

# The zref-clever package implementation\*

Gustavo Barros<sup>†</sup>

2021-09-29

## Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>3</b>
<b>3</b>	<b>zref setup</b>	<b>3</b>
<b>4</b>	<b>Plumbing</b>	<b>7</b>
4.1	Messages . . . . .	7
4.2	Data extraction . . . . .	10
4.3	Reference format . . . . .	10
4.4	Languages . . . . .	12
4.5	Dictionaries . . . . .	13
4.6	Options . . . . .	20
<b>5</b>	<b>Configuration</b>	<b>36</b>
5.1	\zcsetup . . . . .	36
5.2	\zcRefTypeSetup . . . . .	36
5.3	\zcLanguageSetup . . . . .	37
<b>6</b>	<b>User interface</b>	<b>41</b>
6.1	\zceref . . . . .	41
6.2	\zcpageref . . . . .	42
<b>7</b>	<b>Sorting</b>	<b>43</b>
<b>8</b>	<b>Typesetting</b>	<b>50</b>

---

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

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

<b>9</b>	<b>Compatibility</b>	<b>76</b>
9.1	<code>\footnote</code> . . . . .	76
9.2	<code>\appendix</code> . . . . .	76
9.3	<code>appendix</code> package . . . . .	77
9.4	<code>amsmath</code> package . . . . .	78
9.5	<code>mathtools</code> package . . . . .	81
9.6	<code>breqn</code> package . . . . .	82
9.7	<code>listings</code> package . . . . .	83
9.8	<code>enumitem</code> package . . . . .	83
<b>10</b>	<b>Dictionaries</b>	<b>84</b>
10.1	English . . . . .	84
10.2	German . . . . .	88
10.3	French . . . . .	100
10.4	Portuguese . . . . .	103
10.5	Spanish . . . . .	107
	<b>Index</b>	<b>110</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L<sup>A</sup>T<sub>E</sub>X3 DocStrip convention).

```
2 <@@=zrefclever>
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`, even though I'd have loved to have used `\bool_case_true:...`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the dictionaries (which became the default input encoding in the 2018-04-01 release). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut with the inclusion of the new hook management system (`ltxcmdhooks`), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

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

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-06-01}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
9     'zref-clever' requires a LaTeX kernel newer than 2021-06-01.%
10    \MessageBreak Loading will abort!%
11   }%
12 \endinput
13 }%
```

Identify the package.

```

14 \ProvidesExplPackage {zref-clever} {2021-09-29} {0.1.0-alpha}
15 {Clever LaTeX cross-references based on zref}

```

## 2 Dependencies

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

```

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

```

## 3 zref setup

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

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

`\l__zrefclever_get_enclosing_counters_value:n`

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

```

\l__zrefclever_get_enclosing_counters_value:n {<counter>}

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

```

```

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

```

Both `e` and `f` expansions work for this particular recursive call. I'll stay with the `e` variant, since conceptually it is what I want (`x` itself is not expandable), and this package is anyway not compatible with older kernels for which the performance penalty of the `e` expansion would ensue (see also [https://tex.stackexchange.com/q/611370/#comment1529282\\_611385](https://tex.stackexchange.com/q/611370/#comment1529282_611385), thanks Enrico Gregorio, aka 'egreg').

```

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

```

(End definition for `\_zrefclever_get_enclosing_counters_value:n`.)

`\_zrefclever_counter_reset_by:n`

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

```

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

```

(End definition for `\_zrefclever_counter_reset_by:n`.)

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

```

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

```

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

```

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

```

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

## 4 Plumbing

### 4.1 Messages

```

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

```

```

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

```



```

173 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
174 {
175   Option~'sg'~signals-that-a-singular-type-name-was-expected~
176   \msg_line_context:..~But~type~'#1'~has~plural~type~name.
177 }
178 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
179 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
180 \msg_new:nnn { zref-clever } { option-document-only }
181 { Option~'#1'~is-only-available-after~\iow_char:N\\begin\{document\}. }
182 \msg_new:nnn { zref-clever } { dict-loaded }
183 { Loaded~'#1'~dictionary. }
184 \msg_new:nnn { zref-clever } { dict-not-available }
185 { Dictionary~for~'#1'~not~available~\msg_line_context:. }
186 \msg_new:nnn { zref-clever } { unknown-language-load }
187 {
188   Language~'#1'~is~unknown~\msg_line_context:..~Unable~to~load~dictionary.~
189   See~documentation~for~'\iow_char:N\\zcDeclareLanguage'~and~
190   '\iow_char:N\\zcDeclareLanguageAlias'.
191 }
192 \msg_new:nnn { zref-clever } { missing-zref-titleref }
193 {
194   Option~'ref=title'~requested~\msg_line_context:..~
195   But~package~'zref-titleref'~is~not~loaded,~falling-back~to~default~'ref'.
196 }
197 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
198 {
199   Option~'hyperref'~only~available-in~the~preamble~\msg_line_context:..~
200   Use~the~starred~version~of~'\iow_char:N\\zcref'~instead.
201 }
202 \msg_new:nnn { zref-clever } { missing-hyperref }
203 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
204 \msg_new:nnn { zref-clever } { titleref-preamble-only }
205 {
206   Option~'titleref'~only~available-in~the~preamble~\msg_line_context:..~
207   Did~you~mean~'ref=title'?.
208 }
209 \msg_new:nnn { zref-clever } { missing-zref-check }
210 {
211   Option~'check'~requested~\msg_line_context:..~
212   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
213 }
214 \msg_new:nnn { zref-clever } { missing-type }
215 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
216 \msg_new:nnn { zref-clever } { missing-name }
217 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
218 \msg_new:nnn { zref-clever } { missing-string }
219 {
220   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:..~
221   But~we~should~have:~throw~a~rock~at~the~maintainer.
222 }
223 \msg_new:nnn { zref-clever } { single-element-range }
224 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
225 \msg_new:nnn { zref-clever } { compat-package }
226 { Loaded~support~for~'#1'~package. }

```

```

227 \msg_new:nnn { zref-clever } { compat-class }
228 { 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}\

229 \cs_new_protected:Npn \__zrefclever_def_extract:Nnnn #1#2#3#4
230 {
231   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
232   { \zref@extractdefault {#2} {#3} {#4} }
233 }
234 \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}\

235 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
236 {
237   \exp_args:NNNo \exp_args:No
238   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
239 }
240 \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}\

241 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
242 { \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 translation handling, notably in `\__zrefclever_provide_dictionary:n`, `\zcRefTypeSetup`, and `\zcLanguageSetup`. But also for translations retrieval, in `\__zrefclever_get_type_transl:nnnN` and `\__zrefclever_get_default_transl:nnN`.

```

243 \tl_new:N \l__zrefclever_setup_type_tl
244 \tl_new:N \l__zrefclever_dict_language_tl
245 \tl_new:N \l__zrefclever_dict_decl_case_tl
246 \seq_new:N \l__zrefclever_dict_declension_seq

```

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

`f_options_necessarily_not_type_specific_seq` 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_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq
247 \seq_const_from_clist:Nn
248 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
249 {
250     tpairsep ,
251     tlistsep ,
252     tlastsep ,
253     notesep ,
254 }
255 \seq_const_from_clist:Nn
256 \c__zrefclever_ref_options_possibly_type_specific_seq
257 {
258     namesep ,
259     pairsep ,
260     listsep ,
261     lastsep ,
262     rangesep ,
263     refpre ,
264     refpos ,
265     refpre-in ,
266     refpos-in ,
267 }

```

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:.`

```

268 \seq_const_from_clist:Nn
269 \c__zrefclever_ref_options_type_names_seq
270 {
271     Name-sg ,
272     name-sg ,
273     Name-pl ,
274     name-pl ,
275     Name-sg-ab ,
276     name-sg-ab ,
277     Name-pl-ab ,
278     name-pl-ab ,
279 }

```

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

```

280 \seq_const_from_clist:Nn
281 \c__zrefclever_ref_options_font_seq
282 {
283     namefont ,
284     reffont ,
285     reffont-in ,
286 }
287 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
288 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
289 \c__zrefclever_ref_options_possibly_type_specific_seq
290 \c__zrefclever_ref_options_type_names_seq
291 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
292 \c__zrefclever_ref_options_typesetup_seq
293 \c__zrefclever_ref_options_font_seq
294 \seq_new:N \c__zrefclever_ref_options_reference_seq
295 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
296 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
297 \c__zrefclever_ref_options_possibly_type_specific_seq
298 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
299 \c__zrefclever_ref_options_reference_seq
300 \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 or 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”.

```

301 \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”. Optional argument  $[\langle cases \rangle]$  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. If  $\langle language \rangle$  is already known, just warn. This implies a particular restriction regarding  $[\langle cases \rangle]$ , namely, that the declension of languages 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 [ $\langle cases \rangle$ ] { $\langle language \rangle$ }

302 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
303 {
304     \tl_if_empty:nF {#2}
305     {
306         \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
307         { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
308         {

```

```

309         \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
310     \tl_if_empty:nF {#1}
311     {
312         \prop_if_exist:cF
313         { g__zrefclever_dict_ #2 _prop }
314         { \prop_new:c { g__zrefclever_dict_ #2 _prop } }
315         \prop_put:cnn
316         { g__zrefclever_dict_ #2 _prop } { declension } {#1}
317     }
318 }
319 }
320 }
321 \@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}

322 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
323 {
324     \tl_if_empty:nF {#1}
325     {
326         \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
327         {
328             \exp_args:NnNx
329             \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
330             { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
331         }
332         { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
333     }
334 }
335 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

## 4.5 Dictionaries

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

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

<b>\g_zrefclever_loaded_dictionaries_seq</b>	Used to keep track of whether a dictionary has already been loaded or not.
336 <b>\seq_new:N \g__zrefclever_loaded_dictionaries_seq</b>	
(End definition for <b>\g__zrefclever_loaded_dictionaries_seq</b> .)	
<b>\l_zrefclever_load_dict_verbose_bool</b>	Controls whether <b>\__zrefclever_provide_dictionary:n</b> fails silently or verbosely in case of unknown languages or dictionaries not found.
337 <b>\bool_new:N \l__zrefclever_load_dict_verbose_bool</b>	
(End definition for <b>\l__zrefclever_load_dict_verbose_bool</b> .)	
<b>\__zrefclever_provide_dictionary:n</b>	Load dictionary for known <i>⟨language⟩</i> if it is available and if it has not already been loaded.
<b>\__zrefclever_provide_dictionary:n {⟨language⟩}</b>	
338 <b>\cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1</b>	
339 <b>{</b>	
340 <b>\group_begin:</b>	
341 <b>\@bsphack</b>	
342 <b>\prop_get:NnNTF \g__zrefclever_languages_prop {#1}</b>	

```

343 \l__zrefclever_dict_language_tl
344 {
345   \seq_if_in:NVF
346     \g__zrefclever_loaded_dictionaries_seq
347     \l__zrefclever_dict_language_tl
348   {
349     \exp_args:Nx \file_get:nnNTF
350     { zref-clever- \l__zrefclever_dict_language_tl .dict }
351     { \ExplSyntaxOn }
352     \l_tmpa_tl
353     {
354       \prop_if_exist:cF
355       {
356         g__zrefclever_dict_
357         \l__zrefclever_dict_language_tl _prop
358       }
359       {
360         \prop_new:c
361         {
362           g__zrefclever_dict_
363           \l__zrefclever_dict_language_tl _prop
364         }
365       }
366       \tl_clear:N \l__zrefclever_setup_type_tl
367       \exp_args:NNx \seq_set_from_clist:Nn
368       \l__zrefclever_dict_declension_seq
369       {
370         \prop_item:cn
371         {
372           g__zrefclever_dict_
373           \l__zrefclever_dict_language_tl _prop
374         }
375         { declension }
376       }
377       \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
378       { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
379       {
380         \seq_get_left:NN \l__zrefclever_dict_declension_seq
381         \l__zrefclever_dict_decl_case_tl
382       }
383       \keys_set:nV { zref-clever / dictionary } \l_tmpa_tl
384       \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
385       \l__zrefclever_dict_language_tl
386       \msg_note:nmx { zref-clever } { dict-loaded }
387       { \l__zrefclever_dict_language_tl }
388     }
389   {
390     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
391     {
392       \msg_warning:nmx { zref-clever } { dict-not-available }
393       { \l__zrefclever_dict_language_tl }
394     }
395   }

```

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.

```

395             \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
396             \l__zrefclever_dict_language_tl
397         }
398     }
399 }
400 {
401     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
402     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
403 }
404 \@esphack
405 \group_end:
406 }
407 \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>}

408 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
409 {
410     \group_begin:
411     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
412     \__zrefclever_provide_dictionary:n {#1}
413     \group_end:
414 }
415 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for `\__zrefclever_provide_dictionary_verbose:n`.)

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

```

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

416 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
417 {
418     \exp_args:Nnx \prop_gput_if_new:cnn
419     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
420     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
421 }
422 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
423 {

```



```

424 \prop_gput_if_new:cnn
425 { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
426 { default- #1 } {#2}
427 }

```

(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.

