

1415     { type- #2 - #3 } {#4}
1416   }
1417   \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn , VVxn }
1418   \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
1419   {
1420     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1421     { default- #2 } {#3}
1422   }
1423   \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`.

```

1424 \keys_define:nn { zref-clever / langsetup }
1425 {
1426   type .code:n =
1427   {
1428     \tl_if_empty:NTF {#1}
1429     { \tl_clear:N \l__zrefclever_setup_type_tl }
1430     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1431   } ,
1432   case .code:n =
1433   {
1434     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1435     {
1436       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
1437       { \l__zrefclever_dict_language_tl } {#1}
1438     }
1439     {
1440       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
1441       { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
1442       {
1443         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
1444         {#1} { \l__zrefclever_dict_language_tl }
1445         \seq_get_left:NN \l__zrefclever_dict_declension_seq
1446         \l__zrefclever_dict_decl_case_tl
1447       }
1448     }
1449   } ,
1450   case .value_required:n = true ,
1451   gender .code:n =
1452   {
1453     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
1454     {
1455       \msg_warning:nxxx { zref-clever } { language-no-gender }
1456       { \l__zrefclever_dict_language_tl } { gender } {#1}
1457     }
1458     {
1459       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1460       {
1461         \msg_warning:nnn { zref-clever }
1462         { option-only-type-specific } { gender }
1463       }

```

```

1464         {
1465             \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
1466             {
1467                 \__zrefclever_declare_type_transl:VWnn
1468                 \l__zrefclever_dict_language_tl
1469                 \l__zrefclever_setup_type_tl
1470                 { gender } {#1}
1471             }
1472             {
1473                 \msg_warning:nxxx { zref-clever } { gender-not-declared }
1474                 { \l__zrefclever_dict_language_tl } {#1}
1475             }
1476         }
1477     }
1478 },
1479 gender .value_required:n = true ,
1480 }
1481 \seq_map_inline:Nn
1482 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1483 {
1484     \keys_define:nn { zref-clever / langsetup }
1485     {
1486         #1 .value_required:n = true ,
1487         #1 .code:n =
1488         {
1489             \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1490             {
1491                 \__zrefclever_declare_default_transl:Vnn
1492                 \l__zrefclever_dict_language_tl
1493                 {#1} {##1}
1494             }
1495             {
1496                 \msg_warning:nnn { zref-clever }
1497                 { option-not-type-specific } {#1}
1498             }
1499         } ,
1500     }
1501 }
1502 \seq_map_inline:Nn
1503 \c__zrefclever_ref_options_possibly_type_specific_seq
1504 {
1505     \keys_define:nn { zref-clever / langsetup }
1506     {
1507         #1 .value_required:n = true ,
1508         #1 .code:n =
1509         {
1510             \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1511             {
1512                 \__zrefclever_declare_default_transl:Vnn
1513                 \l__zrefclever_dict_language_tl
1514                 {#1} {##1}
1515             }
1516             {
1517                 \__zrefclever_declare_type_transl:VWnn

```

```

1518         \l__zrefclever_dict_language_tl
1519         \l__zrefclever_setup_type_tl
1520         {#1} {##1}
1521     }
1522 } ,
1523 }
1524 }
1525 \seq_map_inline:Nn
1526 \c__zrefclever_ref_options_type_names_seq
1527 {
1528     \keys_define:nn { zref-clever / langsetup }
1529     {
1530         #1 .value_required:n = true ,
1531         #1 .code:n =
1532         {
1533             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1534             {
1535                 \msg_warning:nnn { zref-clever }
1536                 { option-only-type-specific } {#1}
1537             }
1538             {
1539                 \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
1540                 {
1541                     \__zrefclever_declare_type_transl:VVnn
1542                     \l__zrefclever_dict_language_tl
1543                     \l__zrefclever_setup_type_tl
1544                     {#1} {##1}
1545                 }
1546                 {
1547                     \__zrefclever_declare_type_transl:VVxn
1548                     \l__zrefclever_dict_language_tl
1549                     \l__zrefclever_setup_type_tl
1550                     { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
1551                 }
1552             }
1553         } ,
1554     }
1555 }

```

6 User interface

6.1 \zcref

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

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

```

1556 \NewDocumentCommand \zcref { s O { } m }
1557 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for `\zcref`.)

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

```

    \__zrefclever_zcref:nnnn {\<labels>} {\<*>} {\<options>}}
1558 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1559 {
1560     \group_begin:
Set options.
1561     \keys_set:nn { zref-clever / reference } {#3}
Store arguments values.
1562     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1563     \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.
1564     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
Process \zcDeclareLanguage options.
1565     \__zrefclever_process_language_options:
Integration with zref-check.
1566     \bool_lazy_and:nnT
1567     { \l__zrefclever_zrefcheck_available_bool }
1568     { \l__zrefclever_zcref_with_check_bool }
1569     { \zrefcheck_zcref_beg_label: }
Sort the labels.
1570     \bool_lazy_or:nnT
1571     { \l__zrefclever_typeset_sort_bool }
1572     { \l__zrefclever_typeset_range_bool }
1573     { \__zrefclever_sort_labels: }
Typeset the references. Also, set the reference font, and group it, so that it does not leak
to the note.
1574     \group_begin:
1575     \l__zrefclever_ref_typeset_font_tl
1576     \__zrefclever_typeset_refs:
1577     \group_end:
Typeset note.
1578     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1579     {
1580         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1581         \l_tmpa_tl
1582         \l__zrefclever_zcref_note_tl
1583     }
Integration with zref-check.
1584     \bool_lazy_and:nnT
1585     { \l__zrefclever_zrefcheck_available_bool }
1586     { \l__zrefclever_zcref_with_check_bool }
1587     {
1588         \zrefcheck_zcref_end_label_maybe:
1589         \zrefcheck_zcref_run_checks_on_labels:n
1590         { \l__zrefclever_zcref_labels_seq }
1591     }

```

Integration with mathtools.

```

1592     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1593     {
1594         \__zrefclever_mathtools_showonlyrefs:n
1595         { \l__zrefclever_zcref_labels_seq }
1596     }
1597     \group_end:
1598 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1599 \seq_new:N \l__zrefclever_zcref_labels_seq
1600 \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}

```

```

1601 \NewDocumentCommand \zcpageref { s O { } m }
1602 {
1603     \IfBooleanTF {#1}
1604     { \zcref*{#2, ref = page} {#3} }
1605     { \zcref [ #2, ref = page] {#3} }
1606 }

```

(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
1607 \tl_new:N \l__zrefclever_label_type_a_tl
1608 \tl_new:N \l__zrefclever_label_type_b_tl
1609 \tl_new:N \l__zrefclever_label_enclval_a_tl
1610 \tl_new:N \l__zrefclever_label_enclval_b_tl
1611 \tl_new:N \l__zrefclever_label_extdoc_a_tl
1612 \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.

```
1613 \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.

```
1614 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
1615 \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.

```
1616 \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.

```
1617 \cs_new_protected:Npn \__zrefclever_sort_labels:
```

```
1618 {
```

Store label types sequence.

```
1619 \seq_clear:N \l__zrefclever_label_types_seq
```

```
1620 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
```

```
1621 {
```

```
1622 \seq_map_function:NN \l__zrefclever_zcref_labels_seq
```

```
1623 \__zrefclever_label_type_put_new_right:n
```

```
1624 }
```

Sort.

```
1625 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
```

```
1626 {
```

```
1627 \zref@ifrefundefined {##1}
```

```
1628 {
```

```
1629 \zref@ifrefundefined {##2}
```

```
1630 {
```

```
1631 % Neither label is defined.
```

```
1632 \sort_return_same:
```

```
1633 }
```

```
1634 {
```

```
1635 % The second label is defined, but the first isn't, leave the
```

```
1636 % undefined first (to be more visible).
```

```
1637 \sort_return_same:
```

```

1638     }
1639   }
1640   {
1641     \zref@ifrefundefined {##2}
1642     {
1643       % The first label is defined, but the second isn't, bring the
1644       % second forward.
1645       \sort_return_swapped:
1646     }
1647     {
1648       % The interesting case: both labels are defined. References
1649       % to the "default" property or to the "page" are quite
1650       % different with regard to sorting, so we branch them here to
1651       % specialized functions.
1652       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1653       { \__zrefclever_sort_page:nn {##1} {##2} }
1654       { \__zrefclever_sort_default:nn {##1} {##2} }
1655     }
1656   }
1657 }
1658 }

```

(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>}

1659 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1660 {
1661   \__zrefclever_def_extract:Nnnn
1662   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1663   \seq_if_in:NVF \l__zrefclever_label_types_seq
1664   \l__zrefclever_label_type_a_tl
1665   {
1666     \seq_put_right:NV \l__zrefclever_label_types_seq
1667     \l__zrefclever_label_type_a_tl
1668   }
1669 }

```

(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:.


```

1670 \__zrefclever_sort_default:nn {\label a}} {\label b}}
1671 {
1672   \__zrefclever_def_extract:Nnnn
1673   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1674   \__zrefclever_def_extract:Nnnn
1675   \l__zrefclever_label_type_b_tl {#2} {zc@type} {\c_empty_tl}
1676
1677   \bool_if:nTF
1678   {
1679     % The second label has a type, but the first doesn't, leave the
1680     % undefined first (to be more visible).
1681     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1682     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1683   }
1684   { \sort_return_same: }
1685   {
1686     \bool_if:nTF
1687     {
1688       % The first label has a type, but the second doesn't, bring the
1689       % second forward.
1690       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1691       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1692     }
1693     { \sort_return_swapped: }
1694     {
1695       \bool_if:nTF
1696       {
1697         % The interesting case: both labels have a type...
1698         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1699         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1700       }
1701       {
1702         \tl_if_eq:NNTF
1703         \l__zrefclever_label_type_a_tl
1704         \l__zrefclever_label_type_b_tl
1705         % ...and it's the same type.
1706         { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1707         % ...and they are different types.
1708         { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1709       }
1710     }
1711     % Neither label has a type. We can't do much of meaningful
1712     % here, but if it's the same counter, compare it.
1713     \exp_args:Nxx \tl_if_eq:nnTF
1714     { \__zrefclever_extract_unexp:nnn {#1} {zc@counter} { } }
1715     { \__zrefclever_extract_unexp:nnn {#2} {zc@counter} { } }
1716     {
1717       \int_compare:nNnTF
1718       { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
1719       >
1720       { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
1721       { \sort_return_swapped: }

```

```

1722         { \sort_return_same: }
1723     }
1724     { \sort_return_same: }
1725 }
1726 }
1727 }
1728 }

```

(End definition for _zrefclever_sort_default:nn.)

```

\_zrefclever_sort_default_same_type:nn      \_zrefclever_sort_default_same_type:nn {\label a}\{\label b}\}
1729 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1730 {
1731   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_a_tl
1732   {#1} { zc@enclval } { \c_empty_tl }
1733   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1734   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_b_tl
1735   {#2} { zc@enclval } { \c_empty_tl }
1736   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1737   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
1738   {#1} { externaldocument } { \c_empty_tl }
1739   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
1740   {#2} { externaldocument } { \c_empty_tl }
1741
1742   \bool_set_false:N \l__zrefclever_sort_decided_bool
1743
1744   % First we check if there's any "external document" difference (coming
1745   % from 'zref-xr') and, if so, sort based on that.
1746   \tl_if_eq:NNF
1747     \l__zrefclever_label_extdoc_a_tl
1748     \l__zrefclever_label_extdoc_b_tl
1749   {
1750     \bool_if:nTF
1751     {
1752       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1753       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1754     }
1755     {
1756       \bool_set_true:N \l__zrefclever_sort_decided_bool
1757       \sort_return_same:
1758     }
1759     {
1760       \bool_if:nTF
1761       {
1762         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1763         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1764       }
1765       {
1766         \bool_set_true:N \l__zrefclever_sort_decided_bool
1767         \sort_return_swapped:
1768       }
1769       {
1770         \bool_set_true:N \l__zrefclever_sort_decided_bool
1771         % Two different "external documents": last resort, sort by the

```

```

1772             % document name itself.
1773             \str_compare:eNeTF
1774             { \l__zrefclever_label_extdoc_b_tl } <
1775             { \l__zrefclever_label_extdoc_a_tl }
1776             { \sort_return_swapped: }
1777             { \sort_return_same:    }
1778         }
1779     }
1780 }
1781
1782 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1783 {
1784     \bool_if:nTF
1785     {
1786         % Both are empty: neither label has any (further) "enclosing
1787         % counters" (left).
1788         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1789         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1790     }
1791     {
1792         \bool_set_true:N \l__zrefclever_sort_decided_bool
1793         \int_compare:nNnTF
1794         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1795         >
1796         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1797         { \sort_return_swapped: }
1798         { \sort_return_same:    }
1799     }
1800     {
1801         \bool_if:nTF
1802         {
1803             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1804             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1805         }
1806         {
1807             \bool_set_true:N \l__zrefclever_sort_decided_bool
1808             \int_compare:nNnTF
1809             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
1810             >
1811             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1812             { \sort_return_swapped: }
1813             { \sort_return_same:    }
1814         }
1815     }
1816     \bool_if:nTF
1817     {
1818         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1819         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1820     }
1821     {
1822         \bool_set_true:N \l__zrefclever_sort_decided_bool
1823         \int_compare:nNnTF
1824         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1825         <

```

```

1826         { \_zrefclever_extract:nnn {#2} { zc@cntval } { } }
1827         { \sort_return_same:    }
1828         { \sort_return_swapped: }
1829     }
1830     {
1831         % Neither is empty: we can compare the values of the
1832         % current enclosing counter in the loop, if they are
1833         % equal, we are still in the loop, if they are not, a
1834         % sorting decision can be made directly.
1835         \int_compare:nNnTF
1836             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1837             =
1838             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1839             {
1840                 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1841                     { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1842                 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1843                     { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1844             }
1845             {
1846                 \bool_set_true:N \l__zrefclever_sort_decided_bool
1847                 \int_compare:nNnTF
1848                     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1849                     >
1850                     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1851                     { \sort_return_swapped: }
1852                     { \sort_return_same:    }
1853             }
1854         }
1855     }
1856 }
1857 }
1858 }

```

(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)}
1859 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1860 {

```

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”.

```

1861     \int_zero:N \l__zrefclever_sort_prior_a_int
1862     \int_zero:N \l__zrefclever_sort_prior_b_int
1863     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1864     {
1865         \tl_if_eq:nnTF {##2} {{othertypes}}
1866         {
1867             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1868             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1869             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1870             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1871         }

```

```

1872     {
1873       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1874       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1875       {
1876         \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1877         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1878       }
1879     }
1880   }

```

Then do the actual sorting.

```

1881   \bool_if:nTF
1882   {
1883     \int_compare_p:nNn
1884     { \l__zrefclever_sort_prior_a_int } <
1885     { \l__zrefclever_sort_prior_b_int }
1886   }
1887   { \sort_return_same: }
1888   {
1889     \bool_if:nTF
1890     {
1891       \int_compare_p:nNn
1892       { \l__zrefclever_sort_prior_a_int } >
1893       { \l__zrefclever_sort_prior_b_int }
1894     }
1895     { \sort_return_swapped: }
1896     {
1897       % Sort priorities are equal: the type that occurs first in
1898       % ‘labels’, as given by the user, is kept (or brought) forward.
1899       \seq_map_inline:Nn \l__zrefclever_label_types_seq
1900       {
1901         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1902         { \seq_map_break:n { \sort_return_same: } }
1903         {
1904           \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##1}
1905           { \seq_map_break:n { \sort_return_swapped: } }
1906         }
1907       }
1908     }
1909   }
1910 }

```

(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)}

1911 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1912 {
1913   \int_compare:nNnTF

```

```

1914     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1915     >
1916     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1917     { \sort_return_swapped: }
1918     { \sort_return_same:    }
1919   }

```

(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
1920 \seq_new:N \l__zrefclever_typeset_labels_seq
1921 \bool_new:N \l__zrefclever_typeset_last_bool
1922 \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
1923 \int_new:N \l__zrefclever_type_count_int
1924 \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
1925 \tl_new:N \l__zrefclever_label_a_tl
1926 \tl_new:N \l__zrefclever_label_b_tl
1927 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1928 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1929 \tl_new:N \l__zrefclever_type_first_label_tl
```

1930 \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.

1931 \tl_new:N \l__zrefclever_type_name_tl
1932 \bool_new:N \l__zrefclever_name_in_link_bool
1933 \tl_new:N \l__zrefclever_name_format_tl
1934 \tl_new:N \l__zrefclever_name_format_fallback_tl
1935 \tl_new:N \l__zrefclever_type_name_gender_tl

(End definition for \l__zrefclever_type_name_tl and others.)

\l__zrefclever_range_count_int Auxiliary variables for __zrefclever_typeset_refs: range handling.

\l__zrefclever_range_same_count_int
1936 \int_new:N \l__zrefclever_range_count_int
1937 \int_new:N \l__zrefclever_range_same_count_int
1938 \tl_new:N \l__zrefclever_range_beg_label_tl
1939 \bool_new:N \l__zrefclever_next_maybe_range_bool
1940 \bool_new:N \l__zrefclever_next_is_same_bool

(End definition for \l__zrefclever_range_count_int and others.)