```

428 \keys_define:nn { zref-clever / dictionary }
429 {
430   type .code:n =
431   {
432     \tl_if_empty:NTF {#1}
433     { \tl_clear:N \l__zrefclever_setup_type_tl }
434     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
435   } ,
436   case .code:n =
437   {
438     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
439     {
440       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
441       { \l__zrefclever_dict_language_tl } {#1}
442     }
443     {
444       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
445       { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
446       {
447         \msg_info:nxxx { zref-clever } { unknown-decl-case }
448         {#1} { \l__zrefclever_dict_language_tl }
449         \seq_get_left:NN \l__zrefclever_dict_declension_seq
450         \l__zrefclever_dict_decl_case_tl
451       }
452     }
453   } ,
454   case .value_required:n = true ,
455 }
456 \seq_map_inline:Nn
457 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
458 {
459   \keys_define:nn { zref-clever / dictionary }
460   {
461     #1 .value_required:n = true ,
462     #1 .code:n =
463     {
464       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
465       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
466       {
467         \msg_info:nnn { zref-clever }
468         { option-not-type-specific } {#1}
469       }

```

```

470     } ,
471   }
472 }
473 \seq_map_inline:Nn
474 \c__zrefclever_ref_options_possibly_type_specific_seq
475 {
476   \keys_define:nn { zref-clever / dictionary }
477   {
478     #1 .value_required:n = true ,
479     #1 .code:n =
480     {
481       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
482       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
483       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
484     } ,
485   }
486 }
487 \seq_map_inline:Nn
488 \c__zrefclever_ref_options_type_names_seq
489 {
490   \keys_define:nn { zref-clever / dictionary }
491   {
492     #1 .value_required:n = true ,
493     #1 .code:n =
494     {
495       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
496       {
497         \msg_info:nnn { zref-clever }
498         { option-only-type-specific } {#1}
499       }
500       {
501         \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
502         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
503         {
504           \__zrefclever_provide_dict_type_transl:nn
505           { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
506         }
507       }
508     } ,
509   }
510 }

```

## 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.

```

511 \prop_new:N \g__zrefclever_fallback_dict_prop
512 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
513 {
514     tpairsep = {,~} ,
515     tlistsep = {,~} ,
516     tlastsep = {,~} ,
517     notesep  = {~} ,
518     namesep  = {\nobreakspace} ,
519     pairsep  = {,~} ,
520     listsep  = {,~} ,
521     lastsep  = {,~} ,
522     rangesep = {\textendash} ,
523     refpre   = {} ,
524     refpos   = {} ,
525     refpre-in = {} ,
526     refpos-in = {} ,
527 }

```

### Get translations

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

```

\__zrefclever_get_type_transl:nnnNF {<language>} {<type>} {<key>}
<tl variable> {<false code>}

528 \prg_new_protected_conditional:Npnn
529 \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
530 {
531     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
532     \l__zrefclever_dict_language_tl
533     {
534         \prop_get:cnNTF
535         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
536         { type- #2 - #3 } #4
537         { \prg_return_true: }
538         { \prg_return_false: }
539     }
540     { \prg_return_false: }
541 }
542 \prg_generate_conditional_variant:Nnn
543 \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }

```

(End definition for `\_zrefclever_get_type_transl:nnnNF`.)

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

```

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

```

```

544 \prg_new_protected_conditional:Npnn
545 \__zrefclever_get_default_transl:nnN #1#2#3 { F }
546 {
547   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
548   \l__zrefclever_dict_language_tl
549   {
550     \prop_get:cnNTF
551     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
552     { default- #2 } #3
553     { \prg_return_true: }
554     { \prg_return_false: }
555   }
556   { \prg_return_false: }
557 }
558 \prg_generate_conditional_variant:Nnn
559 \__zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for \\_\_zrefclever\_get\_default\_transl:nnNF.)

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

```

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

560 % {<key>}<tl var to set>
561 \prg_new_protected_conditional:Npnn
562 \__zrefclever_get_fallback_transl:nN #1#2 { F }
563 {
564   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
565   { #1 } #2
566   { \prg_return_true: }
567   { \prg_return_false: }
568 }

```

(End definition for \\_\_zrefclever\_get\_fallback\_transl:nNF.)

## 4.6 Options

### Auxiliary

\\_\_zrefclever\_prop\_put\_non\_empty:Nnn If  $\langle value \rangle$  is empty, remove  $\langle key \rangle$  from  $\langle property\ list \rangle$ . Otherwise, add  $\langle key \rangle = \langle value \rangle$  to  $\langle property\ list \rangle$ .

```

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

569 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
570 {
571   \tl_if_empty:nTF {#3}
572   { \prop_remove:Nn #1 {#2} }
573   { \prop_put:Nnn #1 {#2} {#3} }
574 }

```

(End definition for \\_\_zrefclever\_prop\_put\_non\_empty:Nnn.)

## ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these three (or four) alternatives – `default`, `zc@thecnt`, `page`, and `title` if `zref-titleref` is loaded –, but there might be a case for making this more flexible. The infrastructure can already handle receiving an arbitrary property, as long as one is satisfied with sorting and compressing from the current counter. If more flexibility is granted, one thing *must* be handled at this point: the existence of the property itself, as far as `zref` is concerned. This because typesetting relies on the check `\zref@ifrefcontainsprop`, which *presumes* the property is defined and silently expands the *true* branch if it is not (see <https://github.com/ho-tex/zref/issues/13>, thanks Ulrike Fischer). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door.

```

575 \tl_new:N \l__zrefclever_ref_property_tl
576 \keys_define:nn { zref-clever / reference }
577 {
578   ref .choice: ,
579   ref / default .code:n =
580     { \tl_set:Nn \l__zrefclever_ref_property_tl { default } } ,
581   ref / zc@thecnt .code:n =
582     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
583   ref / page .code:n =
584     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
585   ref / title .code:n =
586     {
587       \AddToHook { begindocument }
588       {
589         \@ifpackageloaded { zref-titleref }
590         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
591         {
592           \msg_warning:nn { zref-clever } { missing-zref-titleref }
593           \tl_set:Nn \l__zrefclever_ref_property_tl { default }
594         }
595       }
596     } ,
597   ref .initial:n = default ,
598   ref .default:n = default ,
599   page .meta:n = { ref = page } ,
600   page .value_forbidden:n = true ,
601 }
602 \AddToHook { begindocument }
603 {
604   \@ifpackageloaded { zref-titleref }
605   {
606     \keys_define:nn { zref-clever / reference }
607     {
608       ref / title .code:n =
609       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
610     }
611   }
612   {
613     \keys_define:nn { zref-clever / reference }
614     {

```

```

615         ref / title .code:n =
616         {
617             \msg_warning:nn { zref-clever } { missing-zref-titleref }
618             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
619         }
620     }
621 }
622 }

```

### typeset option

```

623 \bool_new:N \l__zrefclever_typeset_ref_bool
624 \bool_new:N \l__zrefclever_typeset_name_bool
625 \keys_define:nn { zref-clever / reference }
626 {
627     typeset .choice: ,
628     typeset / both .code:n =
629     {
630         \bool_set_true:N \l__zrefclever_typeset_ref_bool
631         \bool_set_true:N \l__zrefclever_typeset_name_bool
632     } ,
633     typeset / ref .code:n =
634     {
635         \bool_set_true:N \l__zrefclever_typeset_ref_bool
636         \bool_set_false:N \l__zrefclever_typeset_name_bool
637     } ,
638     typeset / name .code:n =
639     {
640         \bool_set_false:N \l__zrefclever_typeset_ref_bool
641         \bool_set_true:N \l__zrefclever_typeset_name_bool
642     } ,
643     typeset .initial:n = both ,
644     typeset .value_required:n = true ,
645
646     noname .meta:n = { typeset = ref },
647     noname .value_forbidden:n = true ,
648 }

```

### sort option

```

649 \bool_new:N \l__zrefclever_typeset_sort_bool
650 \keys_define:nn { zref-clever / reference }
651 {
652     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
653     sort .initial:n = true ,
654     sort .default:n = true ,
655     nosort .meta:n = { sort = false },
656     nosort .value_forbidden:n = true ,
657 }

```

### 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`.

```

658 \seq_new:N \l__zrefclever_typesort_seq
659 \keys_define:nn { zref-clever / reference }
660 {
661   typesort .code:n =
662   {
663     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
664     \seq_reverse:N \l__zrefclever_typesort_seq
665   } ,
666   typesort .initial:n =
667   { part , chapter , section , paragraph } ,
668   typesort .value_required:n = true ,
669   notypesort .code:n =
670   { \seq_clear:N \l__zrefclever_typesort_seq } ,
671   notypesort .value_forbidden:n = true ,
672 }

```

#### comp option

```

673 \bool_new:N \l__zrefclever_typeset_compress_bool
674 \keys_define:nn { zref-clever / reference }
675 {
676   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
677   comp .initial:n = true ,
678   comp .default:n = true ,
679   nocomp .meta:n = { comp = false } ,
680   nocomp .value_forbidden:n = true ,
681 }

```

#### range option

```

682 \bool_new:N \l__zrefclever_typeset_range_bool
683 \keys_define:nn { zref-clever / reference }
684 {
685   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
686   range .initial:n = false ,
687   range .default:n = true ,
688 }

```

#### cap and capfirst options

```

689 \bool_new:N \l__zrefclever_capitalize_bool
690 \bool_new:N \l__zrefclever_capitalize_first_bool
691 \keys_define:nn { zref-clever / reference }
692 {
693   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
694   cap .initial:n = false ,
695   cap .default:n = true ,
696   nocap .meta:n = { cap = false } ,
697   nocap .value_forbidden:n = true ,
698
699   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
700   capfirst .initial:n = false ,
701   capfirst .default:n = true ,

```

```
702 }
```

### abbrev and noabbrevfirst options

```
703 \bool_new:N \l__zrefclever_abbrev_bool
704 \bool_new:N \l__zrefclever_noabbrev_first_bool
705 \keys_define:nn { zref-clever / reference }
706 {
707     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
708     abbrev .initial:n = false ,
709     abbrev .default:n = true ,
710     noabbrev .meta:n = { abbrev = false },
711     noabbrev .value_forbidden:n = true ,
712
713     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
714     noabbrevfirst .initial:n = false ,
715     noabbrevfirst .default:n = true ,
716 }
```

### S option

```
717 \keys_define:nn { zref-clever / reference }
718 {
719     S .meta:n =
720     { capfirst = true , noabbrevfirst = true },
721     S .value_forbidden:n = true ,
722 }
```

### hyperref option

```
723 \bool_new:N \l__zrefclever_use_hyperref_bool
724 \bool_new:N \l__zrefclever_warn_hyperref_bool
725 \keys_define:nn { zref-clever / reference }
726 {
727     hyperref .choice: ,
728     hyperref / auto .code:n =
729     {
730         \bool_set_true:N \l__zrefclever_use_hyperref_bool
731         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
732     } ,
733     hyperref / true .code:n =
734     {
735         \bool_set_true:N \l__zrefclever_use_hyperref_bool
736         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
737     } ,
738     hyperref / false .code:n =
739     {
740         \bool_set_false:N \l__zrefclever_use_hyperref_bool
741         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
742     } ,
743     hyperref .initial:n = auto ,
744     hyperref .default:n = auto
745 }
746 \AddToHook { begindocument }
747 {
748     \@ifpackageloaded { hyperref }
```



```

749     {
750       \bool_if:NT \l__zrefclever_use_hyperref_bool
751       { \RequirePackage { zref-hyperref } }
752     }
753     {
754       \bool_if:NT \l__zrefclever_warn_hyperref_bool
755       { \msg_warning:nn { zref-clever } { missing-hyperref } }
756       \bool_set_false:N \l__zrefclever_use_hyperref_bool
757     }
758     \keys_define:nn { zref-clever / reference }
759     {
760       hyperref .code:n =
761       { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
762     }
763   }

```

### nameinlink option

```

764 \str_new:N \l__zrefclever_nameinlink_str
765 \keys_define:nn { zref-clever / reference }
766 {
767   nameinlink .choice: ,
768   nameinlink / true .code:n =
769   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
770   nameinlink / false .code:n =
771   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
772   nameinlink / single .code:n =
773   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
774   nameinlink / tsingle .code:n =
775   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
776   nameinlink .initial:n = tsingle ,
777   nameinlink .default:n = true ,
778 }

```

### 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`.

```

779 \tl_new:N \l__zrefclever_ref_language_tl
780 \tl_new:N \l__zrefclever_main_language_tl
781 \tl_new:N \l__zrefclever_current_language_tl
782 \AddToHook { begindocument }
783 {
784   \ifpackageloaded { babel }
785   {
786     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
787     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
788   }
789   {
790     \ifpackageloaded { polyglossia }
791     {
792       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
793       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
794     }
795     {
796       \tl_set:Nn \l__zrefclever_current_language_tl { english }
797       \tl_set:Nn \l__zrefclever_main_language_tl { english }
798     }
799   }

```

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.

```

800   \tl_set:Nn \l__zrefclever_ref_language_tl
801   { \l__zrefclever_main_language_tl }
802 }
803 \keys_define:nn { zref-clever / reference }
804 {
805   lang .code:n =
806   {
807     \AddToHook { begindocument }
808     {
809       \str_case:nnF {#1}
810       {
811         { main }
812         {
813           \tl_set:Nn \l__zrefclever_ref_language_tl
814           { \l__zrefclever_main_language_tl }
815           \__zrefclever_provide_dictionary_verbosely:x
816           { \l__zrefclever_ref_language_tl }
817         }
818       }

```

```

819         { current }
820     {
821         \tl_set:Nn \l__zrefclever_ref_language_tl
822         { \l__zrefclever_current_language_tl }
823         \__zrefclever_provide_dictionary_verbose:x
824         { \l__zrefclever_ref_language_tl }
825     }
826 }
827 {
828     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
829     {
830         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
831     }
832     {
833         \msg_warning:nnn { zref-clever }
834         { unknown-language-opt } {#1}
835         \tl_set:Nn \l__zrefclever_ref_language_tl
836         { \l__zrefclever_main_language_tl }
837     }
838     \__zrefclever_provide_dictionary_verbose:x
839     { \l__zrefclever_ref_language_tl }
840 }
841 }
842 } ,
843 lang .value_required:n = true ,
844 }
845 \AddToHook { begindocument / before }
846 {
847     \AddToHook { begindocument }
848     {

```

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.

```

849     \__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.

```

850     \keys_define:nn { zref-clever / reference }
851     {
852         lang .code:n =
853         {
854             \str_case:nnF {#1}
855             {
856                 { main }
857                 {
858                     \tl_set:Nn \l__zrefclever_ref_language_tl
859                     { \l__zrefclever_main_language_tl }
860                     \__zrefclever_provide_dictionary:x
861                     { \l__zrefclever_ref_language_tl }
862                 }
863             }
864             { current }

```

```

865         {
866             \tl_set:Nn \l__zrefclever_ref_language_tl
867             { \l__zrefclever_current_language_tl }
868             \__zrefclever_provide_dictionary:x
869             { \l__zrefclever_ref_language_tl }
870         }
871     }
872     {
873         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
874         {
875             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
876         }
877         {
878             \msg_warning:nnn { zref-clever }
879             { unknown-language-opt } {#1}
880             \tl_set:Nn \l__zrefclever_ref_language_tl
881             { \l__zrefclever_main_language_tl }
882         }
883         \__zrefclever_provide_dictionary:x
884         { \l__zrefclever_ref_language_tl }
885     }
886     } ,
887     lang .value_required:n = true ,
888 }
889 }
890 }

```

#### d option

```

891 \tl_new:N \l__zrefclever_ref_decl_case_tl
892 \keys_define:nn { zref-clever / reference }
893 {
894     d .code:n =
895     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
896 }
897 \AddToHook { begindocument }
898 {
899     \keys_define:nn { zref-clever / reference }
900     {

```

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

```

901         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
902         d .value_required:n = true ,
903     }
904 }

```

`\__zrefclever_validate_decl_option:` Auxiliary function for `\__zrefclever_zcref:nnn`, responsible for validating the declension case (d) option for the reference language. It is expected to be called right (or soon) after `\keys_set:nn` in `\__zrefclever_zcref:nnn`, where current values for `\l__zrefclever_ref_language_tl` and `\l__zrefclever_ref_decl_case_tl`. 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.

```

905 \cs_new_protected:Npn \__zrefclever_validate_decl_option:
906 {
907   \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
908   { \l__zrefclever_ref_language_tl }
909   \l__zrefclever_dict_language_tl
910   {
911     \prop_if_exist:cTF
912     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
913     {
914       \exp_args:NNx \seq_set_from_clist:Nn
915       \l__zrefclever_dict_declension_seq
916       {
917         \prop_item:cn
918         {
919           g__zrefclever_dict_
920           \l__zrefclever_dict_language_tl _prop
921         }
922         { declension }
923       }
924       \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
925       {
926         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
927         {
928           \msg_warning:nxxx { zref-clever }
929           { language-no-decl-ref }
930           { \l__zrefclever_ref_language_tl }
931           { \l__zrefclever_ref_decl_case_tl }
932           \tl_clear:N \l__zrefclever_ref_decl_case_tl
933         }
934       }
935       {
936         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
937         {
938           \seq_get_left:NN \l__zrefclever_dict_declension_seq
939           \l__zrefclever_ref_decl_case_tl
940         }
941         {
942           \seq_if_in:NVF \l__zrefclever_dict_declension_seq
943           \l__zrefclever_ref_decl_case_tl
944           {
945             \msg_warning:nxxx { zref-clever }
946             { unknown-decl-case }
947             { \l__zrefclever_ref_decl_case_tl }
948             { \l__zrefclever_ref_language_tl }
949             \seq_get_left:NN \l__zrefclever_dict_declension_seq
950             \l__zrefclever_ref_decl_case_tl
951           }
952         }
953       }
954     }
955     {
956       \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl

```

```

957         {
958             \msg_warning:nxxx { zref-clever } { language-no-decl-ref }
959             { \l__zrefclever_ref_language_tl }
960             { \l__zrefclever_ref_decl_case_tl }
961             \tl_clear:N \l__zrefclever_ref_decl_case_tl
962         }
963     }
964 }
965 {
966     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
967     {
968         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
969         { \l__zrefclever_ref_decl_case_tl }
970         { \l__zrefclever_ref_language_tl }
971         \tl_clear:N \l__zrefclever_ref_decl_case_tl
972     }
973 }
974 }

```

(End definition for `\__zrefclever_validate_decl_option:.`)

### nudge, nudgeif, and sg options

```

975 \bool_new:N \l__zrefclever_nudge_enabled_bool
976 \bool_new:N \l__zrefclever_nudge_multitype_bool
977 \bool_new:N \l__zrefclever_nudge_comptosing_bool
978 \bool_new:N \l__zrefclever_nudge_singular_bool
979 \keys_define:nn { zref-clever / reference }
980 {
981     nudge .choice: ,
982     nudge / true .code:n =
983     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
984     nudge / false .code:n =
985     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
986     nudge / obeydraft .code:n =
987     {
988         \ifdraft
989         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
990         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
991     } ,
992     nudge / obeyfinal .code:n =
993     {
994         \ifoptionfinal
995         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
996         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
997     } ,
998     nudge .initial:n = false ,
999     nudge .default:n = true ,
1000     nonnudge .meta:n = { nudge = false } ,
1001     nonnudge .value_forbidden:n = true ,
1002     nudgeif .code:n =
1003     {
1004         \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1005         \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1006         \clist_map_inline:nn {#1}

```

```

1007     {
1008         \str_case:nnF {##1}
1009         {
1010             { multitype }
1011             { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1012             { comptosing }
1013             { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1014             { all }
1015             {
1016                 \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1017                 \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1018             }
1019         }
1020         {
1021             \msg_warning:nnn { zref-clever }
1022             { nudgeif-unknown-value } {##1}
1023         }
1024     }
1025 },
1026 nudgeif .value_required:n = true ,
1027 nudgeif .initial:n = all ,
1028 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1029 sg .initial:n = false ,
1030 sg .default:n = true ,
1031 }

```

### font option

`font` *can't be used as a package option*, since the options get expanded by L<sup>A</sup>T<sub>E</sub>X before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can't be set in `\zcref` and, for global settings, with `\zcsetup`.