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

\l__zrefclever_tlistsep_tl
1941 \tl_new:N \l__zrefclever_tpairsep_tl
1942 \tl_new:N \l__zrefclever_tlistsep_tl
1943 \tl_new:N \l__zrefclever_tlastsep_tl
1944 \tl_new:N \l__zrefclever_namesep_tl
1945 \tl_new:N \l__zrefclever_pairsep_tl
1946 \tl_new:N \l__zrefclever_listsep_tl
1947 \tl_new:N \l__zrefclever_lastsep_tl
1948 \tl_new:N \l__zrefclever_rangesep_tl
1949 \tl_new:N \l__zrefclever_refpre_tl
1950 \tl_new:N \l__zrefclever_refpos_tl
1951 \tl_new:N \l__zrefclever_namefont_tl
1952 \tl_new:N \l__zrefclever_reffont_tl

(End definition for \l__zrefclever_tpairsep_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

1953 \cs_new_protected:Npn __zrefclever_typeset_refs:
1954 {
1955 \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1956 \l__zrefclever_zceref_labels_seq
1957 \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1958 \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1959 \tl_clear:N \l__zrefclever_type_first_label_tl
1960 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1961 \tl_clear:N \l__zrefclever_range_beg_label_tl
1962 \int_zero:N \l__zrefclever_label_count_int
1963 \int_zero:N \l__zrefclever_type_count_int
1964 \int_zero:N \l__zrefclever_range_count_int


```

1965 \int_zero:N \l__zrefclever_range_same_count_int
1966
1967 % Get type block options (not type-specific).
1968 \__zrefclever_get_ref_string:nN { tpairsep }
1969 \l__zrefclever_tpairsep_tl
1970 \__zrefclever_get_ref_string:nN { tlistsep }
1971 \l__zrefclever_tlistsep_tl
1972 \__zrefclever_get_ref_string:nN { tlastsep }
1973 \l__zrefclever_tlastsep_tl
1974
1975 % Process label stack.
1976 \bool_set_false:N \l__zrefclever_typeset_last_bool
1977 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1978 {
1979   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1980   \l__zrefclever_label_a_tl
1981   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1982   {
1983     \tl_clear:N \l__zrefclever_label_b_tl
1984     \bool_set_true:N \l__zrefclever_typeset_last_bool
1985   }
1986   {
1987     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1988     \l__zrefclever_label_b_tl
1989   }
1990
1991   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1992   {
1993     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1994     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1995   }
1996   {
1997     \__zrefclever_def_extract:NVnn \l__zrefclever_label_type_a_tl
1998     \l__zrefclever_label_a_tl { zc@type } { \c_empty_tl }
1999     \__zrefclever_def_extract:NVnn \l__zrefclever_label_type_b_tl
2000     \l__zrefclever_label_b_tl { zc@type } { \c_empty_tl }
2001   }
2002
2003   % First, we establish whether the "current label" (i.e. 'a') is the
2004   % last one of its type. This can happen because the "next label"
2005   % (i.e. 'b') is of a different type (or different definition status),
2006   % or because we are at the end of the list.
2007   \bool_if:NTF \l__zrefclever_typeset_last_bool
2008   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2009   {
2010     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2011     {
2012       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2013       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2014       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2015     }
2016     {
2017       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2018       { \bool_set_true:N \l__zrefclever_last_of_type_bool }

```

```

2019 {
2020     % Neither is undefined, we must check the types.
2021     \bool_if:nTF
2022     {
2023         % Both empty: same "type".
2024         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
2025         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2026     }
2027     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2028     {
2029         \bool_if:nTF
2030         {
2031             % Neither empty: compare types.
2032             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
2033             &&
2034             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
2035         }
2036         {
2037             \tl_if_eq:NNTF
2038             \l__zrefclever_label_type_a_tl
2039             \l__zrefclever_label_type_b_tl
2040             {
2041                 \bool_set_false:N
2042                 \l__zrefclever_last_of_type_bool
2043             }
2044             {
2045                 \bool_set_true:N
2046                 \l__zrefclever_last_of_type_bool
2047             }
2048         }
2049         % One empty, the other not: different "types".
2050         {
2051             \bool_set_true:N
2052             \l__zrefclever_last_of_type_bool
2053         }
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 % Get type-specific separators, refpre/pos and font options, once per
2072 % type.

```

```

2073 \int_compare:nNt { \l__zrefclever_label_count_int } = { 0 }
2074 {
2075   \__zrefclever_get_ref_string:nN { namesep }
2076   \l__zrefclever_namesep_tl
2077   \__zrefclever_get_ref_string:nN { rangesep }
2078   \l__zrefclever_rangesep_tl
2079   \__zrefclever_get_ref_string:nN { pairsep }
2080   \l__zrefclever_pairsep_tl
2081   \__zrefclever_get_ref_string:nN { listsep }
2082   \l__zrefclever_listsep_tl
2083   \__zrefclever_get_ref_string:nN { lastsep }
2084   \l__zrefclever_lastsep_tl
2085   \__zrefclever_get_ref_string:nN { refpre }
2086   \l__zrefclever_refpre_tl
2087   \__zrefclever_get_ref_string:nN { refpos }
2088   \l__zrefclever_refpos_tl
2089   \__zrefclever_get_ref_font:nN { namefont }
2090   \l__zrefclever_namefont_tl
2091   \__zrefclever_get_ref_font:nN { reffont }
2092   \l__zrefclever_reffont_tl
2093 }
2094
2095 % Here we send this to a couple of auxiliary functions.
2096 \bool_if:NTF \l__zrefclever_last_of_type_bool
2097 { % There exists no next label of the same type as the current.
2098   { \__zrefclever_typeset_refs_last_of_type: }
2099   % There exists a next label of the same type as the current.
2100   { \__zrefclever_typeset_refs_not_last_of_type: }
2101 }
2102 }

```

(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.

```

2103 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
2104 {
2105   % Process the current label to the current queue.
2106   \int_case:nnF { \l__zrefclever_label_count_int }
2107   {
2108     % It is the last label of its type, but also the first one, and that's
2109     % what matters here: just store it.
2110     { 0 }
2111     {
2112       \tl_set:NV \l__zrefclever_type_first_label_tl
2113       \l__zrefclever_label_a_tl

```

```

2114 \tl_set:NV \l__zrefclever_type_first_label_type_tl
2115 \l__zrefclever_label_type_a_tl
2116 }
2117
2118 % The last is the second: we have a pair (if not repeated).
2119 { 1 }
2120 {
2121 \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
2122 {
2123 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2124 {
2125 \exp_not:V \l__zrefclever_pairsep_tl
2126 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2127 }
2128 }
2129 }
2130 }
2131 % Last is third or more of its type: without repetition, we'd have the
2132 % last element on a list, but control for possible repetition.
2133 {
2134 \int_case:nnF { \l__zrefclever_range_count_int }
2135 {
2136 % There was no range going on.
2137 { 0 }
2138 {
2139 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2140 {
2141 \exp_not:V \l__zrefclever_lastsep_tl
2142 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2143 }
2144 }
2145 % Last in the range is also the second in it.
2146 { 1 }
2147 {
2148 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2149 {
2150 % We know 'range_beg_label' is not empty, since this is the
2151 % second element in the range, but the third or more in the
2152 % type list.
2153 \exp_not:V \l__zrefclever_listsep_tl
2154 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2155 \int_compare:nNnF
2156 { \l__zrefclever_range_same_count_int } = { 1 }
2157 {
2158 \exp_not:V \l__zrefclever_lastsep_tl
2159 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2160 }
2161 }
2162 }
2163 }
2164 % Last in the range is third or more in it.
2165 {
2166 \int_case:nnF
2167 {

```

```

2168 \l__zrefclever_range_count_int -
2169 \l__zrefclever_range_same_count_int
2170 }
2171 {
2172 % Repetition, not a range.
2173 { 0 }
2174 {
2175 % If 'range_beg_label' is empty, it means it was also the
2176 % first of the type, and hence was already handled.
2177 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2178 {
2179 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2180 {
2181 \exp_not:V \l__zrefclever_lastsep_tl
2182 \__zrefclever_get_ref:V
2183 \l__zrefclever_range_beg_label_tl
2184 }
2185 }
2186 }
2187 % A 'range', but with no skipped value, treat as list.
2188 { 1 }
2189 {
2190 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2191 {
2192 % Ditto.
2193 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2194 {
2195 \exp_not:V \l__zrefclever_listsep_tl
2196 \__zrefclever_get_ref:V
2197 \l__zrefclever_range_beg_label_tl
2198 }
2199 \exp_not:V \l__zrefclever_lastsep_tl
2200 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2201 }
2202 }
2203 }
2204 {
2205 % An actual range.
2206 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2207 {
2208 % Ditto.
2209 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2210 {
2211 \exp_not:V \l__zrefclever_lastsep_tl
2212 \__zrefclever_get_ref:V
2213 \l__zrefclever_range_beg_label_tl
2214 }
2215 \exp_not:V \l__zrefclever_rangeseq_tl
2216 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2217 }
2218 }
2219 }
2220 }
2221

```

```

2222 % Handle "range" option. The idea is simple: if the queue is not empty,
2223 % we replace it with the end of the range (or pair). We can still
2224 % retrieve the end of the range from 'label_a' since we know to be
2225 % processing the last label of its type at this point.
2226 \bool_if:NT \l__zrefclever_typeset_range_bool
2227 {
2228   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2229   {
2230     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2231     { }
2232     {
2233       \msg_warning:nxx { zref-clever } { single-element-range }
2234       { \l__zrefclever_type_first_label_type_tl }
2235     }
2236   }
2237   {
2238     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2239     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2240     { }
2241     {
2242       \__zrefclever_labels_in_sequence:nn
2243       { \l__zrefclever_type_first_label_tl }
2244       { \l__zrefclever_label_a_tl }
2245     }
2246     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2247     {
2248       \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2249       { \exp_not:V \l__zrefclever_pairsep_tl }
2250       { \exp_not:V \l__zrefclever_rangesep_tl }
2251       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2252     }
2253   }
2254 }
2255
2256 % Now that the type block is finished, we can add the name and the first
2257 % ref to the queue. Also, if "typeset" option is not "both", handle it
2258 % here as well.
2259 \__zrefclever_type_name_setup:
2260 \bool_if:nTF
2261 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2262 {
2263   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2264   { \__zrefclever_get_ref_first: }
2265 }
2266 {
2267   \bool_if:NTF \l__zrefclever_typeset_ref_bool
2268   {
2269     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2270     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2271   }
2272   {
2273     \bool_if:NTF \l__zrefclever_typeset_name_bool
2274     {
2275       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

2276         {
2277             \bool_if:NTF \l__zrefclever_name_in_link_bool
2278             {
2279                 \exp_not:N \group_begin:
2280                 \exp_not:V \l__zrefclever_namefont_tl
2281                 % It's two '@s', but escaped for DocStrip.
2282                 \exp_not:N \hyper@@link
2283                 {
2284                     \__zrefclever_extract_url_unexp:V
2285                     \l__zrefclever_type_first_label_tl
2286                 }
2287                 {
2288                     \__zrefclever_extract_unexp:Vnn
2289                     \l__zrefclever_type_first_label_tl
2290                     { anchor } { }
2291                 }
2292                 { \exp_not:V \l__zrefclever_type_name_tl }
2293                 \exp_not:N \group_end:
2294             }
2295             {
2296                 \exp_not:N \group_begin:
2297                 \exp_not:V \l__zrefclever_namefont_tl
2298                 \exp_not:V \l__zrefclever_type_name_tl
2299                 \exp_not:N \group_end:
2300             }
2301         }
2302     }
2303     {
2304         % Logically, this case would correspond to "typeset=none", but
2305         % it should not occur, given that the options are set up to
2306         % typeset either "ref" or "name". Still, leave here a
2307         % sensible fallback, equal to the behavior of "both".
2308         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2309         { \__zrefclever_get_ref_first: }
2310     }
2311 }
2312 }
2313
2314 % Typeset the previous type block, if there is one.
2315 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
2316 {
2317     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
2318     { \l__zrefclever_tlistsep_tl }
2319     \l__zrefclever_typeset_queue_prev_tl
2320 }
2321
2322 % Wrap up loop, or prepare for next iteration.
2323 \bool_if:NTF \l__zrefclever_typeset_last_bool
2324 {
2325     % We are finishing, typeset the current queue.
2326     \int_case:nnF { \l__zrefclever_type_count_int }
2327     {
2328         % Single type.
2329         { 0 }

```

```

2330         { \l__zrefclever_typeset_queue_curr_tl }
2331         % Pair of types.
2332         { 1 }
2333         {
2334             \l__zrefclever_tpairsep_tl
2335             \l__zrefclever_typeset_queue_curr_tl
2336         }
2337     }
2338     {
2339         % Last in list of types.
2340         \l__zrefclever_tlastsep_tl
2341         \l__zrefclever_typeset_queue_curr_tl
2342     }
2343     % And nudge in case of multitype reference.
2344     \bool_lazy_all:nT
2345     {
2346         { \l__zrefclever_nudge_enabled_bool }
2347         { \l__zrefclever_nudge_multitype_bool }
2348         { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 1 } }
2349     }
2350     { \msg_warning:nn { zref-clever } { nudge-multitype } }
2351 }
2352 {
2353     % There are further labels, set variables for next iteration.
2354     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2355         \l__zrefclever_typeset_queue_curr_tl
2356     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2357     \tl_clear:N \l__zrefclever_type_first_label_tl
2358     \tl_clear:N \l__zrefclever_type_first_label_type_tl
2359     \tl_clear:N \l__zrefclever_range_beg_label_tl
2360     \int_zero:N \l__zrefclever_label_count_int
2361     \int_incr:N \l__zrefclever_type_count_int
2362     \int_zero:N \l__zrefclever_range_count_int
2363     \int_zero:N \l__zrefclever_range_same_count_int
2364 }
2365 }

```

(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.

```

2366 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
2367 {
2368     % Signal if next label may form a range with the current one (only
2369     % considered if compression is enabled in the first place).
2370     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2371     \bool_set_false:N \l__zrefclever_next_is_same_bool
2372     \bool_if:NT \l__zrefclever_typeset_compress_bool
2373     {
2374         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2375         { }
2376         {
2377             \_zrefclever_labels_in_sequence:nn
2378             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2379         }

```



```

2380     }
2381
2382 % Process the current label to the current queue.
2383 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2384 {
2385     % Current label is the first of its type (also not the last, but it
2386     % doesn't matter here): just store the label.
2387     \tl_set:NV \l__zrefclever_type_first_label_tl
2388         \l__zrefclever_label_a_tl
2389     \tl_set:NV \l__zrefclever_type_first_label_type_tl
2390         \l__zrefclever_label_type_a_tl
2391
2392     % If the next label may be part of a range, we set 'range_beg_label'
2393     % to "empty" (we deal with it as the "first", and must do it there, to
2394     % handle hyperlinking), but also step the range counters.
2395     \bool_if:NT \l__zrefclever_next_maybe_range_bool
2396     {
2397         \tl_clear:N \l__zrefclever_range_beg_label_tl
2398         \int_incr:N \l__zrefclever_range_count_int
2399         \bool_if:NT \l__zrefclever_next_is_same_bool
2400             { \int_incr:N \l__zrefclever_range_same_count_int }
2401     }
2402 }
2403 {
2404     % Current label is neither the first (nor the last) of its type.
2405     \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
2406     {
2407         % Starting, or continuing a range.
2408         \int_compare:nNnTF
2409             { \l__zrefclever_range_count_int } = { 0 }
2410         {
2411             % There was no range going, we are starting one.
2412             \tl_set:NV \l__zrefclever_range_beg_label_tl
2413                 \l__zrefclever_label_a_tl
2414             \int_incr:N \l__zrefclever_range_count_int
2415             \bool_if:NT \l__zrefclever_next_is_same_bool
2416                 { \int_incr:N \l__zrefclever_range_same_count_int }
2417         }
2418         {
2419             % Second or more in the range, but not the last.
2420             \int_incr:N \l__zrefclever_range_count_int
2421             \bool_if:NT \l__zrefclever_next_is_same_bool
2422                 { \int_incr:N \l__zrefclever_range_same_count_int }
2423         }
2424     }
2425 }
2426 % Next element is not in sequence: there was no range, or we are
2427 % closing one.
2428 \int_case:nnF { \l__zrefclever_range_count_int }
2429 {
2430     % There was no range going on.
2431     { 0 }
2432     {
2433         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

2434         {
2435             \exp_not:V \l__zrefclever_listsep_tl
2436             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2437         }
2438     }
2439     % Last is second in the range: if 'range_same_count' is also
2440     % '1', it's a repetition (drop it), otherwise, it's a "pair
2441     % within a list", treat as list.
2442     { 1 }
2443     {
2444         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2445         {
2446             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2447             {
2448                 \exp_not:V \l__zrefclever_listsep_tl
2449                 \__zrefclever_get_ref:V
2450                 \l__zrefclever_range_beg_label_tl
2451             }
2452             \int_compare:nNnF
2453             { \l__zrefclever_range_same_count_int } = { 1 }
2454             {
2455                 \exp_not:V \l__zrefclever_listsep_tl
2456                 \__zrefclever_get_ref:V
2457                 \l__zrefclever_label_a_tl
2458             }
2459         }
2460     }
2461 }
2462 {
2463     % Last is third or more in the range: if 'range_count' and
2464     % 'range_same_count' are the same, its a repetition (drop it),
2465     % if they differ by '1', its a list, if they differ by more,
2466     % it is a real range.
2467     \int_case:nnF
2468     {
2469         \l__zrefclever_range_count_int -
2470         \l__zrefclever_range_same_count_int
2471     }
2472     {
2473         { 0 }
2474         {
2475             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2476             {
2477                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2478                 {
2479                     \exp_not:V \l__zrefclever_listsep_tl
2480                     \__zrefclever_get_ref:V
2481                     \l__zrefclever_range_beg_label_tl
2482                 }
2483             }
2484         }
2485         { 1 }
2486         {
2487             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

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

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`.)

Aux functions

`__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:n` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nic requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` get called, as they must, in the context of x type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after

that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```

2522 \cs_new_protected:Npn \__zrefclever_ref_default:
2523   { \zref@default }
2524 \cs_new_protected:Npn \__zrefclever_name_default:
2525   { \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>}

2526 \cs_new:Npn \__zrefclever_get_ref:n #1
2527 {
2528   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2529   {
2530     \bool_if:nTF
2531     {
2532       \l__zrefclever_use_hyperref_bool &&
2533       ! \l__zrefclever_link_star_bool
2534     }
2535     {
2536       \bool_if:NF \l__zrefclever_preposinlink_bool
2537       { \exp_not:V \l__zrefclever_refpre_tl }
2538       % It’s two ‘@s’, but escaped for DocStrip.
2539       \exp_not:N \hyper@@link
2540       { \__zrefclever_extract_url_unexp:n {#1} }
2541       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2542       {
2543         \bool_if:NT \l__zrefclever_preposinlink_bool
2544         { \exp_not:V \l__zrefclever_refpre_tl }
2545         \exp_not:N \group_begin:
2546         \exp_not:V \l__zrefclever_reffont_tl
2547         \__zrefclever_extract_unexp:nvn {#1}
2548         { \l__zrefclever_ref_property_tl } { }
2549         \exp_not:N \group_end:
2550         \bool_if:NT \l__zrefclever_preposinlink_bool
2551         { \exp_not:V \l__zrefclever_refpos_tl }
2552       }
2553       \bool_if:NF \l__zrefclever_preposinlink_bool
2554       { \exp_not:V \l__zrefclever_refpos_tl }
2555     }
2556   }