```

1032 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1033 \keys_define:nn { zref-clever / reference }
1034 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

### titleref option

```

1035 \keys_define:nn { zref-clever / reference }
1036 {
1037     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1038     titleref .value_forbidden:n = true ,
1039 }
1040 \AddToHook { begindocument }
1041 {
1042     \keys_define:nn { zref-clever / reference }
1043     {
1044         titleref .code:n =
1045         { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1046     }
1047 }

```

### note option

```

1048 \tl_new:N \l__zrefclever_zcref_note_tl
1049 \keys_define:nn { zref-clever / reference }

```

```

1050 {
1051   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
1052   note .value_required:n = true ,
1053 }

```

### check option

Integration with zref-check.

```

1054 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1055 \bool_new:N \l__zrefclever_zceref_with_check_bool
1056 \keys_define:nn { zref-clever / reference }
1057 {
1058   check .code:n = { \RequirePackage { zref-check } } ,
1059   check .value_forbidden:n = true ,
1060 }
1061 \AddToHook { begindocument }
1062 {
1063   \@ifpackageloaded { zref-check }
1064   {
1065     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1066     \keys_define:nn { zref-clever / reference }
1067     {
1068       check .code:n =
1069       {
1070         \bool_set_true:N \l__zrefclever_zceref_with_check_bool
1071         \keys_set:nn { zref-check / zcheck } {#1}
1072       } ,
1073       check .value_required:n = true ,
1074     }
1075   }
1076   {
1077     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1078     \keys_define:nn { zref-clever / reference }
1079     {
1080       check .value_forbidden:n = false ,
1081       check .code:n =
1082       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1083     }
1084   }
1085 }

```

### 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`.

```

1086 \prop_new:N \l__zrefclever_counter_type_prop
1087 \keys_define:nn { zref-clever / label }
1088 {
1089   countertype .code:n =
1090   {
1091     \keyval_parse:nnn
1092     {

```



```

1093         \msg_warning:nnnn { zref-clever }
1094         { key-requires-value } { countertype }
1095     }
1096     {
1097         \__zrefclever_prop_put_non_empty:Nnn
1098         \l__zrefclever_counter_type_prop
1099     }
1100     {#1}
1101 } ,
1102 countertype .value_required:n = true ,
1103 countertype .initial:n =
1104 {
1105     subsection    = section ,
1106     subsubsection = section ,
1107     subparagraph  = paragraph ,
1108     enumi         = item ,
1109     enumii        = item ,
1110     enumiii       = item ,
1111     enumiv        = item ,
1112     mpfootnote    = footnote ,
1113 } ,
1114 }

```

### 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.

```

1115 \seq_new:N \l__zrefclever_counter_resetters_seq
1116 \keys_define:nn { zref-clever / label }
1117 {
1118     counterresetters .code:n =
1119     {
1120         \clist_map_inline:nn {#1}
1121         {
1122             \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1123             {
1124                 \seq_put_right:Nn
1125                 \l__zrefclever_counter_resetters_seq {##1}
1126             }
1127         }
1128     } ,
1129     counterresetters .initial:n =
1130     {
1131         part ,
1132         chapter ,
1133         section ,
1134         subsection ,
1135         subsubsection ,

```

```

1136     paragraph ,
1137     subparagraph ,
1138   },
1139   counterresetters .value_required:n = true ,
1140 }

```

### 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`.

```

1141 \prop_new:N \l__zrefclever_counter_resetby_prop
1142 \keys_define:nn { zref-clever / label }
1143 {
1144   counterresetby .code:n =
1145   {
1146     \keyval_parse:nnn
1147     {
1148       \msg_warning:nnn { zref-clever }
1149       { key-requires-value } { counterresetby }
1150     }
1151     {
1152       \__zrefclever_prop_put_non_empty:Nnn
1153       \l__zrefclever_counter_resetby_prop
1154     }
1155     {#1}
1156   } ,
1157   counterresetby .value_required:n = true ,
1158   counterresetby .initial:n =
1159   {

```

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.

```

1160     enumii = enumi ,
1161     enumiii = enumii ,
1162     enumiv = enumiii ,
1163   } ,
1164 }

```

### 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.

```

1165 \tl_new:N \l__zrefclever_current_counter_tl
1166 \keys_define:nn { zref-clever / label }
1167 {
1168   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1169   currentcounter .value_required:n = true ,

```

```

1170     currentcounter .initial:n = \@currentcounter ,
1171 }

```

## 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.

```

1172 \prop_new:N \l__zrefclever_ref_options_prop
1173 \seq_map_inline:Nn
1174   \c__zrefclever_ref_options_reference_seq
1175   {
1176     \keys_define:nn { zref-clever / reference }
1177     {
1178       #1 .default:V = \c_novalue_tl ,
1179       #1 .code:n =
1180       {
1181         \tl_if_novalue:nTF {##1}
1182         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1183         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1184       } ,
1185     }
1186   }

```

## 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.

```

1187 \keys_define:nn { }
1188 {
1189   zref-clever / zcsetup .inherit:n =
1190   {
1191     zref-clever / label ,
1192     zref-clever / reference ,
1193   }
1194 }

```

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

```

1195 \ProcessKeysOptions { zref-clever / zcsetup }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

1196 \NewDocumentCommand \zcsetup { m }
1197   { \_zrefclever_zcsetup:n {#1} }

(End definition for \zcsetup.)

```

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

```

\_zrefclever_zcsetup:n{<options>}

1198 \cs_new_protected:Npn \_zrefclever_zcsetup:n #1
1199   { \keys_set:nn { zref-clever / zcsetup } {#1} }
1200 \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>}

1201 \NewDocumentCommand \zcRefTypeSetup { m m }
1202   {
1203     \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1204     { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1205     \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1206     \keys_set:nn { zref-clever / typesetup } {#2}
1207   }

(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>.

```

1208 \seq_map_inline:Nn
1209   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1210   {
1211     \keys_define:nn { zref-clever / typesetup }
1212     {
1213       #1 .code:n =
1214       {
1215         \msg_warning:nnn { zref-clever }
1216         { option-not-type-specific } {#1}
1217       } ,
1218     }
1219   }
1220 \seq_map_inline:Nn
1221   \c__zrefclever_ref_options_typesetup_seq
1222   {
1223     \keys_define:nn { zref-clever / typesetup }
1224     {
1225       #1 .default:V = \c_novalue_tl ,
1226       #1 .code:n =
1227       {
1228         \tl_if_novalue:nTF {##1}
1229         {
1230           \prop_remove:cn
1231           {
1232             l__zrefclever_type_
1233             \l__zrefclever_setup_type_tl _options_prop
1234           }
1235           {#1}
1236         }
1237         {
1238           \prop_put:cnn
1239           {
1240             l__zrefclever_type_
1241             \l__zrefclever_setup_type_tl _options_prop
1242           }
1243           {#1} {##1}
1244         }
1245       } ,
1246     }
1247   }

```

### 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>}}
1248 \NewDocumentCommand \zcLanguageSetup { m m }
1249 {
1250   \group_begin:
1251   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1252   \l__zrefclever_dict_language_tl
1253   {
1254     \tl_clear:N \l__zrefclever_setup_type_tl
1255     \prop_if_exist:cF
1256     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
1257     {
1258       \prop_new:c
1259       { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
1260     }
1261     \exp_args:NNx \seq_set_from_clist:Nn
1262     \l__zrefclever_dict_declension_seq
1263     {
1264       \prop_item:cn
1265       {
1266         g__zrefclever_dict_
1267         \l__zrefclever_dict_language_tl _prop
1268       }
1269       { declension }
1270     }
1271     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1272     { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
1273     {
1274       \seq_get_left:NN \l__zrefclever_dict_declension_seq
1275       \l__zrefclever_dict_decl_case_tl
1276     }
1277     \keys_set:nn { zref-clever / langsetup } {#2}
1278   }
1279   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1280   \group_end:
1281 }
1282 \@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>}}

1283 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1284 {
1285   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1286   { type- #2 - #3 } {#4}
1287 }
1288 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn , VVxn }
1289 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3

```

```

1290 {
1291   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1292   { default- #2 } {#3}
1293 }
1294 \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.

```

1295 \keys_define:nn { zref-clever / langsetup }
1296 {
1297   type .code:n =
1298   {
1299     \tl_if_empty:NTF {#1}
1300     { \tl_clear:N \l__zrefclever_setup_type_tl }
1301     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1302   } ,
1303   case .code:n =
1304   {
1305     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1306     {
1307       \msg_warning:nnxx { zref-clever } { language-no-decl-setup }
1308       { \l__zrefclever_dict_language_tl } {#1}
1309     }
1310     {
1311       \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
1312       { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
1313       {
1314         \msg_warning:nnxx { zref-clever } { unknown-decl-case }
1315         {#1} { \l__zrefclever_dict_language_tl }
1316         \seq_get_left:NN \l__zrefclever_dict_declension_seq
1317         \l__zrefclever_dict_decl_case_tl
1318       }
1319     }
1320   } ,
1321   case .value_required:n = true ,
1322 }
1323 \seq_map_inline:Nn
1324 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1325 {
1326   \keys_define:nn { zref-clever / langsetup }
1327   {
1328     #1 .value_required:n = true ,
1329     #1 .code:n =
1330     {
1331       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1332       {
1333         \__zrefclever_declare_default_transl:Vnn
1334         \l__zrefclever_dict_language_tl
1335         {#1} {##1}
1336       }
1337       {
1338         \msg_warning:nnn { zref-clever }

```

```

1339         { option-not-type-specific } {#1}
1340     }
1341 } ,
1342 }
1343 }
1344 \seq_map_inline:Nn
1345   \c__zrefclever_ref_options_possibly_type_specific_seq
1346   {
1347     \keys_define:nn { zref-clever / langsetup }
1348     {
1349       #1 .value_required:n = true ,
1350       #1 .code:n =
1351       {
1352         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1353         {
1354           \__zrefclever_declare_default_transl:Vnn
1355             \l__zrefclever_dict_language_tl
1356             {#1} {##1}
1357         }
1358         {
1359           \__zrefclever_declare_type_transl:VVnn
1360             \l__zrefclever_dict_language_tl
1361             \l__zrefclever_setup_type_tl
1362             {#1} {##1}
1363         }
1364       } ,
1365     }
1366   }
1367 \seq_map_inline:Nn
1368   \c__zrefclever_ref_options_type_names_seq
1369   {
1370     \keys_define:nn { zref-clever / langsetup }
1371     {
1372       #1 .value_required:n = true ,
1373       #1 .code:n =
1374       {
1375         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1376         {
1377           \msg_warning:nnn { zref-clever }
1378             { option-only-type-specific } {#1}
1379         }
1380         {
1381           \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
1382           {
1383             \__zrefclever_declare_type_transl:VVnn
1384               \l__zrefclever_dict_language_tl
1385               \l__zrefclever_setup_type_tl
1386               {#1} {##1}
1387           }
1388           {
1389             \__zrefclever_declare_type_transl:VVxn
1390               \l__zrefclever_dict_language_tl
1391               \l__zrefclever_setup_type_tl
1392               { \l__zrefclever_dict_decl_case_tl - #1 } {##1}

```



```

1393         }
1394     }
1395 } ,
1396 }
1397 }

```

## 6 User interface

### 6.1 `\zcref`

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

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

```

1398 \NewDocumentCommand \zcref { s O { } m }
1399 { \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>}
```

```

1400 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1401 {
1402     \group_begin:

```

Set options.

```
1403     \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```

1404     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1405     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

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

```
1406     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Validate type name declension option.

```
1407     \__zrefclever_validate_decl_option:
```

Integration with `zref-check`.

```

1408     \bool_lazy_and:nnT
1409     { \l__zrefclever_zrefcheck_available_bool }
1410     { \l__zrefclever_zcref_with_check_bool }
1411     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1412     \bool_lazy_or:nnT
1413     { \l__zrefclever_typeset_sort_bool }
1414     { \l__zrefclever_typeset_range_bool }
1415     { \__zrefclever_sort_labels: }

```

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

```

1416     \group_begin:
1417     \l__zrefclever_ref_typeset_font_tl
1418     \__zrefclever_typeset_refs:
1419     \group_end:

```

Typeset note.

```

1420     \tl_if_empty:NF \l__zrefclever_zceref_note_tl
1421     {
1422         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1423         \l_tmpa_tl
1424         \l__zrefclever_zceref_note_tl
1425     }

```

Integration with zref-check.

```

1426     \bool_lazy_and:nnT
1427     { \l__zrefclever_zrefcheck_available_bool }
1428     { \l__zrefclever_zceref_with_check_bool }
1429     {
1430         \zrefcheck_zceref_end_label_maybe:
1431         \zrefcheck_zceref_run_checks_on_labels:n
1432         { \l__zrefclever_zceref_labels_seq }
1433     }

```

Integration with mathtools.

```

1434     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1435     {
1436         \__zrefclever_mathtools_showonlyrefs:n
1437         { \l__zrefclever_zceref_labels_seq }
1438     }
1439     \group_end:
1440 }

```

(End definition for \\_\_zrefclever\_zceref:nnnn.)

```

\l__zrefclever_zceref_labels_seq
\l__zrefclever_link_star_bool

```

```

1441 \seq_new:N \l__zrefclever_zceref_labels_seq
1442 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for \l\_\_zrefclever\_zceref\_labels\_seq and \l\_\_zrefclever\_link\_star\_bool.)

## 6.2 \zcpageref

\zcpageref A \pageref equivalent of \zceref.

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

```

1443 \NewDocumentCommand \zcpageref { s O { } m }
1444 {
1445     \IfBooleanTF {#1}
1446     { \zceref*{#2, ref = page} {#3} }
1447     { \zceref [#2, ref = page] {#3} }
1448 }

```

(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 `\zceref`. 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.

`\l_zrefclever_label_type_a_tl` 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_b_tl`

`\l_zrefclever_label_enclval_a_tl` 1449 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 1450 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 1451 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 1452 `\tl_new:N \l__zrefclever_label_enclval_b_tl`

1453 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`

1454 `\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.

1455 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End definition for `\l__zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

1456 `\int_new:N \l__zrefclever_sort_prior_a_int`

1457 `\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 `\zceref`. 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`.

1458 `\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_zceref:nnnn` where a number of environment variables are to be set appropriately. In particular, `\l__zrefclever_zceref_labels_seq` should contain the labels received as argument to `\zceref`, and the function performs its task by sorting this variable.

1459 `\cs_new_protected:Npn \__zrefclever_sort_labels:`  
1460 `{`

Store label types sequence.

```

1461 \seq_clear:N \l__zrefclever_label_types_seq
1462 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1463 {
1464   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1465   \__zrefclever_label_type_put_new_right:n
1466 }

```

Sort.

```

1467 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1468 {
1469   \zref@ifrefundefined {##1}
1470   {
1471     \zref@ifrefundefined {##2}
1472     {
1473       % Neither label is defined.
1474       \sort_return_same:
1475     }
1476     {
1477       % The second label is defined, but the first isn't, leave the
1478       % undefined first (to be more visible).
1479       \sort_return_same:
1480     }
1481   }
1482   {
1483     \zref@ifrefundefined {##2}
1484     {
1485       % The first label is defined, but the second isn't, bring the
1486       % second forward.
1487       \sort_return_swapped:
1488     }
1489     {
1490       % The interesting case: both labels are defined. References
1491       % to the "default" property or to the "page" are quite
1492       % different with regard to sorting, so we branch them here to
1493       % specialized functions.
1494       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1495       { \__zrefclever_sort_page:nn {##1} {##2} }
1496       { \__zrefclever_sort_default:nn {##1} {##2} }
1497     }
1498   }
1499 }
1500 }

```

(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}}
1501 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1502 {
1503   \_zrefclever_def_extract:Nnnn
1504   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1505   \seq_if_in:NVF \l__zrefclever_label_types_seq
1506   \l__zrefclever_label_type_a_tl
1507   {
1508     \seq_put_right:NV \l__zrefclever_label_types_seq
1509     \l__zrefclever_label_type_a_tl
1510   }
1511 }

```

(End definition for \\_zrefclever\_label\_type\_put\_new\_right:n.)

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

```

    \_zrefclever_sort_default:nn {\label a} {\label b}
1512 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
1513 {
1514   \_zrefclever_def_extract:Nnnn
1515   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1516   \_zrefclever_def_extract:Nnnn
1517   \l__zrefclever_label_type_b_tl {#2} {zc@type} {\c_empty_tl}
1518
1519   \bool_if:nTF
1520   {
1521     % The second label has a type, but the first doesn't, leave the
1522     % undefined first (to be more visible).
1523     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1524     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1525   }
1526   { \sort_return_same: }
1527   {
1528     \bool_if:nTF
1529     {
1530       % The first label has a type, but the second doesn't, bring the
1531       % second forward.
1532       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1533       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1534     }
1535     { \sort_return_swapped: }
1536     {
1537       \bool_if:nTF
1538       {
1539         % The interesting case: both labels have a type...
1540         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1541         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1542       }

```

```

1543 {
1544   \tl_if_eq:NNTF
1545     \l__zrefclever_label_type_a_tl
1546     \l__zrefclever_label_type_b_tl
1547     % ...and it's the same type.
1548     { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1549     % ...and they are different types.
1550     { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1551   }
1552 {
1553   % Neither label has a type. We can't do much of meaningful
1554   % here, but if it's the same counter, compare it.
1555   \exp_args:Nxx \tl_if_eq:nnTF
1556     { \__zrefclever_extract_unexp:nnn {#1} {zc@counter} { } }
1557     { \__zrefclever_extract_unexp:nnn {#2} {zc@counter} { } }
1558     {
1559       \int_compare:nNnTF
1560         { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
1561         >
1562         { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
1563         { \sort_return_swapped: }
1564         { \sort_return_same: }
1565       }
1566     { \sort_return_same: }
1567   }
1568 }
1569 }
1570 }

```

(End definition for \\_\_zrefclever\_sort\_default:nn.)

```

\__zrefclever_sort_default_same_type:nn      \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
1571 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1572 {
1573   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_a_tl
1574     {#1} {zc@enclval} { \c_empty_tl }
1575   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1576   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_b_tl
1577     {#2} {zc@enclval} { \c_empty_tl }
1578   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1579   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
1580     {#1} {externaldocument} { \c_empty_tl }
1581   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
1582     {#2} {externaldocument} { \c_empty_tl }
1583
1584   \bool_set_false:N \l__zrefclever_sort_decided_bool
1585
1586   % First we check if there's any "external document" difference (coming
1587   % from 'zref-xr') and, if so, sort based on that.
1588   \tl_if_eq:NNTF
1589     \l__zrefclever_label_extdoc_a_tl
1590     \l__zrefclever_label_extdoc_b_tl
1591     {
1592       \bool_if:nTF

```

```

1593 {
1594   \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1595   ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1596 }
1597 {
1598   \bool_set_true:N \l__zrefclever_sort_decided_bool
1599   \sort_return_same:
1600 }
1601 {
1602   \bool_if:nTF
1603   {
1604     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1605     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1606   }
1607   {
1608     \bool_set_true:N \l__zrefclever_sort_decided_bool
1609     \sort_return_swapped:
1610   }
1611   {
1612     \bool_set_true:N \l__zrefclever_sort_decided_bool
1613     % Two different "external documents": last resort, sort by the
1614     % document name itself.
1615     \str_compare:eNeTF
1616     { \l__zrefclever_label_extdoc_b_tl } <
1617     { \l__zrefclever_label_extdoc_a_tl }
1618     { \sort_return_swapped: }
1619     { \sort_return_same: }
1620   }
1621 }
1622 }
1623
1624 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1625 {
1626   \bool_if:nTF
1627   {
1628     % Both are empty: neither label has any (further) "enclosing
1629     % counters" (left).
1630     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1631     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1632   }
1633   {
1634     \bool_set_true:N \l__zrefclever_sort_decided_bool
1635     \int_compare:nNnTF
1636     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1637     >
1638     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1639     { \sort_return_swapped: }
1640     { \sort_return_same: }
1641   }
1642   {
1643     \bool_if:nTF
1644     {
1645       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1646       \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl

```

```

1647     }
1648     {
1649         \bool_set_true:N \l__zrefclever_sort_decided_bool
1650         \int_compare:nNnTF
1651             { \__zrefclever_extract:nnn {#1} {zc@cntval} { } }
1652             >
1653             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1654             { \sort_return_swapped: }
1655             { \sort_return_same: }
1656     }
1657     {
1658         \bool_if:nTF
1659         {
1660             % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1661             \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1662         }
1663         {
1664             \bool_set_true:N \l__zrefclever_sort_decided_bool
1665             \int_compare:nNnTF
1666                 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1667                 <
1668                 { \__zrefclever_extract:nnn {#2} {zc@cntval} { } }
1669                 { \sort_return_same: }
1670                 { \sort_return_swapped: }
1671         }
1672         {
1673             % Neither is empty: we can compare the values of the
1674             % current enclosing counter in the loop, if they are
1675             % equal, we are still in the loop, if they are not, a
1676             % sorting decision can be made directly.
1677             \int_compare:nNnTF
1678                 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1679                 =
1680                 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1681                 {
1682                     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1683                     { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1684                     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1685                     { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1686                 }
1687                 {
1688                     \bool_set_true:N \l__zrefclever_sort_decided_bool
1689                     \int_compare:nNnTF
1690                         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1691                         >
1692                         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1693                         { \sort_return_swapped: }
1694                         { \sort_return_same: }
1695                 }
1696         }
1697     }
1698 }
1699 }
1700 }

```



(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}
```

```
1701 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1702 {
```

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”.

```
1703   \int_zero:N \l__zrefclever_sort_prior_a_int
1704   \int_zero:N \l__zrefclever_sort_prior_b_int
1705   \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1706   {
1707     \tl_if_eq:nnTF {##2} {{othertypes}}
1708     {
1709       \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1710       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1711       \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1712       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1713     }
1714     {
1715       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1716       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1717       {
1718         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1719         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1720       }
1721     }
1722   }
```

Then do the actual sorting.

```
1723   \bool_if:nTF
1724   {
1725     \int_compare_p:nNn
1726     { \l__zrefclever_sort_prior_a_int } <
1727     { \l__zrefclever_sort_prior_b_int }
1728   }
1729   { \sort_return_same: }
1730   {
1731     \bool_if:nTF
1732     {
1733       \int_compare_p:nNn
1734       { \l__zrefclever_sort_prior_a_int } >
1735       { \l__zrefclever_sort_prior_b_int }
1736     }
1737     { \sort_return_swapped: }
1738     {
1739       % Sort priorities are equal: the type that occurs first in
1740       % ‘labels’, as given by the user, is kept (or brought) forward.
1741       \seq_map_inline:Nn \l__zrefclever_label_types_seq
1742       {
1743         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1744         { \seq_map_break:n { \sort_return_same: } }
1745         {
```

```

1746             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1747             { \seq_map_break:n { \sort_return_swapped: } }
1748         }
1749     }
1750 }
1751 }
1752 }