```

```

2556     {
2557         \exp_not:V \l__zrefclever_refpre_tl
2558         \exp_not:N \group_begin:
2559         \exp_not:V \l__zrefclever_reffont_tl
2560         \__zrefclever_extract_unexp:nvn {#1}
2561         { \l__zrefclever_ref_property_tl } { }
2562         \exp_not:N \group_end:
2563         \exp_not:V \l__zrefclever_refpos_tl
2564     }
2565 }
2566 { \__zrefclever_ref_default: }
2567 }
2568 \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.

```

2569 \cs_new:Npn \__zrefclever_get_ref_first:
2570 {
2571     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2572     { \__zrefclever_ref_default: }
2573     {
2574         \bool_if:NTF \l__zrefclever_name_in_link_bool
2575         {
2576             \zref@ifrefcontainsprop
2577             { \l__zrefclever_type_first_label_tl }
2578             { \l__zrefclever_ref_property_tl }
2579             {
2580                 % It's two '@s', but escaped for DocStrip.
2581                 \exp_not:N \hyper@@link
2582                 {
2583                     \__zrefclever_extract_url_unexp:V
2584                     \l__zrefclever_type_first_label_tl
2585                 }
2586                 {
2587                     \__zrefclever_extract_unexp:Vnn
2588                     \l__zrefclever_type_first_label_tl { anchor } { }
2589                 }
2590             }
2591             {
2592                 \exp_not:N \group_begin:
2593                 \exp_not:V \l__zrefclever_namefont_tl
2594                 \exp_not:V \l__zrefclever_type_name_tl
2595                 \exp_not:N \group_end:
2596                 \exp_not:V \l__zrefclever_namesep_tl
2597                 \exp_not:V \l__zrefclever_refpre_tl
2598                 \exp_not:N \group_begin:
2599                 \exp_not:V \l__zrefclever_reffont_tl

```

```

2599         \__zrefclever_extract_unexp:Vvn
2600         \l__zrefclever_type_first_label_tl
2601         { l__zrefclever_ref_property_tl } { }
2602         \exp_not:N \group_end:
2603         \bool_if:NT \l__zrefclever_preposinlink_bool
2604         { \exp_not:V \l__zrefclever_refpos_tl }
2605     }
2606     \bool_if:NF \l__zrefclever_preposinlink_bool
2607     { \exp_not:V \l__zrefclever_refpos_tl }
2608 }
2609 {
2610     \exp_not:N \group_begin:
2611     \exp_not:V \l__zrefclever_namefont_tl
2612     \exp_not:V \l__zrefclever_type_name_tl
2613     \exp_not:N \group_end:
2614     \exp_not:V \l__zrefclever_namesep_tl
2615     \__zrefclever_ref_default:
2616 }
2617 }
2618 {
2619     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2620     {
2621         \__zrefclever_name_default:
2622         \exp_not:V \l__zrefclever_namesep_tl
2623     }
2624     {
2625         \exp_not:N \group_begin:
2626         \exp_not:V \l__zrefclever_namefont_tl
2627         \exp_not:V \l__zrefclever_type_name_tl
2628         \exp_not:N \group_end:
2629         \exp_not:V \l__zrefclever_namesep_tl
2630     }
2631     \zref@ifrefcontainsprop
2632     { \l__zrefclever_type_first_label_tl }
2633     { \l__zrefclever_ref_property_tl }
2634     {
2635         \bool_if:nTF
2636         {
2637             \l__zrefclever_use_hyperref_bool &&
2638             ! \l__zrefclever_link_star_bool
2639         }
2640         {
2641             \bool_if:NF \l__zrefclever_preposinlink_bool
2642             { \exp_not:V \l__zrefclever_refpre_tl }
2643             % It's two '@s', but escaped for DocStrip.
2644             \exp_not:N \hyper@@link
2645             {
2646                 \__zrefclever_extract_url_unexp:V
2647                 \l__zrefclever_type_first_label_tl
2648             }
2649             {
2650                 \__zrefclever_extract_unexp:Vnn
2651                 \l__zrefclever_type_first_label_tl { anchor } { }
2652             }

```

```

2653         {
2654             \bool_if:NT \l__zrefclever_preposinlink_bool
2655             { \exp_not:V \l__zrefclever_refpre_tl }
2656             \exp_not:N \group_begin:
2657             \exp_not:V \l__zrefclever_reffont_tl
2658             \__zrefclever_extract_unexp:Vvn
2659             \l__zrefclever_type_first_label_tl
2660             { \l__zrefclever_ref_property_tl } { }
2661             \exp_not:N \group_end:
2662             \bool_if:NT \l__zrefclever_preposinlink_bool
2663             { \exp_not:V \l__zrefclever_refpos_tl }
2664         }
2665         \bool_if:NF \l__zrefclever_preposinlink_bool
2666         { \exp_not:V \l__zrefclever_refpos_tl }
2667     }
2668     {
2669         \exp_not:V \l__zrefclever_refpre_tl
2670         \exp_not:N \group_begin:
2671         \exp_not:V \l__zrefclever_reffont_tl
2672         \__zrefclever_extract_unexp:Vvn
2673         \l__zrefclever_type_first_label_tl
2674         { \l__zrefclever_ref_property_tl } { }
2675         \exp_not:N \group_end:
2676         \exp_not:V \l__zrefclever_refpos_tl
2677     }
2678 }
2679 { \__zrefclever_ref_default: }
2680 }
2681 }
2682 }

```

(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.

```

2683 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2684 {
2685     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2686     { \tl_clear:N \l__zrefclever_type_name_tl }
2687     {
2688         \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl
2689         { \tl_clear:N \l__zrefclever_type_name_tl }
2690         {
2691             % Determine whether we should use capitalization, abbreviation,

```

```

2692 % and plural.
2693 \bool_lazy_or:nnTF
2694 { \l__zrefclever_capitalize_bool }
2695 {
2696   \l__zrefclever_capitalize_first_bool &&
2697   \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2698 }
2699 { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2700 { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2701 % If the queue is empty, we have a singular, otherwise, plural.
2702 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2703 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2704 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2705 \bool_lazy_and:nnTF
2706 { \l__zrefclever_abbrev_bool }
2707 {
2708   ! \int_compare_p:nNn
2709     { \l__zrefclever_type_count_int } = { 0 } ||
2710   ! \l__zrefclever_noabbrev_first_bool
2711 }
2712 {
2713   \tl_set:NV \l__zrefclever_name_format_fallback_tl
2714     \l__zrefclever_name_format_tl
2715   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2716 }
2717 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2718
2719 % Handle number and gender nudges.
2720 \bool_if:NT \l__zrefclever_nudge_enabled_bool
2721 {
2722   \bool_if:NTF \l__zrefclever_nudge_singular_bool
2723   {
2724     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2725     {
2726       \msg_warning:nnx { zref-clever }
2727       { nudge-plural-when-sg }
2728       { \l__zrefclever_type_first_label_type_tl }
2729     }
2730   }
2731   {
2732     \bool_lazy_all:nT
2733     {
2734       { \l__zrefclever_nudge_comptosing_bool }
2735       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2736     }
2737     {
2738       \int_compare_p:nNn
2739       { \l__zrefclever_label_count_int } > { 0 }
2740     }
2741   }
2742   {
2743     \msg_warning:nnx { zref-clever }
2744     { nudge-comptosing }
2745     { \l__zrefclever_type_first_label_type_tl }
2746   }
2747 }

```



```

2746     }
2747 \bool_lazy_and:nnT
2748 { \l__zrefclever_nudge_gender_bool }
2749 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
2750 {
2751   \__zrefclever_get_type_transl:xxnNF
2752   { \l__zrefclever_ref_language_tl }
2753   { \l__zrefclever_type_first_label_type_tl }
2754   { gender }
2755   \l__zrefclever_type_name_gender_tl
2756   { \tl_clear:N \l__zrefclever_type_name_gender_tl }
2757 \tl_if_eq:NNF
2758   \l__zrefclever_ref_gender_tl
2759   \l__zrefclever_type_name_gender_tl
2760   {
2761     \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
2762     {
2763       \msg_warning:nnxxx { zref-clever }
2764       { nudge-gender-not-declared-for-type }
2765       { \l__zrefclever_ref_gender_tl }
2766       { \l__zrefclever_type_first_label_type_tl }
2767       { \l__zrefclever_ref_language_tl }
2768     }
2769     {
2770       \msg_warning:nnxxxx { zref-clever }
2771       { nudge-gender-mismatch }
2772       { \l__zrefclever_type_first_label_type_tl }
2773       { \l__zrefclever_ref_gender_tl }
2774       { \l__zrefclever_type_name_gender_tl }
2775       { \l__zrefclever_ref_language_tl }
2776     }
2777   }
2778 }
2779 }
2780
2781 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2782 {
2783   \prop_get:cvNF
2784   {
2785     \l__zrefclever_type_
2786     \l__zrefclever_type_first_label_type_tl _options_prop
2787   }
2788   \l__zrefclever_name_format_tl
2789   \l__zrefclever_type_name_tl
2790   {
2791     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2792     {
2793       \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
2794       \tl_put_left:NV \l__zrefclever_name_format_tl
2795         \l__zrefclever_ref_decl_case_tl
2796     }
2797     \__zrefclever_get_type_transl:xxxNF
2798     { \l__zrefclever_ref_language_tl }
2799     { \l__zrefclever_type_first_label_type_tl }

```

```

2800         { \l__zrefclever_name_format_tl }
2801         \l__zrefclever_type_name_tl
2802         {
2803             \tl_clear:N \l__zrefclever_type_name_tl
2804             \msg_warning:nxxx { zref-clever } { missing-name }
2805             { \l__zrefclever_name_format_tl }
2806             { \l__zrefclever_type_first_label_type_tl }
2807         }
2808     }
2809 }
2810 {
2811     \prop_get:cVNF
2812     {
2813         l__zrefclever_type_
2814         \l__zrefclever_type_first_label_type_tl _options_prop
2815     }
2816     \l__zrefclever_name_format_tl
2817     \l__zrefclever_type_name_tl
2818     {
2819         \prop_get:cVNF
2820         {
2821             l__zrefclever_type_
2822             \l__zrefclever_type_first_label_type_tl _options_prop
2823         }
2824         \l__zrefclever_name_format_fallback_tl
2825         \l__zrefclever_type_name_tl
2826         {
2827             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2828             {
2829                 \tl_put_left:Nn
2830                 \l__zrefclever_name_format_tl { - }
2831                 \tl_put_left:NV \l__zrefclever_name_format_tl
2832                 \l__zrefclever_ref_decl_case_tl
2833                 \tl_put_left:Nn
2834                 \l__zrefclever_name_format_fallback_tl { - }
2835                 \tl_put_left:NV
2836                 \l__zrefclever_name_format_fallback_tl
2837                 \l__zrefclever_ref_decl_case_tl
2838             }
2839             \__zrefclever_get_type_transl:xxxNF
2840             { \l__zrefclever_ref_language_tl }
2841             { \l__zrefclever_type_first_label_type_tl }
2842             { \l__zrefclever_name_format_tl }
2843             \l__zrefclever_type_name_tl
2844             {
2845                 \__zrefclever_get_type_transl:xxxNF
2846                 { \l__zrefclever_ref_language_tl }
2847                 { \l__zrefclever_type_first_label_type_tl }
2848                 { \l__zrefclever_name_format_fallback_tl }
2849                 \l__zrefclever_type_name_tl
2850                 {
2851                     \tl_clear:N \l__zrefclever_type_name_tl
2852                     \msg_warning:nxxx { zref-clever }
2853                     { missing-name }

```

```

2854                                     { \l__zrefclever_name_format_tl }
2855                                     { \l__zrefclever_type_first_label_type_tl }
2856                                     }
2857                                 }
2858                            }
2859                       }
2860                   }
2861               }
2862           }
2863
2864       % Signal whether the type name is to be included in the hyperlink or not.
2865       \bool_lazy_any:nTF
2866       {
2867         { ! \l__zrefclever_use_hyperref_bool }
2868         { \l__zrefclever_link_star_bool }
2869         { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2870         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2871       }
2872       { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2873       {
2874         \bool_lazy_any:nTF
2875         {
2876           { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2877           {
2878             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2879             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2880           }
2881           {
2882             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2883             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2884             \l__zrefclever_typeset_last_bool &&
2885             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2886           }
2887         }
2888         { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2889         { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2890       }
2891   }

```

(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.