```

(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}

1753 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1754 {
1755     \int_compare:nNnTF
1756     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1757     >
1758     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1759     { \sort_return_swapped: }
1760     { \sort_return_same: }
1761 }

```

(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 a 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

`\l_zrefclever_typeset_labels_seq`  
`\l_zrefclever_typeset_last_bool`  
`\l_zrefclever_last_of_type_bool`

Auxiliary variables for `\__zrefclever_typeset_refs`: main stack control.

1762 `\seq_new:N \l__zrefclever_typeset_labels_seq`  
1763 `\bool_new:N \l__zrefclever_typeset_last_bool`  
1764 `\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`.)

`\l_zrefclever_type_count_int`  
`\l_zrefclever_label_count_int`

Auxiliary variables for `\__zrefclever_typeset_refs`: main counters.

1765 `\int_new:N \l__zrefclever_type_count_int`  
1766 `\int_new:N \l__zrefclever_label_count_int`

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

`\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`

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

1767 `\tl_new:N \l__zrefclever_label_a_tl`  
1768 `\tl_new:N \l__zrefclever_label_b_tl`  
1769 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`  
1770 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`  
1771 `\tl_new:N \l__zrefclever_type_first_label_tl`  
1772 `\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`  
`\l_zrefclever_name_in_link_bool`  
`\l_zrefclever_name_format_tl`  
`\l_zrefclever_name_format_fallback_tl`

Auxiliary variables for `\__zrefclever_typeset_refs`: type name handling.

1773 `\tl_new:N \l__zrefclever_type_name_tl`  
1774 `\bool_new:N \l__zrefclever_name_in_link_bool`  
1775 `\tl_new:N \l__zrefclever_name_format_tl`  
1776 `\tl_new:N \l__zrefclever_name_format_fallback_tl`

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

`\l_zrefclever_range_count_int`  
`\l_zrefclever_range_same_count_int`  
`\l_zrefclever_range_beg_label_tl`  
`\l_zrefclever_next_maybe_range_bool`  
`\l_zrefclever_next_is_same_bool`

Auxiliary variables for `\__zrefclever_typeset_refs`: range handling.

1777 `\int_new:N \l__zrefclever_range_count_int`  
1778 `\int_new:N \l__zrefclever_range_same_count_int`  
1779 `\tl_new:N \l__zrefclever_range_beg_label_tl`  
1780 `\bool_new:N \l__zrefclever_next_maybe_range_bool`  
1781 `\bool_new:N \l__zrefclever_next_is_same_bool`

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

`\l__zrefclever_tpairsep_tl`  
`\l__zrefclever_tlistsep_tl`  
`\l__zrefclever_tlastsep_tl`  
`\l__zrefclever_namesep_tl`  
`\l__zrefclever_pairsep_tl`  
`\l__zrefclever_listsep_tl`  
`\l__zrefclever_lastsep_tl`  
`\l__zrefclever_rangesep_tl`  
`\l__zrefclever_refpre_out_tl`  
`\l__zrefclever_refpos_out_tl`  
`\l__zrefclever_refpre_in_tl`  
`\l__zrefclever_refpos_in_tl`  
`\l__zrefclever_namefont_tl`  
`\l_zrefclever_reffont_out_tl`  
`\l__zrefclever_reffont_in_tl`

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

1782 `\tl_new:N \l__zrefclever_tpairsep_tl`  
1783 `\tl_new:N \l__zrefclever_tlistsep_tl`  
1784 `\tl_new:N \l__zrefclever_tlastsep_tl`  
1785 `\tl_new:N \l__zrefclever_namesep_tl`  
1786 `\tl_new:N \l__zrefclever_pairsep_tl`  
1787 `\tl_new:N \l__zrefclever_listsep_tl`  
1788 `\tl_new:N \l__zrefclever_lastsep_tl`  
1789 `\tl_new:N \l__zrefclever_rangesep_tl`  
1790 `\tl_new:N \l__zrefclever_refpre_out_tl`

```

1791 \tl_new:N \l__zrefclever_refpos_out_tl
1792 \tl_new:N \l__zrefclever_refpre_in_tl
1793 \tl_new:N \l__zrefclever_refpos_in_tl
1794 \tl_new:N \l__zrefclever_namefont_tl
1795 \tl_new:N \l__zrefclever_reffont_out_tl
1796 \tl_new:N \l__zrefclever_reffont_in_tl

```

(End definition for \l\_\_zrefclever\_tpairsep\_tl and others.)

## Main functions

\\_\_zrefclever\_typeset\_refs: Main typesetting function for \zcref.

```

1797 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1798 {
1799   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1800   \l__zrefclever_zcref_labels_seq
1801   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1802   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1803   \tl_clear:N \l__zrefclever_type_first_label_tl
1804   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1805   \tl_clear:N \l__zrefclever_range_beg_label_tl
1806   \int_zero:N \l__zrefclever_label_count_int
1807   \int_zero:N \l__zrefclever_type_count_int
1808   \int_zero:N \l__zrefclever_range_count_int
1809   \int_zero:N \l__zrefclever_range_same_count_int
1810
1811   % Get type block options (not type-specific).
1812   \__zrefclever_get_ref_string:nN { tpairsep }
1813   \l__zrefclever_tpairsep_tl
1814   \__zrefclever_get_ref_string:nN { tlistsep }
1815   \l__zrefclever_tlistsep_tl
1816   \__zrefclever_get_ref_string:nN { tlastsep }
1817   \l__zrefclever_tlastsep_tl
1818
1819   % Process label stack.
1820   \bool_set_false:N \l__zrefclever_typeset_last_bool
1821   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1822   {
1823     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1824     \l__zrefclever_label_a_tl
1825     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1826     {
1827       \tl_clear:N \l__zrefclever_label_b_tl
1828       \bool_set_true:N \l__zrefclever_typeset_last_bool
1829     }
1830     {
1831       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1832       \l__zrefclever_label_b_tl
1833     }
1834
1835     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1836     {
1837       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1838       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }

```

```

1839     }
1840     {
1841         \__zrefclever_def_extract:N\mn \l__zrefclever_label_type_a_tl
1842         \l__zrefclever_label_a_tl { zc@type } { \c_empty_tl }
1843         \__zrefclever_def_extract:N\mn \l__zrefclever_label_type_b_tl
1844         \l__zrefclever_label_b_tl { zc@type } { \c_empty_tl }
1845     }
1846
1847     % First, we establish whether the "current label" (i.e. 'a') is the
1848     % last one of its type. This can happen because the "next label"
1849     % (i.e. 'b') is of a different type (or different definition status),
1850     % or because we are at the end of the list.
1851     \bool_if:NTF \l__zrefclever_typeset_last_bool
1852     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1853     {
1854         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1855         {
1856             \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1857             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1858             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1859         }
1860         {
1861             \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1862             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1863             {
1864                 % Neither is undefined, we must check the types.
1865                 \bool_if:nTF
1866                 {
1867                     % Both empty: same "type".
1868                     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1869                     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1870                 }
1871                 { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1872                 {
1873                     \bool_if:nTF
1874                     {
1875                         % Neither empty: compare types.
1876                         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1877                         &&
1878                         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1879                     }
1880                     {
1881                         \tl_if_eq:NNTF
1882                         \l__zrefclever_label_type_a_tl
1883                         \l__zrefclever_label_type_b_tl
1884                         {
1885                             \bool_set_false:N
1886                             \l__zrefclever_last_of_type_bool
1887                         }
1888                         {
1889                             \bool_set_true:N
1890                             \l__zrefclever_last_of_type_bool
1891                         }
1892                     }
1893                 }
1894             }
1895         }
1896     }

```

```

1893                                     % One empty, the other not: different "types".
1894                                     {
1895                                         \bool_set_true:N
1896                                         \l__zrefclever_last_of_type_bool
1897                                     }
1898                                 }
1899                            }
1900                    }
1901            }
1902
1903    % Handle warnings in case of reference or type undefined.
1904    \zref@refused { \l__zrefclever_label_a_tl }
1905    \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1906        {}
1907        {
1908            \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1909                {
1910                    \msg_warning:nxx { zref-clever } { missing-type }
1911                    { \l__zrefclever_label_a_tl }
1912                }
1913            }
1914
1915    % Get type-specific separators, refpre/pos and font options, once per
1916    % type.
1917    \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1918        {
1919            \__zrefclever_get_ref_string:nN { namesep      }
1920            \l__zrefclever_namesep_tl
1921            \__zrefclever_get_ref_string:nN { rangesep     }
1922            \l__zrefclever_rangesep_tl
1923            \__zrefclever_get_ref_string:nN { pairsep      }
1924            \l__zrefclever_pairsep_tl
1925            \__zrefclever_get_ref_string:nN { listsep       }
1926            \l__zrefclever_listsep_tl
1927            \__zrefclever_get_ref_string:nN { lastsep       }
1928            \l__zrefclever_lastsep_tl
1929            \__zrefclever_get_ref_string:nN { refpre        }
1930            \l__zrefclever_refpre_out_tl
1931            \__zrefclever_get_ref_string:nN { refpos        }
1932            \l__zrefclever_refpos_out_tl
1933            \__zrefclever_get_ref_string:nN { refpre-in     }
1934            \l__zrefclever_refpre_in_tl
1935            \__zrefclever_get_ref_string:nN { refpos-in     }
1936            \l__zrefclever_refpos_in_tl
1937            \__zrefclever_get_ref_font:nN   { namefont      }
1938            \l__zrefclever_namefont_tl
1939            \__zrefclever_get_ref_font:nN   { reffont       }
1940            \l__zrefclever_reffont_out_tl
1941            \__zrefclever_get_ref_font:nN   { reffont-in    }
1942            \l__zrefclever_reffont_in_tl
1943        }
1944
1945    % Here we send this to a couple of auxiliary functions.
1946    \bool_if:NTF \l__zrefclever_last_of_type_bool

```

```

1947         % There exists no next label of the same type as the current.
1948         { \_zrefclever_typeset_refs_last_of_type: }
1949         % There exists a next label of the same type as the current.
1950         { \_zrefclever_typeset_refs_not_last_of_type: }
1951     }
1952 }

```

(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.

```

1953 \cs_new_protected:Npn \_zrefclever_typeset_refs_last_of_type:
1954 {
1955     % Process the current label to the current queue.
1956     \int_case:nnF { \l_zrefclever_label_count_int }
1957     {
1958         % It is the last label of its type, but also the first one, and that's
1959         % what matters here: just store it.
1960         { 0 }
1961         {
1962             \tl_set:NV \l_zrefclever_type_first_label_tl
1963             \l_zrefclever_label_a_tl
1964             \tl_set:NV \l_zrefclever_type_first_label_type_tl
1965             \l_zrefclever_label_type_a_tl
1966         }
1967
1968         % The last is the second: we have a pair (if not repeated).
1969         { 1 }
1970         {
1971             \int_compare:nNnF { \l_zrefclever_range_same_count_int } = { 1 }
1972             {
1973                 \tl_put_right:Nx \l_zrefclever_typeset_queue_curr_tl
1974                 {
1975                     \exp_not:V \l_zrefclever_pairsep_tl
1976                     \_zrefclever_get_ref:V \l_zrefclever_label_a_tl
1977                 }
1978             }
1979         }
1980     }
1981     % Last is third or more of its type: without repetition, we'd have the
1982     % last element on a list, but control for possible repetition.
1983     {
1984         \int_case:nnF { \l_zrefclever_range_count_int }
1985         {
1986             % There was no range going on.
1987             { 0 }

```



```

1988 {
1989   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1990   {
1991     \exp_not:V \l__zrefclever_lastsep_tl
1992     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1993   }
1994 }
1995 % Last in the range is also the second in it.
1996 { 1 }
1997 {
1998   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1999   {
2000     % We know 'range_beg_label' is not empty, since this is the
2001     % second element in the range, but the third or more in the
2002     % type list.
2003     \exp_not:V \l__zrefclever_listsep_tl
2004     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2005     \int_compare:nNnF
2006       { \l__zrefclever_range_same_count_int } = { 1 }
2007       {
2008         \exp_not:V \l__zrefclever_lastsep_tl
2009         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2010       }
2011     }
2012   }
2013 }
2014 % Last in the range is third or more in it.
2015 {
2016   \int_case:nnF
2017     {
2018       \l__zrefclever_range_count_int -
2019       \l__zrefclever_range_same_count_int
2020     }
2021     {
2022       % Repetition, not a range.
2023       { 0 }
2024       {
2025         % If 'range_beg_label' is empty, it means it was also the
2026         % first of the type, and hence was already handled.
2027         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2028         {
2029           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2030           {
2031             \exp_not:V \l__zrefclever_lastsep_tl
2032             \__zrefclever_get_ref:V
2033             \l__zrefclever_range_beg_label_tl
2034           }
2035         }
2036       }
2037       % A 'range', but with no skipped value, treat as list.
2038       { 1 }
2039       {
2040         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2041         {

```