```

2892   \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2893   {
2894     \zref@ifpropundefined { urluse }
2895     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2896     {
2897       \zref@ifrefcontainsprop {#1} { urluse }
2898       { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2899       { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2900     }

```

```

2901 }
2902 \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 $\langle label\ b \rangle$ comes in immediate sequence from $\langle label\ a \rangle$. 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 {\langle label\ a \rangle} {\langle label\ b \rangle}

2903 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2904 {
2905   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2906   {#1} { externaldocument } { \c_empty_tl }
2907   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2908   {#2} { externaldocument } { \c_empty_tl }
2909
2910   \tl_if_eq:NNT
2911     \l__zrefclever_label_extdoc_a_tl
2912     \l__zrefclever_label_extdoc_b_tl
2913   {
2914     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2915     {
2916       \exp_args:Nxx \tl_if_eq:nnT
2917       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2918       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2919       {
2920         \int_compare:nNnTF
2921           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2922           =
2923           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2924           { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2925           {
2926             \int_compare:nNnT
2927               { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2928               =
2929               { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2930             {
2931               \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2932               \bool_set_true:N \l__zrefclever_next_is_same_bool
2933             }
2934           }
2935         }
2936       }
2937     }
2938     \exp_args:Nxx \tl_if_eq:nnT
2939     { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2940     { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2941     {
2942       \exp_args:Nxx \tl_if_eq:nnT

```

```

2943 { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2944 { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2945 {
2946   \int_compare:nNnTF
2947     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2948     =
2949     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2950     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2951     {
2952       \int_compare:nNnT
2953         { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2954         =
2955         { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2956         {
2957           \bool_set_true:N
2958             \l__zrefclever_next_maybe_range_bool
2959           \exp_args:Nxx \tl_if_eq:nnT
2960             {
2961               \_zrefclever_extract_unexp:nvn {#1}
2962               { \l__zrefclever_ref_property_tl } { }
2963             }
2964             {
2965               \_zrefclever_extract_unexp:nvn {#2}
2966               { \l__zrefclever_ref_property_tl } { }
2967             }
2968             {
2969               \bool_set_true:N
2970               \l__zrefclever_next_is_same_bool
2971             }
2972           }
2973         }
2974       }
2975     }
2976   }
2977 }
2978 }

```

(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>}
2979 \cs_new_protected:Npn \_zrefclever_get_ref_string:nN #1#2

```

```

2980 {
2981   % First attempt: general options.
2982   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2983   {
2984     % If not found, try type specific options.
2985     \bool_lazy_all:nTF
2986     {
2987       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2988       {
2989         \prop_if_exist_p:c
2990         {
2991           l__zrefclever_type_
2992           \l__zrefclever_label_type_a_tl _options_prop
2993         }
2994       }
2995       {
2996         \prop_if_in_p:cn
2997         {
2998           l__zrefclever_type_
2999           \l__zrefclever_label_type_a_tl _options_prop
3000         }
3001         {#1}
3002       }
3003     }
3004     {
3005       \prop_get:cnN
3006       {
3007         l__zrefclever_type_
3008         \l__zrefclever_label_type_a_tl _options_prop
3009       }
3010       {#1} #2
3011     }
3012     {
3013       % If not found, try type specific translations.
3014       \__zrefclever_get_type_transl:xnNF
3015       { \l__zrefclever_ref_language_tl }
3016       { \l__zrefclever_label_type_a_tl }
3017       {#1} #2
3018       {
3019         % If not found, try default translations.
3020         \__zrefclever_get_default_transl:xnNF
3021         { \l__zrefclever_ref_language_tl }
3022         {#1} #2
3023         {
3024           % If not found, try fallback.
3025           \__zrefclever_get_fallback_transl:nNF {#1} #2
3026           {
3027             \tl_clear:N #2
3028             \msg_warning:nnn { zref-clever }
3029             { missing-string } {#1}
3030           }
3031         }
3032       }
3033     }
  }

```

```

3034     }
3035 }

(End definition for \_zrefclever_get_ref_string:nN.)

\_zrefclever_get_ref_font:nN      \_zrefclever_get_ref_font:nN {<option>} {<tl variable>}
3036 \cs_new_protected:Npn \_zrefclever_get_ref_font:nN #1#2
3037 {
3038   % First attempt: general options.
3039   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3040   {
3041     % If not found, try type specific options.
3042     \bool_lazy_and:nnTF
3043       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
3044       {
3045         \prop_if_exist_p:c
3046           {
3047             l__zrefclever_type_
3048             \l__zrefclever_label_type_a_tl _options_prop
3049           }
3050       }
3051       {
3052         \prop_get:cnNF
3053           {
3054             l__zrefclever_type_
3055             \l__zrefclever_label_type_a_tl _options_prop
3056           }
3057           {#1} #2
3058           { \tl_clear:N #2 }
3059       }
3060       { \tl_clear:N #2 }
3061   }
3062 }

(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.

```

3063 \tl_new:N \l__zrefclever_footnote_type_tl
3064 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }

```

```

3065 \AddToHook { env / minipage / begin }
3066 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
3067 \AddToHook { cmd / @makefntext / before }
3068 {
3069   \__zrefclever_zcsetup:x
3070   { currentcounter = \l__zrefclever_footnote_type_tl }
3071 }

```

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.

```

3072 \AddToHook { cmd / appendix / before }
3073 {
3074   \__zrefclever_zcsetup:n
3075   {
3076     countertype =
3077     {
3078       chapter      = appendix ,
3079       section      = appendix ,
3080       subsection   = appendix ,
3081       subsubsection = appendix ,
3082     }
3083   }
3084 }

```

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 already handle this gracefully (see <https://github.com/latex3/latex2e/pull/699>, thanks Phelype Oleinik). 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.

```

3085 \AddToHook { begindocument }
3086 {
3087   \@ifpackageloaded { appendix }
3088   {
3089     \newcounter { zc@appendix }
3090     \newcounter { zc@save@appendix }
3091     \setcounter { zc@appendix } { 0 }
3092     \setcounter { zc@save@appendix } { 0 }
3093     \cs_if_exist:cTF { chapter }
3094     {
3095       \__zrefclever_zcsetup:n
3096       { counterresetby = { chapter = zc@appendix } }
3097     }
3098     {
3099       \cs_if_exist:cT { section }
3100       {
3101         \__zrefclever_zcsetup:n
3102         { counterresetby = { section = zc@appendix } }
3103       }
3104     }
3105   \AddToHook { env / appendices / begin }
3106   {
3107     \stepcounter { zc@save@appendix }
3108     \setcounter { zc@appendix } { \value { zc@save@appendix } }
3109     \__zrefclever_zcsetup:n
3110     {
3111       countertype =
3112       {
3113         chapter      = appendix ,
3114         section      = appendix ,
3115         subsection   = appendix ,
3116         subsubsection = appendix ,
3117       }
3118     }
3119   }
3120   \AddToHook { env / appendices / end }
3121   { \setcounter { zc@appendix } { 0 } }
3122   \AddToHook { cmd / appendix / before }
3123   {
3124     \stepcounter { zc@save@appendix }

```

```

3125         \setcounter { zc@appendix } { \value { zc@save@appendix } }
3126     }
3127     \AddToHook { env / subappendices / begin }
3128     {
3129         \__zrefclever_zcsetup:n
3130         {
3131             countertype =
3132             {
3133                 section      = appendix ,
3134                 subsection   = appendix ,
3135                 subsubsection = appendix ,
3136             } ,
3137         }
3138     }
3139     \msg_info:nnn { zref-clever } { compat-package } { appendix }
3140 }
3141 {}
3142 }

```

9.4 amsmath package

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

```

3143 \AddToHook { begindocument }
3144 {
3145     \ifpackageloaded { amsmath }
3146     {

```

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...).

```

3147         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
3148         {
3149             \__zrefclever_orig_ltxlabel:n {#1}
3150             \zref@wrapper@babel \zref@label {#1}
3151         }

```

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 `zref-clever`, 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? TODO Remove this compatibility conditional when 2021-11-15 release comes.

```

3152 \IfFormatAtLeastTF { 2021-11-15 }
3153 {
3154   \@ifpackageloaded { hyperref }
3155   {
3156     \AddToHook { package / nameref / after }
3157     {
3158       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3159       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3160     }
3161   }
3162   {
3163     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3164     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3165   }
3166 }
3167 {
3168   \@ifpackageloaded { hyperref }
3169   {
3170     \@ifpackageloaded { nameref }
3171     {
3172       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3173       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3174     }
3175     {
3176       \AddToHook { package / after / nameref }
3177       {
3178         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3179         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3180       }
3181     }
3182   }
3183   {
3184     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3185     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3186   }
3187 }

```

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).

```

3188 \AddToHook { env / subequations / begin }
3189 {
3190   \__zrefclever_zcsetup:x
3191   {
3192     counterresetby =
3193     {
3194       parentequation =
3195         \__zrefclever_counter_reset_by:n { equation } ,

```

```

3196         equation = parentequation ,
3197     } ,
3198     currentcounter = parentequation ,
3199     countertype = { parentequation = equation } ,
3200 }
3201 }

```

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

```

3202 \clist_map_inline:nn
3203 {
3204     equation ,
3205     equation* ,
3206     align ,
3207     align* ,
3208     alignat ,
3209     alignat* ,
3210     flalign ,
3211     flalign* ,
3212     xalignat ,
3213     xalignat* ,
3214     gather ,
3215     gather* ,
3216     multiline ,
3217     multiline* ,
3218 }
3219 {
3220     \AddToHook { env / #1 / begin }
3221     { \__zrefclever_zcsetup:n { currentcounter = equation } }
3222 }

```

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

```

3223 \zcRefTypeSetup { equation }
3224 {
3225     reffont = \upshape ,
3226     refpre  = {\textup{()}} ,
3227     refpos  = {\textup{()}} ,
3228 }
3229 \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3230 }
3231 {}

```

3232 }

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.

```

3233 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3234 \AddToHook { begindocument }
3235 {
3236   \@ifpackageloaded { mathtools }
3237   {
3238     \MH_if_boolean:nT { show_only_refs }
3239     {
3240       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3241       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
3242       {
3243         \@bsphack
3244         \seq_map_inline:Nn #1
3245         {
3246           \exp_args:Nx \tl_if_eq:nnTF
3247           { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3248           { equation }
3249           {
3250             \protected@write \@auxout { }
3251             { \string \MT@newlabel {##1} }
3252           }
3253           {
3254             \exp_args:Nx \tl_if_eq:nnT
3255             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3256             { parentequation }
3257             {
3258               \protected@write \@auxout { }
3259               { \string \MT@newlabel {##1} }
3260             }
3261           }
3262         }
3263       }
3264     }
3265     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3266   }
3267 }
3268 {}
3269 }
```

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”.

```
3270 \AddToHook { begindocument }
3271 {
3272   \@ifpackageloaded { breqn }
3273   {
```

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.

```
3274     \AddToHook { env / dgroup / begin }
3275     {
3276       \__zrefclever_zcsetup:x
3277       {
3278         counterresetby =
3279         {
3280           parentequation =
3281             \__zrefclever_counter_reset_by:n { equation } ,
3282           equation = parentequation ,
3283         } ,
3284         currentcounter = parentequation ,
3285         countertype = { parentequation = equation } ,
3286       }
3287     }
3288     \clist_map_inline:nn
3289     {
3290       dmath ,
3291       dseries ,
3292       darray ,
3293     }
3294     {
3295       \AddToHook { env / #1 / begin }
3296       { \__zrefclever_zcsetup:n { currentcounter = equation } }
3297     }
3298   }
3299 {}
3300 }
```

9.7 listings package

```
3301 \AddToHook { begindocument }
3302 {
3303   \@ifpackageloaded { listings }
3304   {
3305     \__zrefclever_zcsetup:n
3306     {
```

```

3307         countertype =
3308         {
3309             lstlisting = listing ,
3310             lstnumber = line ,
3311         } ,
3312         counterresetby = { lstnumber = lstlisting } ,
3313     }
3314     \lst@AddToHook { Init }
3315     {

```

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

```

3316         \tl_if_empty:NF \lst@label
3317         { \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.

```

3318         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
3319     }
3320     \msg_info:nnn { zref-clever } { compat-package } { listings }
3321 }
3322 {}
3323 }

```

9.8 enumitem package

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

The only real reason to “renew” `enumerate` itself is to change `{\max-depth}`. `\renewlist` *hard-codes* `max-depth` in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But `\renewlist` also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from `zref-clever`’s perspective. Since the first four are defined by the kernel and already setup for `zref-clever` by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```

3324 \AddToHook { begindocument }
3325 {
3326     \@ifpackageloaded { enumitem }
3327     {
3328         \int_set:Nn \l_tmpa_int { 5 }
3329         \bool_while_do:nn
3330         {
3331             \cs_if_exist_p:c
3332             { c@_enum \int_to_roman:n { \l_tmpa_int } }

```

```

3333     }
3334     {
3335         \__zrefclever_zcsetup:x
3336         {
3337             counterresetby =
3338             {
3339                 enum \int_to_roman:n { \l_tmpa_int } =
3340                 enum \int_to_roman:n { \l_tmpa_int - 1 }
3341             } ,
3342             countertype =
3343             { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3344         }
3345         \int_incr:N \l_tmpa_int
3346     }
3347     \int_compare:nNnT { \l_tmpa_int } > { 5 }
3348     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3349 }
3350 {}
3351 }
3352 </package>

```

10 Dictionaries

10.1 English

```

3353 <*package>
3354 \zcDeclareLanguage { english }
3355 \zcDeclareLanguageAlias { american } { english }
3356 \zcDeclareLanguageAlias { australian } { english }
3357 \zcDeclareLanguageAlias { british } { english }
3358 \zcDeclareLanguageAlias { canadian } { english }
3359 \zcDeclareLanguageAlias { newzealand } { english }
3360 \zcDeclareLanguageAlias { UKenglish } { english }
3361 \zcDeclareLanguageAlias { USenglish } { english }
3362 </package>
3363 <*dict-english>
3364 namesep = {\nobreakspace} ,
3365 pairsep = {\~and\nobreakspace} ,
3366 listsep = {\~,~} ,
3367 lastsep = {\~and\nobreakspace} ,
3368 tpairsep = {\~and\nobreakspace} ,
3369 tlistsep = {\~,~} ,
3370 tlastsep = {\~,~and\nobreakspace} ,
3371 notesep = {\~} ,
3372 rangesep = {\~to\nobreakspace} ,
3373
3374 type = part ,
3375 Name-sg = Part ,
3376 name-sg = part ,
3377 Name-pl = Parts ,
3378 name-pl = parts ,
3379

```



```

3380 type = chapter ,
3381     Name-sg = Chapter ,
3382     name-sg = chapter ,
3383     Name-pl = Chapters ,
3384     name-pl = chapters ,
3385
3386 type = section ,
3387     Name-sg = Section ,
3388     name-sg = section ,
3389     Name-pl = Sections ,
3390     name-pl = sections ,
3391
3392 type = paragraph ,
3393     Name-sg = Paragraph ,
3394     name-sg = paragraph ,
3395     Name-pl = Paragraphs ,
3396     name-pl = paragraphs ,
3397     Name-sg-ab = Par. ,
3398     name-sg-ab = par. ,
3399     Name-pl-ab = Par. ,
3400     name-pl-ab = par. ,
3401
3402 type = appendix ,
3403     Name-sg = Appendix ,
3404     name-sg = appendix ,
3405     Name-pl = Appendices ,
3406     name-pl = appendices ,
3407
3408 type = subappendix ,
3409     Name-sg = Appendix ,
3410     name-sg = appendix ,
3411     Name-pl = Appendices ,
3412     name-pl = appendices ,
3413
3414 type = page ,
3415     Name-sg = Page ,
3416     name-sg = page ,
3417     Name-pl = Pages ,
3418     name-pl = pages ,
3419     name-sg-ab = p. ,
3420     name-pl-ab = pp. ,
3421     rangesep = {\textendash} ,
3422
3423 type = line ,
3424     Name-sg = Line ,
3425     name-sg = line ,
3426     Name-pl = Lines ,
3427     name-pl = lines ,
3428
3429 type = figure ,
3430     Name-sg = Figure ,
3431     name-sg = figure ,
3432     Name-pl = Figures ,
3433     name-pl = figures ,

```

```

3434 Name-sg-ab = Fig. ,
3435 name-sg-ab = fig. ,
3436 Name-pl-ab = Figs. ,
3437 name-pl-ab = figs. ,
3438
3439 type = table ,
3440 Name-sg = Table ,
3441 name-sg = table ,
3442 Name-pl = Tables ,
3443 name-pl = tables ,
3444
3445 type = item ,
3446 Name-sg = Item ,
3447 name-sg = item ,
3448 Name-pl = Items ,
3449 name-pl = items ,
3450
3451 type = footnote ,
3452 Name-sg = Footnote ,
3453 name-sg = footnote ,
3454 Name-pl = Footnotes ,
3455 name-pl = footnotes ,
3456
3457 type = note ,
3458 Name-sg = Note ,
3459 name-sg = note ,
3460 Name-pl = Notes ,
3461 name-pl = notes ,
3462
3463 type = equation ,
3464 Name-sg = Equation ,
3465 name-sg = equation ,
3466 Name-pl = Equations ,
3467 name-pl = equations ,
3468 Name-sg-ab = Eq. ,
3469 name-sg-ab = eq. ,
3470 Name-pl-ab = Eqs. ,
3471 name-pl-ab = eqs. ,
3472 refpre = {} ,
3473 refpos = {} ,
3474
3475 type = theorem ,
3476 Name-sg = Theorem ,
3477 name-sg = theorem ,
3478 Name-pl = Theorems ,
3479 name-pl = theorems ,
3480
3481 type = lemma ,
3482 Name-sg = Lemma ,
3483 name-sg = lemma ,
3484 Name-pl = Lemmas ,
3485 name-pl = lemmas ,
3486
3487 type = corollary ,

```

```

3488     Name-sg = Corollary ,
3489     name-sg = corollary ,
3490     Name-pl = Corollaries ,
3491     name-pl = corollaries ,
3492
3493 type = proposition ,
3494     Name-sg = Proposition ,
3495     name-sg = proposition ,
3496     Name-pl = Propositions ,
3497     name-pl = propositions ,
3498
3499 type = definition ,
3500     Name-sg = Definition ,
3501     name-sg = definition ,
3502     Name-pl = Definitions ,
3503     name-pl = definitions ,
3504
3505 type = proof ,
3506     Name-sg = Proof ,
3507     name-sg = proof ,
3508     Name-pl = Proofs ,
3509     name-pl = proofs ,
3510
3511 type = result ,
3512     Name-sg = Result ,
3513     name-sg = result ,
3514     Name-pl = Results ,
3515     name-pl = results ,
3516
3517 type = remark ,
3518     Name-sg = Remark ,
3519     name-sg = remark ,
3520     Name-pl = Remarks ,
3521     name-pl = remarks ,
3522
3523 type = example ,
3524     Name-sg = Example ,
3525     name-sg = example ,
3526     Name-pl = Examples ,
3527     name-pl = examples ,
3528
3529 type = algorithm ,
3530     Name-sg = Algorithm ,
3531     name-sg = algorithm ,
3532     Name-pl = Algorithms ,
3533     name-pl = algorithms ,
3534
3535 type = listing ,
3536     Name-sg = Listing ,
3537     name-sg = listing ,
3538     Name-pl = Listings ,
3539     name-pl = listings ,
3540
3541 type = exercise ,

```