```

2042         % Ditto.
2043         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2044         {
2045             \exp_not:V \l__zrefclever_listsep_tl
2046             \__zrefclever_get_ref:V
2047             \l__zrefclever_range_beg_label_tl
2048         }
2049         \exp_not:V \l__zrefclever_lastsep_tl
2050         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2051     }
2052 }
2053 }
2054 {
2055     % An actual range.
2056     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2057     {
2058         % Ditto.
2059         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2060         {
2061             \exp_not:V \l__zrefclever_lastsep_tl
2062             \__zrefclever_get_ref:V
2063             \l__zrefclever_range_beg_label_tl
2064         }
2065         \exp_not:V \l__zrefclever_rangesep_tl
2066         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2067     }
2068 }
2069 }
2070 }
2071
2072 % Handle "range" option. The idea is simple: if the queue is not empty,
2073 % we replace it with the end of the range (or pair). We can still
2074 % retrieve the end of the range from 'label_a' since we know to be
2075 % processing the last label of its type at this point.
2076 \bool_if:NT \l__zrefclever_typeset_range_bool
2077 {
2078     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2079     {
2080         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2081         { }
2082         {
2083             \msg_warning:nxx { zref-clever } { single-element-range }
2084             { \l__zrefclever_type_first_label_type_tl }
2085         }
2086     }
2087     {
2088         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2089         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2090         { }
2091         {
2092             \__zrefclever_labels_in_sequence:nn
2093             { \l__zrefclever_type_first_label_tl }
2094             { \l__zrefclever_label_a_tl }
2095         }
2096     }
2097 }

```

```

2096         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2097         {
2098             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2099             { \exp_not:V \l__zrefclever_pairsep_tl }
2100             { \exp_not:V \l__zrefclever_rangeseq_tl }
2101             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2102         }
2103     }
2104 }
2105
2106 % Now that the type block is finished, we can add the name and the first
2107 % ref to the queue. Also, if "typeset" option is not "both", handle it
2108 % here as well.
2109 \__zrefclever_type_name_setup:
2110 \bool_if:NTF
2111 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2112 {
2113     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2114     { \__zrefclever_get_ref_first: }
2115 }
2116 {
2117     \bool_if:NTF
2118     { \l__zrefclever_typeset_ref_bool }
2119     {
2120         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2121         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2122     }
2123     {
2124         \bool_if:NTF
2125         { \l__zrefclever_typeset_name_bool }
2126         {
2127             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2128             {
2129                 \bool_if:NTF \l__zrefclever_name_in_link_bool
2130                 {
2131                     \exp_not:N \group_begin:
2132                     \exp_not:V \l__zrefclever_namefont_tl
2133                     % It's two '@s', but escaped for DocStrip.
2134                     \exp_not:N \hyper@@link
2135                     {
2136                         \__zrefclever_extract_url_unexp:V
2137                         \l__zrefclever_type_first_label_tl
2138                     }
2139                     {
2140                         \__zrefclever_extract_unexp:Vnn
2141                         \l__zrefclever_type_first_label_tl
2142                         { anchor } { }
2143                     }
2144                     { \exp_not:V \l__zrefclever_type_name_tl }
2145                     \exp_not:N \group_end:
2146                 }
2147                 {
2148                     \exp_not:N \group_begin:
2149                     \exp_not:V \l__zrefclever_namefont_tl

```

```

2150         \exp_not:V \l__zrefclever_type_name_tl
2151         \exp_not:N \group_end:
2152     }
2153 }
2154 }
2155 {
2156     % Logically, this case would correspond to "typeset=none", but
2157     % it should not occur, given that the options are set up to
2158     % typeset either "ref" or "name". Still, leave here a
2159     % sensible fallback, equal to the behavior of "both".
2160     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2161     { \l__zrefclever_get_ref_first: }
2162 }
2163 }
2164 }
2165
2166 % Typeset the previous type, if there is one.
2167 \int_compare:nNt { \l__zrefclever_type_count_int } > { 0 }
2168 {
2169     \int_compare:nNt { \l__zrefclever_type_count_int } > { 1 }
2170     { \l__zrefclever_tlistsep_tl }
2171     \l__zrefclever_typeset_queue_prev_tl
2172 }
2173
2174 % Wrap up loop, or prepare for next iteration.
2175 \bool_if:NTF \l__zrefclever_typeset_last_bool
2176 {
2177     % We are finishing, typeset the current queue.
2178     \int_case:nnF { \l__zrefclever_type_count_int }
2179     {
2180         % Single type.
2181         { 0 }
2182         { \l__zrefclever_typeset_queue_curr_tl }
2183         % Pair of types.
2184         { 1 }
2185         {
2186             \l__zrefclever_tpairsep_tl
2187             \l__zrefclever_typeset_queue_curr_tl
2188         }
2189     }
2190     {
2191         % Last in list of types.
2192         \l__zrefclever_tlastsep_tl
2193         \l__zrefclever_typeset_queue_curr_tl
2194     }
2195     % And nudge in case of multitype reference.
2196     \bool_lazy_all:nT
2197     {
2198         { \l__zrefclever_nudge_enabled_bool }
2199         { \l__zrefclever_nudge_multitype_bool }
2200         { \int_compare_p:nN { \l__zrefclever_type_count_int } > { 1 } }
2201     }
2202     { \msg_warning:nn { zref-clever } { nudge-multitype } }
2203 }

```

```

2204 {
2205     % There are further labels, set variables for next iteration.
2206     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2207         \l__zrefclever_typeset_queue_curr_tl
2208     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2209     \tl_clear:N \l__zrefclever_type_first_label_tl
2210     \tl_clear:N \l__zrefclever_type_first_label_type_tl
2211     \tl_clear:N \l__zrefclever_range_beg_label_tl
2212     \int_zero:N \l__zrefclever_label_count_int
2213     \int_incr:N \l__zrefclever_type_count_int
2214     \int_zero:N \l__zrefclever_range_count_int
2215     \int_zero:N \l__zrefclever_range_same_count_int
2216 }
2217 }

```

(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.

```

2218 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
2219 {
2220     % Signal if next label may form a range with the current one (only
2221     % considered if compression is enabled in the first place).
2222     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2223     \bool_set_false:N \l__zrefclever_next_is_same_bool
2224     \bool_if:NT \l__zrefclever_typeset_compress_bool
2225     {
2226         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2227         { }
2228         {
2229             \__zrefclever_labels_in_sequence:nn
2230             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2231         }
2232     }
2233
2234     % Process the current label to the current queue.
2235     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2236     {
2237         % Current label is the first of its type (also not the last, but it
2238         % doesn't matter here): just store the label.
2239         \tl_set:NV \l__zrefclever_type_first_label_tl
2240             \l__zrefclever_label_a_tl
2241         \tl_set:NV \l__zrefclever_type_first_label_type_tl
2242             \l__zrefclever_label_type_a_tl
2243
2244         % If the next label may be part of a range, we set 'range_beg_label'
2245         % to "empty" (we deal with it as the "first", and must do it there, to
2246         % handle hyperlinking), but also step the range counters.
2247         \bool_if:NT \l__zrefclever_next_maybe_range_bool
2248         {
2249             \tl_clear:N \l__zrefclever_range_beg_label_tl
2250             \int_incr:N \l__zrefclever_range_count_int
2251             \bool_if:NT \l__zrefclever_next_is_same_bool
2252             { \int_incr:N \l__zrefclever_range_same_count_int }
2253         }

```

```

2254 }
2255 {
2256 % Current label is neither the first (nor the last) of its type.
2257 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2258 {
2259 % Starting, or continuing a range.
2260 \int_compare:nNnTF
2261 { \l__zrefclever_range_count_int } = { 0 }
2262 {
2263 % There was no range going, we are starting one.
2264 \tl_set:NV \l__zrefclever_range_beg_label_tl
2265 \l__zrefclever_label_a_tl
2266 \int_incr:N \l__zrefclever_range_count_int
2267 \bool_if:NT \l__zrefclever_next_is_same_bool
2268 { \int_incr:N \l__zrefclever_range_same_count_int }
2269 }
2270 {
2271 % Second or more in the range, but not the last.
2272 \int_incr:N \l__zrefclever_range_count_int
2273 \bool_if:NT \l__zrefclever_next_is_same_bool
2274 { \int_incr:N \l__zrefclever_range_same_count_int }
2275 }
2276 }
2277 {
2278 % Next element is not in sequence: there was no range, or we are
2279 % closing one.
2280 \int_case:nnF { \l__zrefclever_range_count_int }
2281 {
2282 % There was no range going on.
2283 { 0 }
2284 {
2285 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2286 {
2287 \exp_not:V \l__zrefclever_listsep_tl
2288 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2289 }
2290 }
2291 % Last is second in the range: if 'range_same_count' is also
2292 % '1', it's a repetition (drop it), otherwise, it's a "pair
2293 % within a list", treat as list.
2294 { 1 }
2295 {
2296 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2297 {
2298 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2299 {
2300 \exp_not:V \l__zrefclever_listsep_tl
2301 \__zrefclever_get_ref:V
2302 \l__zrefclever_range_beg_label_tl
2303 }
2304 \int_compare:nNnF
2305 { \l__zrefclever_range_same_count_int } = { 1 }
2306 {
2307 \exp_not:V \l__zrefclever_listsep_tl

```

```

2308         \_zrefclever_get_ref:V
2309         \l__zrefclever_label_a_tl
2310     }
2311 }
2312 }
2313 }
2314 {
2315     % Last is third or more in the range: if 'range_count' and
2316     % 'range_same_count' are the same, its a repetition (drop it),
2317     % if they differ by '1', its a list, if they differ by more,
2318     % it is a real range.
2319     \int_case:nnF
2320     {
2321         \l__zrefclever_range_count_int -
2322         \l__zrefclever_range_same_count_int
2323     }
2324     {
2325         { 0 }
2326         {
2327             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2328             {
2329                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2330                 {
2331                     \exp_not:V \l__zrefclever_listsep_tl
2332                     \_zrefclever_get_ref:V
2333                     \l__zrefclever_range_beg_label_tl
2334                 }
2335             }
2336         }
2337         { 1 }
2338         {
2339             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2340             {
2341                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2342                 {
2343                     \exp_not:V \l__zrefclever_listsep_tl
2344                     \_zrefclever_get_ref:V
2345                     \l__zrefclever_range_beg_label_tl
2346                 }
2347                 \exp_not:V \l__zrefclever_listsep_tl
2348                 \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
2349             }
2350         }
2351     }
2352     {
2353         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2354         {
2355             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2356             {
2357                 \exp_not:V \l__zrefclever_listsep_tl
2358                 \_zrefclever_get_ref:V
2359                 \l__zrefclever_range_beg_label_tl
2360             }
2361             \exp_not:V \l__zrefclever_rangeseq_tl

```