```

3542 Name-sg = Exercise ,
3543 name-sg = exercise ,
3544 Name-pl = Exercises ,
3545 name-pl = exercises ,
3546
3547 type = solution ,
3548 Name-sg = Solution ,
3549 name-sg = solution ,
3550 Name-pl = Solutions ,
3551 name-pl = solutions ,
3552 </dict-english>

```

10.2 German

```

3553 <*package>
3554 \zcDeclareLanguage
3555 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
3556 { german }
3557 \zcDeclareLanguageAlias { austrian      } { german }
3558 \zcDeclareLanguageAlias { germanb       } { german }
3559 \zcDeclareLanguageAlias { ngerman       } { german }
3560 \zcDeclareLanguageAlias { naustrian     } { german }
3561 \zcDeclareLanguageAlias { nswissgerman } { german }
3562 \zcDeclareLanguageAlias { swissgerman   } { german }
3563 </package>
3564 <*dict-german>
3565 namesep = {\nobreakspace} ,
3566 pairsep = {\und\nobreakspace} ,
3567 listsep = {,~} ,
3568 lastsep = {\und\nobreakspace} ,
3569 tpairsep = {\und\nobreakspace} ,
3570 tlistsep = {,~} ,
3571 tlastsep = {\und\nobreakspace} ,
3572 notesep = {~} ,
3573 rangesep = {\bis\nobreakspace} ,
3574
3575 type = part ,
3576 gender = m ,
3577 case = N ,
3578   Name-sg = Teil ,
3579   Name-pl = Teile ,
3580 case = A ,
3581   Name-sg = Teil ,
3582   Name-pl = Teile ,
3583 case = D ,
3584   Name-sg = Teil ,
3585   Name-pl = Teilen ,
3586 case = G ,
3587   Name-sg = Teiles ,
3588   Name-pl = Teile ,
3589
3590 type = chapter ,
3591 gender = n ,
3592 case = N ,

```

```

3593     Name-sg = Kapitel ,
3594     Name-pl = Kapitel ,
3595     case = A ,
3596     Name-sg = Kapitel ,
3597     Name-pl = Kapitel ,
3598     case = D ,
3599     Name-sg = Kapitel ,
3600     Name-pl = Kapiteln ,
3601     case = G ,
3602     Name-sg = Kapitels ,
3603     Name-pl = Kapitel ,
3604
3605 type = section ,
3606     gender = m ,
3607     case = N ,
3608     Name-sg = Abschnitt ,
3609     Name-pl = Abschnitte ,
3610     case = A ,
3611     Name-sg = Abschnitt ,
3612     Name-pl = Abschnitte ,
3613     case = D ,
3614     Name-sg = Abschnitt ,
3615     Name-pl = Abschnitten ,
3616     case = G ,
3617     Name-sg = Abschnitts ,
3618     Name-pl = Abschnitte ,
3619
3620 type = paragraph ,
3621     gender = m ,
3622     case = N ,
3623     Name-sg = Absatz ,
3624     Name-pl = Absätze ,
3625     case = A ,
3626     Name-sg = Absatz ,
3627     Name-pl = Absätze ,
3628     case = D ,
3629     Name-sg = Absatz ,
3630     Name-pl = Absätzen ,
3631     case = G ,
3632     Name-sg = Absatzes ,
3633     Name-pl = Absätze ,
3634
3635 type = appendix ,
3636     gender = m ,
3637     case = N ,
3638     Name-sg = Anhang ,
3639     Name-pl = Anhänge ,
3640     case = A ,
3641     Name-sg = Anhang ,
3642     Name-pl = Anhänge ,
3643     case = D ,
3644     Name-sg = Anhang ,
3645     Name-pl = Anhängen ,
3646     case = G ,

```

```

3647     Name-sg = Anhangs ,
3648     Name-pl = Anhänge ,
3649
3650 type = subappendix ,
3651     gender = m ,
3652     case = N ,
3653     Name-sg = Anhang ,
3654     Name-pl = Anhänge ,
3655     case = A ,
3656     Name-sg = Anhang ,
3657     Name-pl = Anhänge ,
3658     case = D ,
3659     Name-sg = Anhang ,
3660     Name-pl = Anhängen ,
3661     case = G ,
3662     Name-sg = Anhangs ,
3663     Name-pl = Anhänge ,
3664
3665 type = page ,
3666     gender = f ,
3667     case = N ,
3668     Name-sg = Seite ,
3669     Name-pl = Seiten ,
3670     case = A ,
3671     Name-sg = Seite ,
3672     Name-pl = Seiten ,
3673     case = D ,
3674     Name-sg = Seite ,
3675     Name-pl = Seiten ,
3676     case = G ,
3677     Name-sg = Seite ,
3678     Name-pl = Seiten ,
3679     rangesep = {\textendash} ,
3680
3681 type = line ,
3682     gender = f ,
3683     case = N ,
3684     Name-sg = Zeile ,
3685     Name-pl = Zeilen ,
3686     case = A ,
3687     Name-sg = Zeile ,
3688     Name-pl = Zeilen ,
3689     case = D ,
3690     Name-sg = Zeile ,
3691     Name-pl = Zeilen ,
3692     case = G ,
3693     Name-sg = Zeile ,
3694     Name-pl = Zeilen ,
3695
3696 type = figure ,
3697     gender = f ,
3698     case = N ,
3699     Name-sg = Abbildung ,
3700     Name-pl = Abbildungen ,

```

```

3701     Name-sg-ab = Abb. ,
3702     Name-pl-ab = Abb. ,
3703     case = A ,
3704         Name-sg = Abbildung ,
3705         Name-pl = Abbildungen ,
3706         Name-sg-ab = Abb. ,
3707         Name-pl-ab = Abb. ,
3708     case = D ,
3709         Name-sg = Abbildung ,
3710         Name-pl = Abbildungen ,
3711         Name-sg-ab = Abb. ,
3712         Name-pl-ab = Abb. ,
3713     case = G ,
3714         Name-sg = Abbildung ,
3715         Name-pl = Abbildungen ,
3716         Name-sg-ab = Abb. ,
3717         Name-pl-ab = Abb. ,
3718
3719     type = table ,
3720         gender = f ,
3721         case = N ,
3722             Name-sg = Tabelle ,
3723             Name-pl = Tabellen ,
3724         case = A ,
3725             Name-sg = Tabelle ,
3726             Name-pl = Tabellen ,
3727         case = D ,
3728             Name-sg = Tabelle ,
3729             Name-pl = Tabellen ,
3730         case = G ,
3731             Name-sg = Tabelle ,
3732             Name-pl = Tabellen ,
3733
3734     type = item ,
3735         gender = m ,
3736         case = N ,
3737             Name-sg = Punkt ,
3738             Name-pl = Punkte ,
3739         case = A ,
3740             Name-sg = Punkt ,
3741             Name-pl = Punkte ,
3742         case = D ,
3743             Name-sg = Punkt ,
3744             Name-pl = Punkten ,
3745         case = G ,
3746             Name-sg = Punktes ,
3747             Name-pl = Punkte ,
3748
3749     type = footnote ,
3750         gender = f ,
3751         case = N ,
3752             Name-sg = Fußnote ,
3753             Name-pl = Fußnoten ,
3754         case = A ,

```

```

3755     Name-sg = Fußnote ,
3756     Name-pl = Fußnoten ,
3757     case = D ,
3758     Name-sg = Fußnote ,
3759     Name-pl = Fußnoten ,
3760     case = G ,
3761     Name-sg = Fußnote ,
3762     Name-pl = Fußnoten ,
3763
3764 type = note ,
3765     gender = f ,
3766     case = N ,
3767     Name-sg = Anmerkung ,
3768     Name-pl = Anmerkungen ,
3769     case = A ,
3770     Name-sg = Anmerkung ,
3771     Name-pl = Anmerkungen ,
3772     case = D ,
3773     Name-sg = Anmerkung ,
3774     Name-pl = Anmerkungen ,
3775     case = G ,
3776     Name-sg = Anmerkung ,
3777     Name-pl = Anmerkungen ,
3778
3779 type = equation ,
3780     gender = f ,
3781     case = N ,
3782     Name-sg = Gleichung ,
3783     Name-pl = Gleichungen ,
3784     case = A ,
3785     Name-sg = Gleichung ,
3786     Name-pl = Gleichungen ,
3787     case = D ,
3788     Name-sg = Gleichung ,
3789     Name-pl = Gleichungen ,
3790     case = G ,
3791     Name-sg = Gleichung ,
3792     Name-pl = Gleichungen ,
3793     refpre = {() ,
3794     refpos = {}} ,
3795
3796 type = theorem ,
3797     gender = n ,
3798     case = N ,
3799     Name-sg = Theorem ,
3800     Name-pl = Theoreme ,
3801     case = A ,
3802     Name-sg = Theorem ,
3803     Name-pl = Theoreme ,
3804     case = D ,
3805     Name-sg = Theorem ,
3806     Name-pl = Theoremen ,
3807     case = G ,
3808     Name-sg = Theorems ,

```



```

3809     Name-pl = Theoreme ,
3810
3811 type = lemma ,
3812     gender = n ,
3813     case = N ,
3814     Name-sg = Lemma ,
3815     Name-pl = Lemmata ,
3816     case = A ,
3817     Name-sg = Lemma ,
3818     Name-pl = Lemmata ,
3819     case = D ,
3820     Name-sg = Lemma ,
3821     Name-pl = Lemmata ,
3822     case = G ,
3823     Name-sg = Lemmas ,
3824     Name-pl = Lemmata ,
3825
3826 type = corollary ,
3827     gender = n ,
3828     case = N ,
3829     Name-sg = Korollar ,
3830     Name-pl = Korollare ,
3831     case = A ,
3832     Name-sg = Korollar ,
3833     Name-pl = Korollare ,
3834     case = D ,
3835     Name-sg = Korollar ,
3836     Name-pl = Korollaren ,
3837     case = G ,
3838     Name-sg = Korollars ,
3839     Name-pl = Korollare ,
3840
3841 type = proposition ,
3842     gender = m ,
3843     case = N ,
3844     Name-sg = Satz ,
3845     Name-pl = Sätze ,
3846     case = A ,
3847     Name-sg = Satz ,
3848     Name-pl = Sätze ,
3849     case = D ,
3850     Name-sg = Satz ,
3851     Name-pl = Sätzen ,
3852     case = G ,
3853     Name-sg = Satzes ,
3854     Name-pl = Sätze ,
3855
3856 type = definition ,
3857     gender = f ,
3858     case = N ,
3859     Name-sg = Definition ,
3860     Name-pl = Definitionen ,
3861     case = A ,
3862     Name-sg = Definition ,

```

```

3863     Name-pl = Definitionen ,
3864     case = D ,
3865     Name-sg = Definition ,
3866     Name-pl = Definitionen ,
3867     case = G ,
3868     Name-sg = Definition ,
3869     Name-pl = Definitionen ,
3870
3871 type = proof ,
3872     gender = m ,
3873     case = N ,
3874     Name-sg = Beweis ,
3875     Name-pl = Beweise ,
3876     case = A ,
3877     Name-sg = Beweis ,
3878     Name-pl = Beweise ,
3879     case = D ,
3880     Name-sg = Beweis ,
3881     Name-pl = Beweisen ,
3882     case = G ,
3883     Name-sg = Beweises ,
3884     Name-pl = Beweise ,
3885
3886 type = result ,
3887     gender = n ,
3888     case = N ,
3889     Name-sg = Ergebnis ,
3890     Name-pl = Ergebnisse ,
3891     case = A ,
3892     Name-sg = Ergebnis ,
3893     Name-pl = Ergebnisse ,
3894     case = D ,
3895     Name-sg = Ergebnis ,
3896     Name-pl = Ergebnissen ,
3897     case = G ,
3898     Name-sg = Ergebnisses ,
3899     Name-pl = Ergebnisse ,
3900
3901 type = remark ,
3902     gender = f ,
3903     case = N ,
3904     Name-sg = Bemerkung ,
3905     Name-pl = Bemerkungen ,
3906     case = A ,
3907     Name-sg = Bemerkung ,
3908     Name-pl = Bemerkungen ,
3909     case = D ,
3910     Name-sg = Bemerkung ,
3911     Name-pl = Bemerkungen ,
3912     case = G ,
3913     Name-sg = Bemerkung ,
3914     Name-pl = Bemerkungen ,
3915
3916 type = example ,

```

```

3917 gender = n ,
3918 case = N ,
3919     Name-sg = Beispiel ,
3920     Name-pl = Beispiele ,
3921 case = A ,
3922     Name-sg = Beispiel ,
3923     Name-pl = Beispiele ,
3924 case = D ,
3925     Name-sg = Beispiel ,
3926     Name-pl = Beispielen ,
3927 case = G ,
3928     Name-sg = Beispiels ,
3929     Name-pl = Beispiele ,
3930
3931 type = algorithm ,
3932 gender = m ,
3933 case = N ,
3934     Name-sg = Algorithmus ,
3935     Name-pl = Algorithmen ,
3936 case = A ,
3937     Name-sg = Algorithmus ,
3938     Name-pl = Algorithmen ,
3939 case = D ,
3940     Name-sg = Algorithmus ,
3941     Name-pl = Algorithmen ,
3942 case = G ,
3943     Name-sg = Algorithmus ,
3944     Name-pl = Algorithmen ,
3945
3946 type = listing ,
3947 gender = n ,
3948 case = N ,
3949     Name-sg = Listing ,
3950     Name-pl = Listings ,
3951 case = A ,
3952     Name-sg = Listing ,
3953     Name-pl = Listings ,
3954 case = D ,
3955     Name-sg = Listing ,
3956     Name-pl = Listings ,
3957 case = G ,
3958     Name-sg = Listings ,
3959     Name-pl = Listings ,
3960
3961 type = exercise ,
3962 gender = f ,
3963 case = N ,
3964     Name-sg = Übungsaufgabe ,
3965     Name-pl = Übungsaufgaben ,
3966 case = A ,
3967     Name-sg = Übungsaufgabe ,
3968     Name-pl = Übungsaufgaben ,
3969 case = D ,
3970     Name-sg = Übungsaufgabe ,

```

```

3971     Name-pl = Übungsaufgaben ,
3972     case = G ,
3973     Name-sg = Übungsaufgabe ,
3974     Name-pl = Übungsaufgaben ,
3975
3976 type = solution ,
3977     gender = f ,
3978     case = N ,
3979     Name-sg = Lösung ,
3980     Name-pl = Lösungen ,
3981     case = A ,
3982     Name-sg = Lösung ,
3983     Name-pl = Lösungen ,
3984     case = D ,
3985     Name-sg = Lösung ,
3986     Name-pl = Lösungen ,
3987     case = G ,
3988     Name-sg = Lösung ,
3989     Name-pl = Lösungen ,
3990 </dict-german>

```

10.3 French

```

3991 <*package>
3992 \zcDeclareLanguage [ gender = { f , m } ] { french }
3993 \zcDeclareLanguageAlias { acadian } { french }
3994 \zcDeclareLanguageAlias { canadien } { french }
3995 \zcDeclareLanguageAlias { francais } { french }
3996 \zcDeclareLanguageAlias { frenchb } { french }
3997 </package>
3998 <*dict-french>
3999 namesep = {\nobreakspace} ,
4000 pairsep = {\~et\nobreakspace} ,
4001 listsep = { ,~ } ,
4002 lastsep = {\~et\nobreakspace} ,
4003 tpairsep = {\~et\nobreakspace} ,
4004 tlistsep = { ,~ } ,
4005 tlastsep = {\~et\nobreakspace} ,
4006 notesep = {~} ,
4007 rangesep = {\~à\nobreakspace} ,
4008
4009 type = part ,
4010     gender = f ,
4011     Name-sg = Partie ,
4012     name-sg = partie ,
4013     Name-pl = Parties ,
4014     name-pl = parties ,
4015
4016 type = chapter ,
4017     gender = m ,
4018     Name-sg = Chapitre ,
4019     name-sg = chapitre ,
4020     Name-pl = Chapitres ,
4021     name-pl = chapitres ,

```

```

4022
4023 type = section ,
4024     gender = f ,
4025     Name-sg = Section ,
4026     name-sg = section ,
4027     Name-pl = Sections ,
4028     name-pl = sections ,
4029
4030 type = paragraph ,
4031     gender = m ,
4032     Name-sg = Paragraphe ,
4033     name-sg = paragraphe ,
4034     Name-pl = Paragraphes ,
4035     name-pl = paragraphes ,
4036
4037 type = appendix ,
4038     gender = f ,
4039     Name-sg = Annexe ,
4040     name-sg = annexe ,
4041     Name-pl = Annexes ,
4042     name-pl = annexes ,
4043
4044 type = subappendix ,
4045     gender = f ,
4046     Name-sg = Annexe ,
4047     name-sg = annexe ,
4048     Name-pl = Annexes ,
4049     name-pl = annexes ,
4050
4051 type = page ,
4052     gender = f ,
4053     Name-sg = Page ,
4054     name-sg = page ,
4055     Name-pl = Pages ,
4056     name-pl = pages ,
4057     rangesep = {\textendash} ,
4058
4059 type = line ,
4060     gender = f ,
4061     Name-sg = Ligne ,
4062     name-sg = ligne ,
4063     Name-pl = Lignes ,
4064     name-pl = lignes ,
4065
4066 type = figure ,
4067     gender = f ,
4068     Name-sg = Figure ,
4069     name-sg = figure ,
4070     Name-pl = Figures ,
4071     name-pl = figures ,
4072
4073 type = table ,
4074     gender = f ,
4075     Name-sg = Table ,

```

```

4076     name-sg = table ,
4077     Name-pl = Tables ,
4078     name-pl = tables ,
4079
4080 type = item ,
4081     gender = m ,
4082     Name-sg = Point ,
4083     name-sg = point ,
4084     Name-pl = Points ,
4085     name-pl = points ,
4086
4087 type = footnote ,
4088     gender = f ,
4089     Name-sg = Note ,
4090     name-sg = note ,
4091     Name-pl = Notes ,
4092     name-pl = notes ,
4093
4094 type = note ,
4095     gender = f ,
4096     Name-sg = Note ,
4097     name-sg = note ,
4098     Name-pl = Notes ,
4099     name-pl = notes ,
4100
4101 type = equation ,
4102     gender = f ,
4103     Name-sg = Équation ,
4104     name-sg = équation ,
4105     Name-pl = Équations ,
4106     name-pl = équations ,
4107     refpre = {(} ,
4108     refpos = {)} ,
4109
4110 type = theorem ,
4111     gender = m ,
4112     Name-sg = Théorème ,
4113     name-sg = théorème ,
4114     Name-pl = Théorèmes ,
4115     name-pl = théorèmes ,
4116
4117 type = lemma ,
4118     gender = m ,
4119     Name-sg = Lemme ,
4120     name-sg = lemme ,
4121     Name-pl = Lemmes ,
4122     name-pl = lemmes ,
4123
4124 type = corollary ,
4125     gender = m ,
4126     Name-sg = Corollaire ,
4127     name-sg = corollaire ,
4128     Name-pl = Corollaires ,
4129     name-pl = corollaires ,

```

```

4130
4131 type = proposition ,
4132     gender = f ,
4133     Name-sg = Proposition ,
4134     name-sg = proposition ,
4135     Name-pl = Propositions ,
4136     name-pl = propositions ,
4137
4138 type = definition ,
4139     gender = f ,
4140     Name-sg = Définition ,
4141     name-sg = définition ,
4142     Name-pl = Définitions ,
4143     name-pl = définitions ,
4144
4145 type = proof ,
4146     gender = f ,
4147     Name-sg = Démonstration ,
4148     name-sg = démonstration ,
4149     Name-pl = Démonstrations ,
4150     name-pl = démonstrations ,
4151
4152 type = result ,
4153     gender = m ,
4154     Name-sg = Résultat ,
4155     name-sg = résultat ,
4156     Name-pl = Résultats ,
4157     name-pl = résultats ,
4158
4159 type = remark ,
4160     gender = f ,
4161     Name-sg = Remarque ,
4162     name-sg = remarque ,
4163     Name-pl = Remarques ,
4164     name-pl = remarques ,
4165
4166 type = example ,
4167     gender = m ,
4168     Name-sg = Exemple ,
4169     name-sg = exemple ,
4170     Name-pl = Exemples ,
4171     name-pl = exemples ,
4172
4173 type = algorithm ,
4174     gender = m ,
4175     Name-sg = Algorithme ,
4176     name-sg = algorithme ,
4177     Name-pl = Algorithmes ,
4178     name-pl = algorithmes ,
4179
4180 type = listing ,
4181     gender = f ,
4182     Name-sg = Liste ,
4183     name-sg = liste ,

```

```

4184   Name-pl = Listes ,
4185   name-pl = listes ,
4186
4187   type = exercise ,
4188   gender = m ,
4189   Name-sg = Exercice ,
4190   name-sg = exercice ,
4191   Name-pl = Exercices ,
4192   name-pl = exercices ,
4193
4194   type = solution ,
4195   gender = f ,
4196   Name-sg = Solution ,
4197   name-sg = solution ,
4198   Name-pl = Solutions ,
4199   name-pl = solutions ,
4200 </dict-french>

```

10.4 Portuguese

```

4201 <*package>
4202 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
4203 \zcDeclareLanguageAlias { brazilian } { portuguese }
4204 \zcDeclareLanguageAlias { brazil } { portuguese }
4205 \zcDeclareLanguageAlias { portuges } { portuguese }
4206 </package>
4207 <*dict-portuguese>
4208 namesep = {\nobreakspace} ,
4209 pairsep = {\e\nobreakspace} ,
4210 listsep = {,~} ,
4211 lastsep = {\e\nobreakspace} ,
4212 tpairsep = {\e\nobreakspace} ,
4213 tlistsep = {,~} ,
4214 tlastsep = {\e\nobreakspace} ,
4215 notesep = {~} ,
4216 rangesep = {\a\nobreakspace} ,
4217
4218 type = part ,
4219 gender = f ,
4220 Name-sg = Parte ,
4221 name-sg = parte ,
4222 Name-pl = Partes ,
4223 name-pl = partes ,
4224
4225 type = chapter ,
4226 gender = m ,
4227 Name-sg = Capítulo ,
4228 name-sg = capítulo ,
4229 Name-pl = Capítulos ,
4230 name-pl = capítulos ,
4231
4232 type = section ,
4233 gender = f ,
4234 Name-sg = Seção ,

```



```

4235     name-sg = seção ,
4236     Name-pl = Seções ,
4237     name-pl = seções ,
4238
4239     type = paragraph ,
4240     gender = m ,
4241     Name-sg = Parágrafo ,
4242     name-sg = parágrafo ,
4243     Name-pl = Parágrafos ,
4244     name-pl = parágrafos ,
4245     Name-sg-ab = Par. ,
4246     name-sg-ab = par. ,
4247     Name-pl-ab = Par. ,
4248     name-pl-ab = par. ,
4249
4250     type = appendix ,
4251     gender = m ,
4252     Name-sg = Apêndice ,
4253     name-sg = apêndice ,
4254     Name-pl = Apêndices ,
4255     name-pl = apêndices ,
4256
4257     type = subappendix ,
4258     gender = m ,
4259     Name-sg = Apêndice ,
4260     name-sg = apêndice ,
4261     Name-pl = Apêndices ,
4262     name-pl = apêndices ,
4263
4264     type = page ,
4265     gender = f ,
4266     Name-sg = Página ,
4267     name-sg = página ,
4268     Name-pl = Páginas ,
4269     name-pl = páginas ,
4270     name-sg-ab = p. ,
4271     name-pl-ab = pp. ,
4272     rangesep = {\textendash} ,
4273
4274     type = line ,
4275     gender = f ,
4276     Name-sg = Linha ,
4277     name-sg = linha ,
4278     Name-pl = Linhas ,
4279     name-pl = linhas ,
4280
4281     type = figure ,
4282     gender = f ,
4283     Name-sg = Figura ,
4284     name-sg = figura ,
4285     Name-pl = Figuras ,
4286     name-pl = figuras ,
4287     Name-sg-ab = Fig. ,
4288     name-sg-ab = fig. ,

```

```

4289     Name-pl-ab = Figs. ,
4290     name-pl-ab = figs. ,
4291
4292     type = table ,
4293     gender = f ,
4294     Name-sg = Tabela ,
4295     name-sg = tabela ,
4296     Name-pl = Tabelas ,
4297     name-pl = tabelas ,
4298
4299     type = item ,
4300     gender = m ,
4301     Name-sg = Item ,
4302     name-sg = item ,
4303     Name-pl = Itens ,
4304     name-pl = itens ,
4305
4306     type = footnote ,
4307     gender = f ,
4308     Name-sg = Nota ,
4309     name-sg = nota ,
4310     Name-pl = Notas ,
4311     name-pl = notas ,
4312
4313     type = note ,
4314     gender = f ,
4315     Name-sg = Nota ,
4316     name-sg = nota ,
4317     Name-pl = Notas ,
4318     name-pl = notas ,
4319
4320     type = equation ,
4321     gender = f ,
4322     Name-sg = Equação ,
4323     name-sg = equação ,
4324     Name-pl = Equações ,
4325     name-pl = equações ,
4326     Name-sg-ab = Eq. ,
4327     name-sg-ab = eq. ,
4328     Name-pl-ab = Eqs. ,
4329     name-pl-ab = eqs. ,
4330     refpre = {(} ,
4331     refpos = {)} ,
4332
4333     type = theorem ,
4334     gender = m ,
4335     Name-sg = Teorema ,
4336     name-sg = teorema ,
4337     Name-pl = Teoremas ,
4338     name-pl = teoremas ,
4339
4340     type = lemma ,
4341     gender = m ,
4342     Name-sg = Lema ,

```

```

4343     name-sg = lema ,
4344     Name-pl = Lemas ,
4345     name-pl = lemas ,
4346
4347 type = corollary ,
4348     gender = m ,
4349     Name-sg = Corolário ,
4350     name-sg = corolário ,
4351     Name-pl = Corolários ,
4352     name-pl = corolários ,
4353
4354 type = proposition ,
4355     gender = f ,
4356     Name-sg = Proposição ,
4357     name-sg = proposição ,
4358     Name-pl = Proposições ,
4359     name-pl = proposições ,
4360
4361 type = definition ,
4362     gender = f ,
4363     Name-sg = Definição ,
4364     name-sg = definição ,
4365     Name-pl = Definições ,
4366     name-pl = definições ,
4367
4368 type = proof ,
4369     gender = f ,
4370     Name-sg = Demonstração ,
4371     name-sg = demonstração ,
4372     Name-pl = Demonstrações ,
4373     name-pl = demonstrações ,
4374
4375 type = result ,
4376     gender = m ,
4377     Name-sg = Resultado ,
4378     name-sg = resultado ,
4379     Name-pl = Resultados ,
4380     name-pl = resultados ,
4381
4382 type = remark ,
4383     gender = f ,
4384     Name-sg = Observação ,
4385     name-sg = observação ,
4386     Name-pl = Observações ,
4387     name-pl = observações ,
4388
4389 type = example ,
4390     gender = m ,
4391     Name-sg = Exemplo ,
4392     name-sg = exemplo ,
4393     Name-pl = Exemplos ,
4394     name-pl = exemplos ,
4395
4396 type = algorithm ,

```

```

4397   gender = m ,
4398   Name-sg = Algoritmo ,
4399   name-sg = algoritmo ,
4400   Name-pl = Algoritmos ,
4401   name-pl = algoritmos ,
4402
4403   type = listing ,
4404   gender = f ,
4405   Name-sg = Listagem ,
4406   name-sg = listagem ,
4407   Name-pl = Listagens ,
4408   name-pl = listagens ,
4409
4410   type = exercise ,
4411   gender = m ,
4412   Name-sg = Exercício ,
4413   name-sg = exercício ,
4414   Name-pl = Exercícios ,
4415   name-pl = exercícios ,
4416
4417   type = solution ,
4418   gender = f ,
4419   Name-sg = Solução ,
4420   name-sg = solução ,
4421   Name-pl = Soluções ,
4422   name-pl = soluções ,
4423 </dict-portuguese>

```

10.5 Spanish

```

4424 <*package>
4425 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
4426 </package>
4427 <*dict-spanish>
4428 namesep = {\nobreakspace} ,
4429 pairsep = {\~y\nobreakspace} ,
4430 listsep = { , ~ } ,
4431 lastsep = {\~y\nobreakspace} ,
4432 tpairsep = {\~y\nobreakspace} ,
4433 tlistsep = { , ~ } ,
4434 tlastsep = {\~y\nobreakspace} ,
4435 notesep = { ~ } ,
4436 rangesep = {\~a\nobreakspace} ,
4437
4438 type = part ,
4439 gender = f ,
4440 Name-sg = Parte ,
4441 name-sg = parte ,
4442 Name-pl = Partes ,
4443 name-pl = partes ,
4444
4445 type = chapter ,
4446 gender = m ,
4447 Name-sg = Capítulo ,

```

```

4448     name-sg = capítulo ,
4449     Name-pl = Capítulos ,
4450     name-pl = capítulos ,
4451
4452     type = section ,
4453     gender = f ,
4454     Name-sg = Sección ,
4455     name-sg = sección ,
4456     Name-pl = Secciones ,
4457     name-pl = secciones ,
4458
4459     type = paragraph ,
4460     gender = m ,
4461     Name-sg = Párrafo ,
4462     name-sg = párrafo ,
4463     Name-pl = Párrafos ,
4464     name-pl = párrafos ,
4465
4466     type = appendix ,
4467     gender = m ,
4468     Name-sg = Apéndice ,
4469     name-sg = apéndice ,
4470     Name-pl = Apéndices ,
4471     name-pl = apéndices ,
4472
4473     type = subappendix ,
4474     gender = m ,
4475     Name-sg = Apéndice ,
4476     name-sg = apéndice ,
4477     Name-pl = Apéndices ,
4478     name-pl = apéndices ,
4479
4480     type = page ,
4481     gender = f ,
4482     Name-sg = Página ,
4483     name-sg = página ,
4484     Name-pl = Páginas ,
4485     name-pl = páginas ,
4486     rangesep = {\textendash} ,
4487
4488     type = line ,
4489     gender = f ,
4490     Name-sg = Línea ,
4491     name-sg = línea ,
4492     Name-pl = Líneas ,
4493     name-pl = líneas ,
4494
4495     type = figure ,
4496     gender = f ,
4497     Name-sg = Figura ,
4498     name-sg = figura ,
4499     Name-pl = Figuras ,
4500     name-pl = figuras ,
4501

```

```

4502 type = table ,
4503     gender = m ,
4504     Name-sg = Cuadro ,
4505     name-sg = cuadro ,
4506     Name-pl = Cuadros ,
4507     name-pl = cuadros ,
4508
4509 type = item ,
4510     gender = m ,
4511     Name-sg = Punto ,
4512     name-sg = punto ,
4513     Name-pl = Puntos ,
4514     name-pl = puntos ,
4515
4516 type = footnote ,
4517     gender = f ,
4518     Name-sg = Nota ,
4519     name-sg = nota ,
4520     Name-pl = Notas ,
4521     name-pl = notas ,
4522
4523 type = note ,
4524     gender = f ,
4525     Name-sg = Nota ,
4526     name-sg = nota ,
4527     Name-pl = Notas ,
4528     name-pl = notas ,
4529
4530 type = equation ,
4531     gender = f ,
4532     Name-sg = Ecuación ,
4533     name-sg = ecuación ,
4534     Name-pl = Ecuaciones ,
4535     name-pl = ecuaciones ,
4536     refpre = {} ,
4537     refpos = {} ,
4538
4539 type = theorem ,
4540     gender = m ,
4541     Name-sg = Teorema ,
4542     name-sg = teorema ,
4543     Name-pl = Teoremas ,
4544     name-pl = teoremas ,
4545
4546 type = lemma ,
4547     gender = m ,
4548     Name-sg = Lema ,
4549     name-sg = lema ,
4550     Name-pl = Lemas ,
4551     name-pl = lemas ,
4552
4553 type = corollary ,
4554     gender = m ,
4555     Name-sg = Corolario ,

```

```

4556     name-sg = corolario ,
4557     Name-pl = Corolarios ,
4558     name-pl = corolarios ,
4559
4560 type = proposition ,
4561     gender = f ,
4562     Name-sg = Proposición ,
4563     name-sg = proposición ,
4564     Name-pl = Proposiciones ,
4565     name-pl = proposiciones ,
4566
4567 type = definition ,
4568     gender = f ,
4569     Name-sg = Definición ,
4570     name-sg = definición ,
4571     Name-pl = Definiciones ,
4572     name-pl = definiciones ,
4573
4574 type = proof ,
4575     gender = f ,
4576     Name-sg = Demostración ,
4577     name-sg = demostración ,
4578     Name-pl = Demostraciones ,
4579     name-pl = demostraciones ,
4580
4581 type = result ,
4582     gender = m ,
4583     Name-sg = Resultado ,
4584     name-sg = resultado ,
4585     Name-pl = Resultados ,
4586     name-pl = resultados ,
4587
4588 type = remark ,
4589     gender = f ,
4590     Name-sg = Observación ,
4591     name-sg = observación ,
4592     Name-pl = Observaciones ,
4593     name-pl = observaciones ,
4594
4595 type = example ,
4596     gender = m ,
4597     Name-sg = Ejemplo ,
4598     name-sg = ejemplo ,
4599     Name-pl = Ejemplos ,
4600     name-pl = ejemplos ,
4601
4602 type = algorithm ,
4603     gender = m ,
4604     Name-sg = Algoritmo ,
4605     name-sg = algoritmo ,
4606     Name-pl = Algoritmos ,
4607     name-pl = algoritmos ,
4608
4609 type = listing ,

```

```

4610 gender = m ,
4611 Name-sg = Listado ,
4612 name-sg = listado ,
4613 Name-pl = Listados ,
4614 name-pl = listados ,
4615
4616 type = exercise ,
4617 gender = m ,
4618 Name-sg = Ejercicio ,
4619 name-sg = ejercicio ,
4620 Name-pl = Ejercicios ,
4621 name-pl = ejercicios ,
4622
4623 type = solution ,
4624 gender = f ,
4625 Name-sg = Solución ,
4626 name-sg = solución ,
4627 Name-pl = Soluciones ,
4628 name-pl = soluciones ,
4629 </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>\!_zrefclever_current_counter_tl</code>	5
A	
<code>\AddToHook</code>	100, 759, 774, 918, 961, 986, 1024, 1026, 1076, 1149, 1165, 1186, 3065, 3067, 3072, 3085, 3105, 3120, 3122, 3127, 3143, 3156, 3176, 3188, 3220, 3234, 3270, 3274, 3295, 3301, 3324
<code>\appendix</code>	2, 80
<code>\appendixname</code>	80
B	
<code>\babelname</code>	971
<code>\babelprovide</code>	17, 29
<code>\begin</code>	83
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_if:N</code>	540, 551, 922, 926, 1592, 2007, 2096, 2226, 2248, 2267, 2273, 2277, 2323, 2372, 2395, 2399, 2405, 2415, 2421, 2536, 2543, 2550, 2553, 2574, 2603, 2606, 2641, 2654, 2662, 2665, 2720, 2722
<code>\bool_if:nTF</code>	68, 1677, 1686, 1695, 1750, 1760, 1784, 1801, 1816, 1881, 1889, 2021, 2029, 2260, 2530, 2635
<code>\bool_lazy_all:nTF</code>	2344, 2732, 2985
<code>\bool_lazy_and:nnTF</code>	1566, 1584, 2705, 2747, 3042
<code>\bool_lazy_any:nTF</code>	2865, 2874
<code>\bool_lazy_or:nnTF</code>	1570, 2693
<code>\bool_new:N</code>	489, 795, 796, 821, 845, 854, 861, 862, 875, 876, 895, 896, 951, 1084, 1085, 1086, 1087, 1088, 1179, 1180, 1600, 1613, 1921, 1922, 1932, 1939, 1940, 3233
<code>\bool_set:Nn</code>	1563
<code>\bool_set_false:N</code>	808, 812, 903, 912, 913, 928, 1096, 1100, 1107, 1115, 1116, 1117, 1202, 1742, 1976, 2013, 2027, 2041, 2238, 2370, 2371, 2872, 2889
<code>\bool_set_true:N</code>	467, 561, 802, 803,

807, 813, 902, 907, 908, 1094, 1101, 1106, 1123, 1125, 1127, 1130, 1131, 1132, 1190, 1195, 1756, 1766, 1770, 1792, 1807, 1822, 1846, 1984, 2008, 2014, 2018, 2045, 2051, 2888, 2924, 2931, 2932, 2950, 2957, 2969, 3240 \bool_until_do:Nn 1782, 1977 \bool_while_do:nn 3329	2594, 2597, 2602, 2610, 2613, 2625, 2628, 2644, 2656, 2661, 2670, 2675 \exp_not:n 255, 2125, 2141, 2153, 2158, 2181, 2195, 2199, 2211, 2215, 2249, 2250, 2280, 2292, 2297, 2298, 2435, 2448, 2455, 2479, 2491, 2495, 2505, 2509, 2537, 2544, 2546, 2551, 2554, 2557, 2559, 2563, 2592, 2593, 2595, 2596, 2598, 2604, 2607, 2611, 2612, 2614, 2622, 2626, 2627, 2629, 2642, 2655, 2657, 2663, 2666, 2669, 2671, 2676 \ExplSyntaxOn 17, 503
C	F
clist commands: \clist_map_inline:nn 1118, 1245, 3202, 3288 \counterwithin 4 \cref 82	file commands: \file_get:nnNTF 501 \fmtversion 3 \footnote 79
cs commands: \cs_generate_variant:Nn 65, 251, 257, 557, 565, 1325, 1417, 1423, 2568, 2902 \cs_if_exist:NTF 25, 28, 46, 49, 58, 78, 3093, 3099 \cs_if_exist_p:N 3331 \cs_new:Npn 56, 66, 76, 87, 252, 258, 2526, 2569, 2892 \cs_new_eq:NN 3158, 3163, 3172, 3178, 3184 \cs_new_protected:Npn .. 246, 375, 490, 558, 566, 572, 741, 1323, 1412, 1418, 1558, 1617, 1659, 1670, 1729, 1859, 1911, 1953, 2103, 2366, 2522, 2524, 2683, 2903, 2979, 3036, 3241 \cs_new_protected:Npx 99 \cs_set_eq:NN 103, 3159, 3164, 3173, 3179, 3185 \cs_set_nopar:Npn 3147	G group commands: \group_begin: 102, 322, 492, 560, 1375, 1560, 1574, 2279, 2296, 2545, 2558, 2591, 2597, 2610, 2625, 2656, 2670 \group_end: 105, 334, 555, 563, 1409, 1577, 1597, 2293, 2299, 2549, 2562, 2594, 2602, 2613, 2628, 2661, 2675
E	I
\endinput 12 exp commands: \exp_args:NNe 35, 38 \exp_args:NNNo 248 \exp_args:NNnx 343 \exp_args:NNo 248, 254 \exp_args:NNx 377, 381, 421, 507, 523, 1380, 1396 \exp_args:Nnx 568 \exp_args:No 254 \exp_args:Nx 501, 3246, 3254 \exp_args:Nxx 1713, 2916, 2938, 2942, 2959 \exp_not:N 68, 2279, 2282, 2293, 2296, 2299, 2539, 2545, 2549, 2558, 2562, 2581, 2591,	\IfBooleanTF 1603 \ifdraft 1099 \IfFormatAtLeastTF 3, 4, 3152 \ifoptionfinal 1105 \input 17 int commands: \int_case:nnTF 2106, 2134, 2166, 2326, 2428, 2467 \int_compare:nNnTF 1717, 1793, 1808, 1823, 1835, 1847, 1867, 1869, 1913, 2073, 2121, 2155, 2315, 2317, 2383, 2408, 2452, 2920, 2926, 2946, 2952, 3347 \int_compare_p:nNn 1883, 1891, 2348, 2697, 2708, 2737, 2885 \int_eval:n 99 \int_incr:N 2361, 2398, 2400, 2414, 2416, 2420, 2422, 2520, 3345 \int_new:N 1614, 1615, 1923, 1924, 1936, 1937 \int_set:Nn 1868, 1870, 1874, 1877, 3328 \int_to_roman:n 3332, 3339, 3340, 3343 \int_use:N 47, 50, 54, 60

<code>\int_zero:N</code>	1147, 1273, 1340, 1408, 1461, 1496, 1535, 2066, 2233, 2726, 2742, 3028
... 1861, 1862, 1962, 1963, 1964, 1965, 2360, 2362, 2363, 2515, 2516	<code>\msg_warning:nnnn</code> .. 395, 412, 449, 472, 1218, 1436, 1443, 1473, 2804, 2852
<code>\l_tmpa_int</code> 3328, 3332, 3339, 3340, 3343, 3345, 3347	<code>\msg_warning:nnnnn</code> 435, 479, 1455, 2763 <code>\msg_warning:nnnnnn</code> 2770
iow commands:	
<code>\iow_char:N</code> 112, 127, 128, 133, 134, 139, 140, 145, 146, 198, 206, 207, 217	
	N
	<code>\newcounter</code> 4, 3089, 3090
	<code>\NewDocumentCommand</code> 320, 337, 1321, 1326, 1373, 1556, 1601
K	<code>\nobreakspace</code> 692, 3364, 3365, 3367, 3368, 3370, 3372, 3565, 3566, 3568, 3569, 3571, 3573, 3999, 4000, 4002, 4003, 4005, 4007, 4208, 4209, 4211, 4212, 4214, 4216, 4428, 4429, 4431, 4432, 4434, 4436
keys commands:	
<code>\keys_define:nn</code> 39, 351, 578, 633, 650, 664, 748, 778, 785, 797, 822, 831, 846, 855, 863, 877, 889, 897, 930, 937, 952, 982, 1029, 1071, 1078, 1090, 1151, 1158, 1160, 1167, 1174, 1181, 1191, 1203, 1212, 1241, 1267, 1291, 1301, 1312, 1336, 1348, 1424, 1484, 1505, 1528	P
<code>\keys_set:nn</code> 14, 17, 32, 34, 39, 45, 331, 533, 1196, 1324, 1331, 1406, 1561	<code>\PackageError</code> 7
keyval commands:	<code>\pagenumbering</code> 7
<code>\keyval_parse:nnn</code> 1216, 1271	<code>\pageref</code> 46
	prg commands:
	<code>\prg_generate_conditional_-</code> variant:Nnn 714, 730
L	<code>\prg_new_protected_conditional:Npnn</code> 700, 716, 733
<code>\label</code> 82, 86	<code>\prg_return_false:</code> 710, 712, 726, 728, 739
<code>\labelformat</code> 3	<code>\prg_return_true:</code> 709, 725, 738
<code>\language</code> 29, 965	<code>\ProcessKeysOptions</code> 1320
	prop commands:
M	<code>\prop_get:NnN</code> 3005
<code>\mainbabelname</code> 29, 972	<code>\prop_get:NnNTF</code> 377, 494, 703, 706, 719, 722, 736, 1376, 2783, 2811, 2819, 2982, 3039, 3052
<code>\MessageBreak</code> 10	<code>\prop_gput:Nnn</code> 328, 344, 355, 362, 369, 1414, 1420
MH commands:	<code>\prop_gput_if_new:Nnn</code> 568, 574
<code>\MH_if_boolean:nTF</code> 3238	<code>\prop_gset_from_keyval:Nn</code> 686
msg commands:	<code>\prop_if_exist:NTF</code> 1328
<code>\msg_info:nnn</code> 615, 641, 671, 3139, 3229, 3265, 3320, 3348	<code>\prop_if_exist_p:N</code> 2989, 3045
<code>\msg_info:nnnn</code> 590, 597, 622	<code>\prop_if_in:NnTF</code> 35, 325, 341, 1007, 1052
<code>\msg_info:nnnnn</code> 609	<code>\prop_if_in_p:Nn</code> 69, 2996
<code>\msg_line_context:</code> 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>\prop_item:Nn</code> 38, 70, 345, 384, 424, 459, 510, 526, 1383, 1399
<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_new:N</code> 319, 329, 685, 1211, 1266, 1297, 1329
<code>\msg_note:nnn</code> 536	<code>\prop_put:Nnn</code> 745, 1308, 1363
<code>\msg_warning:nn</code> 764, 789, 927, 933, 1170, 1207, 2350	<code>\prop_remove:Nn</code> 744, 1307, 1355
<code>\msg_warning:nnn</code> 326, 347, 542, 552, 1012, 1057, 1074, 1136,	<code>\providecommand</code> 3
	<code>\ProvidesExplPackage</code> 14
	<code>\ProvidesFile</code> 17

R		T	
<code>\refstepcounter</code>	3, 79, 83, 84, 87	<code>\tag</code>	84, 86
<code>\renewlist</code>	87	T _E X and L ^A T _E X 2 _ε commands:	
<code>\RequirePackage</code>	16, 17, 18, 19, 20, 923, 1162, 1183	<code>\@Alph</code>	80
S		<code>\@addtoreset</code>	4
<code>\scantokens</code>	80	<code>\@auxout</code>	3250, 3258
seq commands:		<code>\@bsphack</code>	493, 3243
<code>\seq_clear:N</code>	842, 1619	<code>\@chapapp</code>	80
<code>\seq_const_from_clist:Nn</code>	265, 273, 284, 296, 299	<code>\@currentcounter</code>	3, 5, 37, 79, 84, 87, 28, 29, 49, 50, 1295
<code>\seq_gconcat:NNN</code>	306, 309, 313, 316	<code>\@currentlabel</code>	3, 84, 87
<code>\seq_get_left:NN</code>	405, 416, 520, 599, 1393, 1445, 1987	<code>\@elt</code>	5
<code>\seq_gput_right:Nn</code>	534, 545	<code>\@esphack</code>	554, 3263
<code>\seq_if_empty:N</code>	391, 431, 517, 588, 607, 1390, 1434, 1453, 1981	<code>\@ifl@t@r</code>	3
<code>\seq_if_in:NnTF</code>	409, 446, 497, 594, 619, 1247, 1440, 1465, 1663	<code>\@ifpackageloaded</code>	761, 776, 920, 963, 969, 1188, 3087, 3145, 3154, 3168, 3170, 3236, 3272, 3303, 3326
<code>\seq_map_break:n</code>	90, 1902, 1905	<code>\@onlypreamble</code>	336, 350, 1411
<code>\seq_map_function:NN</code>	1622	<code>\@raw@opt@<package>.sty</code>	34
<code>\seq_map_indexed_inline:Nn</code>	26, 1863	<code>\bbl@loaded</code>	29
<code>\seq_map_inline:Nn</code>	630, 647, 661, 1298, 1333, 1345, 1481, 1502, 1525, 1899, 3244	<code>\bbl@main@language</code>	29, 966
<code>\seq_map_tokens:Nn</code>	72	<code>\c@</code>	4
<code>\seq_new:N</code>	263, 264, 305, 312, 488, 830, 1240, 1599, 1616, 1920	<code>\c@enumN</code>	87
<code>\seq_pop_left:NN</code>	1979	<code>\c@lstnumber</code>	87
<code>\seq_put_right:Nn</code>	1249, 1666	<code>\c@page</code>	7, 103
<code>\seq_reverse:N</code>	836	<code>\cl@</code>	5
<code>\seq_set_eq:NN</code>	1955	<code>\hyper@link</code>	68, 2282, 2539, 2581, 2644
<code>\seq_set_from_clist:Nn</code>	381, 421, 507, 523, 835, 1380, 1396, 1562	<code>\lst@AddToHook</code>	3314
<code>\seq_sort:Nn</code>	48, 1625	<code>\lst@label</code>	3316, 3317
<code>\setcounter</code>	3091, 3092, 3108, 3121, 3125	<code>\ltx@gobble</code>	82
sort commands:		<code>\ltx@label</code>	82, 3158, 3159, 3163, 3164, 3172, 3173, 3178, 3179, 3184, 3185
<code>\sort_return_same:</code>	48, 53, 1632, 1637, 1684, 1722, 1724, 1757, 1777, 1798, 1813, 1827, 1852, 1887, 1902, 1918	<code>\MT@newlabel</code>	3251, 3259
<code>\sort_return_swapped:</code>	48, 53, 1645, 1693, 1721, 1767, 1776, 1797, 1812, 1828, 1851, 1895, 1905, 1917	<code>\p@...</code>	3
<code>\stepcounter</code>	3107, 3124	<code>\protected@write</code>	3250, 3258
str commands:		<code>\zref@addprop</code>	22, 32, 43, 53, 55, 97, 108
<code>\str_case:nnTF</code>	988, 1033, 1120	<code>\zref@default</code>	68, 2523, 2525
<code>\str_compare:nNnTF</code>	1773	<code>\zref@extractdefault</code>	10, 75, 249, 255, 259
<code>\str_if_eq:nnTF</code>	89, 457	<code>\zref@ifpropundefined</code>	24, 2894
<code>\str_if_eq_p:nn</code>	2870, 2876, 2878, 2882	<code>\zref@ifrefcontainsprop</code>	24, 2528, 2576, 2631, 2897
<code>\str_new:N</code>	936	<code>\zref@ifrefundefined</code>	1627, 1629, 1641, 2010, 2012, 2017, 2061, 2230, 2239, 2374, 2571, 2685
<code>\str_set:Nn</code>	941, 943, 945, 947	<code>\zref@label</code>	82, 3150
<code>\string</code>	3251, 3259	<code>\ZREF@mainlist</code>	22, 32, 43, 53, 55, 97, 108
		<code>\zref@newprop</code>	5, 7, 21, 23, 33, 44, 54, 92, 107
		<code>\zref@refused</code>	2060
		<code>\zref@wrapper@babel</code>	44, 82, 1557, 3150
		<code>\textendash</code>	696, 3421, 3679, 4057, 4272, 4486
		<code>\textup</code>	84, 3226, 3227
		<code>\the</code>	3

`\l_zrefclever_counter_resetby_`
`prop` 5, 36, 69, 70, 1266, 1278
`\l_zrefclever_counter_resettters_`
`seq` . 5, 36, 37, 72, 1240, 1247, 1250
`\l_zrefclever_counter_type_prop`
. 4, 35, 35, 38, 1211, 1223
`\l_zrefclever_current_counter_`
`tl` 3, 37, 21, 25,
26, 36, 39, 41, 46, 47, 95, 1290, 1293
`\l_zrefclever_current_language_`
`tl` 29, 960, 965, 971, 975, 1001, 1046
`_zrefclever_declare_default_`
`transl:nnn` . . . 41, 1412, 1491, 1512
`_zrefclever_declare_type_`
`transl:nnnn`
. . . . 41, 1412, 1467, 1517, 1541, 1547
`_zrefclever_def_extract:Nnnn` . .
. 10,
246, 1661, 1672, 1674, 1731, 1734,
1737, 1739, 1997, 1999, 2905, 2907
`\g_zrefclever_dict_(language)_prop`
. 17
`\l_zrefclever_dict_decl_case_tl`
. . . . 260, 518, 521, 595, 600, 675,
679, 1391, 1394, 1441, 1446, 1539, 1550
`\l_zrefclever_dict_declension_`
`seq` 260, 382, 391, 405,
409, 416, 508, 517, 520, 588, 594,
599, 1381, 1390, 1393, 1434, 1440, 1445
`\l_zrefclever_dict_gender_seq` . .
. 260, 422, 431,
446, 524, 607, 619, 1397, 1453, 1465
`\l_zrefclever_dict_language_tl` .
. 260,
330, 356, 363, 370, 379, 387, 427,
462, 495, 499, 502, 513, 529, 535,
537, 543, 546, 569, 575, 591, 598,
610, 623, 704, 707, 720, 723, 1377,
1386, 1402, 1437, 1444, 1456, 1468,
1474, 1492, 1513, 1518, 1542, 1548
`_zrefclever_extract:nnn`
. . 10, 258, 1718, 1720, 1794, 1796,
1809, 1826, 1914, 1916, 2921, 2923,
2927, 2929, 2947, 2949, 2953, 2955
`_zrefclever_extract_unexp:nnn` .
. 10, 75, 252,
1714, 1715, 2288, 2541, 2547, 2560,
2587, 2599, 2650, 2658, 2672, 2895,
2898, 2899, 2917, 2918, 2939, 2940,
2943, 2944, 2961, 2965, 3247, 3255
`_zrefclever_extract_url_`
`unexp:n` 2284, 2540, 2583, 2646, 2892
`\g_zrefclever_fallback_dict_`
`prop` 11, 685, 686, 736
`\l_zrefclever_footnote_type_tl` .
. 3063, 3064, 3066, 3070
`_zrefclever_get_default_`
`transl:nnN` 11, 717, 731
`_zrefclever_get_default_`
`transl:nnNTF` 23, 716, 3020
`_zrefclever_get_enclosing_`
`counters_value:n` . 5, 6, 56, 61, 94
`_zrefclever_get_fallback_`
`transl:nN` 734
`_zrefclever_get_fallback_`
`transl:nNTF` 24, 732, 3025
`_zrefclever_get_ref:n`
. 67, 68, 2126, 2142,
2154, 2159, 2182, 2196, 2200, 2212,
2216, 2251, 2270, 2436, 2449, 2456,
2480, 2492, 2496, 2506, 2510, 2526
`_zrefclever_get_ref_first:` . . .
. 67, 68, 71, 2264, 2309, 2569
`_zrefclever_get_ref_font:nN` . . .
. . 11, 22, 38, 77, 79, 2089, 2091, 3036
`_zrefclever_get_ref_string:nN` .
. 11, 22, 38, 77,
1580, 1968, 1970, 1972, 2075, 2077,
2079, 2081, 2083, 2085, 2087, 2979
`_zrefclever_get_type_transl:nnnN`
. 11, 701, 715
`_zrefclever_get_type_transl:nnnNTF`
23, 700, 2751, 2797, 2839, 2845, 3014
`\l_zrefclever_label_a_tl`
. 54, 1925, 1980, 1998, 2010, 2060,
2061, 2067, 2113, 2126, 2142, 2159,
2200, 2216, 2244, 2251, 2374, 2378,
2388, 2413, 2436, 2457, 2496, 2510
`\l_zrefclever_label_b_tl`
. 54, 1925,
1983, 1988, 2000, 2012, 2017, 2378
`\l_zrefclever_label_count_int` . .
. 55, 1923, 1962,
2073, 2106, 2360, 2383, 2520, 2738
`\l_zrefclever_label_enclval_a_`
`tl` 1607, 1731, 1733, 1788,
1804, 1824, 1836, 1840, 1841, 1848
`\l_zrefclever_label_enclval_b_`
`tl` 1607, 1734, 1736, 1789,
1811, 1819, 1838, 1842, 1843, 1850
`\l_zrefclever_label_extdoc_a_tl`
. 1607, 1737,
1747, 1752, 1762, 1775, 2905, 2911
`\l_zrefclever_label_extdoc_b_tl`
. 1607, 1739,
1748, 1753, 1763, 1774, 2907, 2912
`\l_zrefclever_label_type_a_tl` . .
. 77, 1607, 1662, 1664,

1667, 1673, 1681, 1690, 1698, 1703,
 1873, 1901, 1993, 1997, 2024, 2032,
 2038, 2064, 2115, 2390, 2987, 2992,
 2999, 3008, 3016, 3043, 3048, 3055
 \l_zrefclever_label_type_b_tl ..
 1607,
 1675, 1682, 1691, 1699, 1704, 1876,
 1904, 1994, 1999, 2025, 2034, 2039
 _zrefclever_label_type_put_-
 new_right:n 47, 48, 1623, 1659
 \l_zrefclever_label_types_seq ..
 48, 1616, 1619, 1663, 1666, 1899
 _zrefclever_labels_in_sequence:nn
 55, 76, 2242, 2377, 2903
 \g_zrefclever_languages_prop ...
 . 13, 319, 325, 328, 341, 344, 345,
 377, 494, 703, 719, 1007, 1052, 1376
 \l_zrefclever_last_of_type_bool
 54, 1920, 2008, 2013, 2014,
 2018, 2027, 2042, 2046, 2052, 2096
 \l_zrefclever_lastsep_tl . 1941,
 2084, 2141, 2158, 2181, 2199, 2211
 \l_zrefclever_link_star_bool ...
 1563, 1599, 2533, 2638, 2868
 \l_zrefclever_listsep_tl
 ... 1941, 2082, 2153, 2195, 2435,
 2448, 2455, 2479, 2491, 2495, 2505
 \l_zrefclever_load_dict_-
 verbose_bool ... 489, 540, 551, 561
 \g_zrefclever_loaded_dictionaries_-
 seq 488, 498, 534, 545
 _zrefclever_ltxlabel:n
 82, 3147, 3159, 3164, 3173, 3179, 3185
 \l_zrefclever_main_language_tl .
 29, 959, 966,
 972, 976, 980, 993, 1015, 1038, 1060
 _zrefclever_mathtools_showonlyrefs:n
 1594, 3241
 \l_zrefclever_mathtools_-
 showonlyrefs_bool 1592, 3233, 3240
 _zrefclever_name_default:
 2522, 2621
 \l_zrefclever_name_format_-
 fallback_tl 1931, 2713,
 2717, 2781, 2824, 2834, 2836, 2848
 \l_zrefclever_name_format_tl ...
 ... 1931, 2699, 2700, 2703, 2704,
 2714, 2715, 2788, 2793, 2794, 2800,
 2805, 2816, 2830, 2831, 2842, 2854
 \l_zrefclever_name_in_link_bool
 69,
 71, 1931, 2277, 2574, 2872, 2888, 2889
 \l_zrefclever_namefont_tl 1941,
 2090, 2280, 2297, 2592, 2611, 2626
 \l_zrefclever_nameinlink_str ...
 936, 941,
 943, 945, 947, 2870, 2876, 2878, 2882
 \l_zrefclever_namesep_tl
 ... 1941, 2076, 2595, 2614, 2622, 2629
 \l_zrefclever_next_is_same_bool
 55, 76, 1936,
 2371, 2399, 2415, 2421, 2932, 2970
 \l_zrefclever_next_maybe_range_-
 bool
 ... 55, 76, 1936, 2238, 2248, 2370,
 2395, 2405, 2924, 2931, 2950, 2958
 \l_zrefclever_noabbrev_first_-
 bool 876, 885, 2710
 \l_zrefclever_nudge_comptosing_-
 bool ... 1086, 1116, 1125, 1131, 2734
 \l_zrefclever_nudge_enabled_-
 bool 1084, 1094, 1096,
 1100, 1101, 1106, 1107, 2346, 2720
 \l_zrefclever_nudge_gender_bool
 1088, 1117, 1127, 1132, 2748
 \l_zrefclever_nudge_multitype_-
 bool ... 1085, 1115, 1123, 1130, 2347
 \l_zrefclever_nudge_singular_-
 bool 1087, 1143, 2722
 _zrefclever_orig_ltxlabel:n ...
 ... 3149, 3158, 3163, 3172, 3178, 3184
 _zrefclever_page_format_aux: ..
 99, 103
 \g_zrefclever_page_format_tl ...
 7, 98, 104, 107
 \l_zrefclever_pairsep_tl
 1941, 2080, 2125, 2249
 \l_zrefclever_preposinlink_bool
 . 951, 954, 2536, 2543, 2550, 2553,
 2603, 2606, 2641, 2654, 2662, 2665
 _zrefclever_process_language_-
 options: 32, 34, 375, 1565
 _zrefclever_prop_put_non_-
 empty:Nnn 24, 741, 1222, 1277
 _zrefclever_provide_dict_-
 default_transl:nn 20, 566, 639, 656
 _zrefclever_provide_dict_type_-
 transl:nn 20, 566, 620, 657, 676, 678
 _zrefclever_provide_dictionary:n
 11, 17-20, 45,
 490, 562, 1028, 1039, 1047, 1062, 1564
 _zrefclever_provide_dictionary_-
 verbose:n . 19, 558, 994, 1002, 1017
 \l_zrefclever_range_beg_label_-
 tl 55, 1936, 1961,
 2154, 2177, 2183, 2193, 2197, 2209,
 2213, 2359, 2397, 2412, 2446, 2450,
 2477, 2481, 2489, 2493, 2503, 2507

`\l_zrefclever_range_count_int` 55,
1936, 1964, 2134, 2168, 2362, 2398,
2409, 2414, 2420, 2428, 2469, 2515
`\l_zrefclever_range_same_count_-`
`int` 55,
1936, 1965, 2121, 2156, 2169, 2363,
2400, 2416, 2422, 2453, 2470, 2516
`\l_zrefclever_rangesep_tl`
. 1941, 2078, 2215, 2250, 2509
`\l_zrefclever_ref_decl_case_tl`
. 14, 393, 398, 399, 403, 406,
410, 414, 417, 470, 473, 475, 1070,
1080, 2791, 2795, 2827, 2832, 2837
`_zrefclever_ref_default:`
. 2522, 2566, 2572, 2615, 2679
`\l_zrefclever_ref_gender_tl`
. 15, 433, 439,
440, 444, 447, 452, 453, 477, 483,
484, 1089, 1153, 2749, 2758, 2765, 2773
`\l_zrefclever_ref_language_tl`
. 14, 29, 30, 378, 397,
415, 437, 451, 474, 481, 958, 979,
992, 995, 1000, 1003, 1009, 1014,
1018, 1028, 1037, 1040, 1045, 1048,
1054, 1059, 1063, 1564, 2752, 2767,
2775, 2798, 2840, 2846, 3015, 3021
`\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, 631, 1334, 1482
`\c_zrefclever_ref_options_-`
`possibly_type_specific_seq`
. 22, 265, 648, 1503
`\l_zrefclever_ref_options_prop`
. 38, 39, 1297, 1307, 1308, 2982, 3039
`\c_zrefclever_ref_options_-`
`reference_seq` 265, 1299
`\c_zrefclever_ref_options_type_-`
`names_seq` 265, 662, 1526
`\c_zrefclever_ref_options_-`
`typesetup_seq` 265, 1346
`\l_zrefclever_ref_property_tl`
. 24, 747, 752,
754, 756, 762, 765, 781, 790, 1620,
1652, 1991, 2528, 2578, 2633, 2914
`\l_zrefclever_ref_typeset_font_-`
`tl` 1157, 1159, 1575
`\l_zrefclever_reffont_tl` 1941,
2092, 2546, 2559, 2598, 2657, 2671
`\l_zrefclever_refpos_tl`
. 1941, 2088, 2551, 2554,
2563, 2604, 2607, 2663, 2666, 2676
`\l_zrefclever_refpre_tl`
. 1941, 2086, 2537,
2544, 2557, 2596, 2642, 2655, 2669
`\l_zrefclever_setup_type_tl`
. 20, 260, 506, 570, 583,
584, 613, 638, 655, 669, 1330, 1358,
1366, 1379, 1429, 1430, 1459, 1469,
1489, 1510, 1519, 1533, 1543, 1549
`\l_zrefclever_sort_decided_bool`
. 1613, 1742, 1756, 1766,
1770, 1782, 1792, 1807, 1822, 1846
`_zrefclever_sort_default:nn`
. 49, 1654, 1670
`_zrefclever_sort_default_-`
`different_types:nn`
. 26, 47, 52, 1708, 1859
`_zrefclever_sort_default_same_-`
`type:nn` 47, 50, 1706, 1729
`_zrefclever_sort_labels:`
. 47, 48, 53, 1573, 1617
`_zrefclever_sort_page:nn`
. 53, 1653, 1911
`\l_zrefclever_sort_prior_a_int`
. 1614,
1861, 1867, 1868, 1874, 1884, 1892
`\l_zrefclever_sort_prior_b_int`
. 1614,
1862, 1869, 1870, 1877, 1885, 1893
`\l_zrefclever_tlastsep_tl`
. 1941, 1973, 2340
`\l_zrefclever_tlistsep_tl`
. 1941, 1971, 2318
`\l_zrefclever_tpairsep_tl`
. 1941, 1969, 2334
`\l_zrefclever_type_<type>-`
`options_prop` 39
`\l_zrefclever_type_count_int`
. 55, 71, 1923, 1963, 2315, 2317,
2326, 2348, 2361, 2697, 2709, 2885
`\l_zrefclever_type_first_label_-`
`tl` 54, 69, 1925, 1959, 2112, 2230,
2239, 2243, 2270, 2285, 2289, 2357,
2387, 2571, 2577, 2584, 2588, 2600,
2632, 2647, 2651, 2659, 2673, 2685
`\l_zrefclever_type_first_label_-`
`type_tl` 54, 71, 1925, 1960,
2114, 2234, 2358, 2389, 2688, 2728,
2744, 2753, 2766, 2772, 2786, 2799,
2806, 2814, 2822, 2841, 2847, 2855
`\l_zrefclever_type_name_gender_-`
`tl` 1931, 2755, 2756, 2759, 2761, 2774

_zrefclever_type_name_setup: ..	_zrefclever_typeset_refs_last_-
..... 11 , 69 , 2259 , 2683	of_type: . 59 , 67 , 69 , 71 , 2098 , 2103
\l_zrefclever_type_name_tl	_zrefclever_typeset_refs_not_-
..... 69 , 71 ,	last_of_type:
1931 , 2292 , 2298 , 2593 , 2612 , 2619 , 55 , 59 , 67 , 76 , 2100 , 2366
2627 , 2686 , 2689 , 2789 , 2801 , 2803 ,	\l_zrefclever_typeset_sort_bool
2817 , 2825 , 2843 , 2849 , 2851 , 2869 821 , 824 , 1571
\l_zrefclever_typeset_compress_-	\l_zrefclever_typeset_sort_seq
bool 845 , 848 , 2372 26 , 52 , 830 , 835 , 836 , 842 , 1863
\l_zrefclever_typeset_labels_-	\l_zrefclever_use_hyperref_bool
seq 54 , 1920 , 1955 , 1979 , 1981 , 1987 895 , 902 ,
\l_zrefclever_typeset_last_bool	907 , 912 , 922 , 928 , 2532 , 2637 , 2867
..... 54 , 1920 ,	\l_zrefclever_warn_hyperref_-
1976 , 1977 , 1984 , 2007 , 2323 , 2884	bool 896 , 903 , 908 , 913 , 926
\l_zrefclever_typeset_name_bool	_zrefclever_zceref:nnn 14 , 1557 , 1558
..... 796 , 803 , 808 , 813 , 2261 , 2273	_zrefclever_zceref:nnnn 44 , 47 , 1558
\l_zrefclever_typeset_queue_-	\l_zrefclever_zceref_labels_seq .
curr_tl 54 , 67 , 47 , 48 , 1562 ,
71 , 1925 , 1958 , 2123 , 2139 , 2148 ,	1590 , 1595 , 1599 , 1622 , 1625 , 1956
2179 , 2190 , 2206 , 2228 , 2246 , 2263 ,	\l_zrefclever_zceref_note_tl ...
2269 , 2275 , 2308 , 2330 , 2335 , 2341 , 1173 , 1176 , 1578 , 1582
2355 , 2356 , 2433 , 2444 , 2475 , 2487 ,	\l_zrefclever_zceref_with_check_-
2501 , 2702 , 2724 , 2735 , 2879 , 2883	bool 1180 , 1195 , 1568 , 1586
\l_zrefclever_typeset_queue_-	_zrefclever_zcsetup:n
prev_tl . 54 , 1925 , 1957 , 2319 , 2354 39 , 1322 , 1323 , 3069 ,
\l_zrefclever_typeset_range_-	3074 , 3095 , 3101 , 3109 , 3129 , 3190 ,
bool 854 , 857 , 1572 , 2226	3221 , 3276 , 3296 , 3305 , 3318 , 3335
\l_zrefclever_typeset_ref_bool .	\l_zrefclever_zrefcheck_-
..... 795 , 802 , 807 , 812 , 2261 , 2267	available_bool
_zrefclever_typeset_refs: 1179 , 1190 , 1202 , 1567 , 1585
..... 54-56 , 1576 , 1953	