```

2362             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
2363         }
2364     }
2365 }
2366 % Reset counters.
2367 \int_zero:N \l__zrefclever_range_count_int
2368 \int_zero:N \l__zrefclever_range_same_count_int
2369 }
2370 }
2371 % Step label counter for next iteration.
2372 \int_incr:N \l__zrefclever_label_count_int
2373 }

```

(End definition for \\_zrefclever\_typeset\_refs\_not\_last\_of\_type:.)

## Aux functions

\\_zrefclever\_get\_ref:n and \\_zrefclever\_get\_ref\_first: are the two functions which actually build the reference blocks for typesetting. \\_zrefclever\_get\_ref:n handles all references but the first of its type, and \\_zrefclever\_get\_ref\_first: deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in \l\_\_zrefclever\_typeset\_queue\_curr\_tl inside \\_zrefclever\_typeset\_refs\_last\_of\_type: and \\_zrefclever\_typeset\_refs\_not\_last\_of\_type:. And this difference results quite crucial for the  $\TeX$ nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, \\_zrefclever\_get\_ref:n and \\_zrefclever\_get\_ref\_first: get called, as they must, in the context of  $x$  type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the  $n$  signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, \zref@default or \hyper@@link). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```

2374 \cs_new_protected:Npn \_zrefclever_ref_default:
2375 { \zref@default }
2376 \cs_new_protected:Npn \_zrefclever_name_default:
2377 { \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>}

2378 \cs_new:Npn \_zrefclever_get_ref:n #1
2379 {
2380   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2381   {
2382     \bool_if:nTF
2383     {
2384       \l__zrefclever_use_hyperref_bool &&
2385       ! \l__zrefclever_link_star_bool
2386     }
2387     {
2388       \exp_not:N \group_begin:
2389       \exp_not:V \l__zrefclever_reffont_out_tl
2390       \exp_not:V \l__zrefclever_refpre_out_tl
2391       \exp_not:N \group_begin:
2392       \exp_not:V \l__zrefclever_reffont_in_tl
2393       % It's two '@s', but escaped for DocStrip.
2394       \exp_not:N \hyper@@link
2395       { \_zrefclever_extract_url_unexp:n {#1} }
2396       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2397       {
2398         \exp_not:V \l__zrefclever_refpre_in_tl
2399         \_zrefclever_extract_unexp:nvn {#1}
2400         { \l__zrefclever_ref_property_tl } { }
2401         \exp_not:V \l__zrefclever_refpos_in_tl
2402       }
2403       \exp_not:N \group_end:
2404       \exp_not:V \l__zrefclever_refpos_out_tl
2405       \exp_not:N \group_end:
2406     }
2407     {
2408       \exp_not:N \group_begin:
2409       \exp_not:V \l__zrefclever_reffont_out_tl
2410       \exp_not:V \l__zrefclever_refpre_out_tl
2411       \exp_not:N \group_begin:
2412       \exp_not:V \l__zrefclever_reffont_in_tl
2413       \exp_not:V \l__zrefclever_refpre_in_tl
2414       \_zrefclever_extract_unexp:nvn {#1}
2415       { \l__zrefclever_ref_property_tl } { }
2416       \exp_not:V \l__zrefclever_refpos_in_tl
2417       \exp_not:N \group_end:
2418       \exp_not:V \l__zrefclever_refpos_out_tl
2419       \exp_not:N \group_end:
2420     }
2421   }
2422   { \_zrefclever_ref_default: }
2423 }
2424 \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.

```

2425 \cs_new:Npn \__zrefclever_get_ref_first:
2426 {
2427   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2428   { \__zrefclever_ref_default: }
2429   {
2430     \bool_if:NTF \l__zrefclever_name_in_link_bool
2431     {
2432       \zref@ifrefcontainsprop
2433       { \l__zrefclever_type_first_label_tl }
2434       { \l__zrefclever_ref_property_tl }
2435       {
2436         % It's two '@s', but escaped for DocStrip.
2437         \exp_not:N \hyper@@link
2438         {
2439           \__zrefclever_extract_url_unexp:V
2440           \l__zrefclever_type_first_label_tl
2441         }
2442         {
2443           \__zrefclever_extract_unexp:Vnn
2444           \l__zrefclever_type_first_label_tl { anchor } { }
2445         }
2446         {
2447           \exp_not:N \group_begin:
2448           \exp_not:V \l__zrefclever_namefont_tl
2449           \exp_not:V \l__zrefclever_type_name_tl
2450           \exp_not:N \group_end:
2451           \exp_not:V \l__zrefclever_namesep_tl
2452           \exp_not:N \group_begin:
2453           \exp_not:V \l__zrefclever_reffont_out_tl
2454           \exp_not:V \l__zrefclever_refpre_out_tl
2455           \exp_not:N \group_begin:
2456           \exp_not:V \l__zrefclever_reffont_in_tl
2457           \exp_not:V \l__zrefclever_refpre_in_tl
2458           \__zrefclever_extract_unexp:Vnn
2459           \l__zrefclever_type_first_label_tl
2460           { \l__zrefclever_ref_property_tl } { }
2461           \exp_not:V \l__zrefclever_refpos_in_tl
2462           \exp_not:N \group_end:
2463           % hyperlink makes it's own group, we'd like to close the
2464           % 'refpre-out' group after 'refpos-out', but... we close
2465           % it here, and give the trailing 'refpos-out' its own
2466           % group. This will result that formatting given to
2467           % 'refpre-out' will not reach 'refpos-out', but I see no
2468           % alternative, and this has to be handled specially.
2469           \exp_not:N \group_end:
2470         }
2471       }

```

```

2472         % Ditto: special treatment.
2473         \exp_not:V \l__zrefclever_reffont_out_tl
2474         \exp_not:V \l__zrefclever_refpos_out_tl
2475         \exp_not:N \group_end:
2476     }
2477     {
2478         \exp_not:N \group_begin:
2479         \exp_not:V \l__zrefclever_namefont_tl
2480         \exp_not:V \l__zrefclever_type_name_tl
2481         \exp_not:N \group_end:
2482         \exp_not:V \l__zrefclever_namesep_tl
2483         \__zrefclever_ref_default:
2484     }
2485 }
2486 {
2487     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2488     {
2489         \__zrefclever_name_default:
2490         \exp_not:V \l__zrefclever_namesep_tl
2491     }
2492     {
2493         \exp_not:N \group_begin:
2494         \exp_not:V \l__zrefclever_namefont_tl
2495         \exp_not:V \l__zrefclever_type_name_tl
2496         \exp_not:N \group_end:
2497         \exp_not:V \l__zrefclever_namesep_tl
2498     }
2499     \zref@ifrefcontainsprop
2500     { \l__zrefclever_type_first_label_tl }
2501     { \l__zrefclever_ref_property_tl }
2502     {
2503         \bool_if:nTF
2504         {
2505             \l__zrefclever_use_hyperref_bool &&
2506             ! \l__zrefclever_link_star_bool
2507         }
2508         {
2509             \exp_not:N \group_begin:
2510             \exp_not:V \l__zrefclever_reffont_out_tl
2511             \exp_not:V \l__zrefclever_refpre_out_tl
2512             \exp_not:N \group_begin:
2513             \exp_not:V \l__zrefclever_reffont_in_tl
2514             % It's two '@s', but escaped for DocStrip.
2515             \exp_not:N \hyper@@link
2516             {
2517                 \__zrefclever_extract_url_unexp:V
2518                 \l__zrefclever_type_first_label_tl
2519             }
2520             {
2521                 \__zrefclever_extract_unexp:Vnn
2522                 \l__zrefclever_type_first_label_tl { anchor } { }
2523             }
2524             {
2525                 \exp_not:V \l__zrefclever_refpre_in_tl

```

```

2526         \__zrefclever_extract_unexp:Vvn
2527         \l__zrefclever_type_first_label_tl
2528         { \l__zrefclever_ref_property_tl } { }
2529         \exp_not:V \l__zrefclever_refpos_in_tl
2530     }
2531     \exp_not:N \group_end:
2532     \exp_not:V \l__zrefclever_refpos_out_tl
2533     \exp_not:N \group_end:
2534 }
2535 {
2536     \exp_not:N \group_begin:
2537     \exp_not:V \l__zrefclever_reffont_out_tl
2538     \exp_not:V \l__zrefclever_refpre_out_tl
2539     \exp_not:N \group_begin:
2540     \exp_not:V \l__zrefclever_reffont_in_tl
2541     \exp_not:V \l__zrefclever_refpre_in_tl
2542     \__zrefclever_extract_unexp:Vvn
2543     \l__zrefclever_type_first_label_tl
2544     { \l__zrefclever_ref_property_tl } { }
2545     \exp_not:V \l__zrefclever_refpos_in_tl
2546     \exp_not:N \group_end:
2547     \exp_not:V \l__zrefclever_refpos_out_tl
2548     \exp_not:N \group_end:
2549 }
2550 }
2551 { \__zrefclever_ref_default: }
2552 }
2553 }
2554 }

```

(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.

```

2555 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2556 {
2557     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2558     { \tl_clear:N \l__zrefclever_type_name_tl }
2559     {
2560         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2561         { \tl_clear:N \l__zrefclever_type_name_tl }
2562         {
2563             % Determine whether we should use capitalization, abbreviation,
2564             % and plural.

```

```

2565 \bool_lazy_or:nnTF
2566   { \l__zrefclever_capitalize_bool }
2567   {
2568     \l__zrefclever_capitalize_first_bool &&
2569     \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2570   }
2571   { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2572   { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2573 % If the queue is empty, we have a singular, otherwise, plural.
2574 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2575   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2576   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2577 \bool_lazy_and:nnTF
2578   { \l__zrefclever_abbrev_bool }
2579   {
2580     ! \int_compare_p:nNn
2581       { \l__zrefclever_type_count_int } = { 0 } ||
2582     ! \l__zrefclever_noabbrev_first_bool
2583   }
2584   {
2585     \tl_set:NV \l__zrefclever_name_format_fallback_tl
2586       \l__zrefclever_name_format_tl
2587     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2588   }
2589   { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2590
2591 % Handle singular/plural nudges.
2592 \bool_if:NT \l__zrefclever_nudge_enabled_bool
2593   {
2594     \bool_if:NTF \l__zrefclever_nudge_singular_bool
2595       {
2596         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2597         {
2598           \msg_warning:nxx { zref-clever }
2599             { nudge-plural-when-sg }
2600             { \l__zrefclever_type_first_label_type_tl }
2601         }
2602       }
2603     {
2604       \bool_lazy_all:nT
2605         {
2606           { \l__zrefclever_nudge_comptosing_bool }
2607           { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2608         }
2609         {
2610           \int_compare_p:nNn
2611             { \l__zrefclever_label_count_int } > { 0 }
2612         }
2613       }
2614     {
2615       \msg_warning:nxx { zref-clever }
2616         { nudge-comptosing }
2617         { \l__zrefclever_type_first_label_type_tl }
2618     }
2619   }

```

```

2619     }
2620
2621 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2622 {
2623     \prop_get:cVNF
2624     {
2625         l__zrefclever_type_
2626         \l__zrefclever_type_first_label_type_tl _options_prop
2627     }
2628     \l__zrefclever_name_format_tl
2629     \l__zrefclever_type_name_tl
2630     {
2631         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2632         {
2633             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
2634             \tl_put_left:NV \l__zrefclever_name_format_tl
2635                 \l__zrefclever_ref_decl_case_tl
2636         }
2637         \__zrefclever_get_type_transl:xxxNF
2638         { \l__zrefclever_ref_language_tl }
2639         { \l__zrefclever_type_first_label_type_tl }
2640         { \l__zrefclever_name_format_tl }
2641         \l__zrefclever_type_name_tl
2642         {
2643             \tl_clear:N \l__zrefclever_type_name_tl
2644             \msg_warning:nxxx { zref-clever } { missing-name }
2645             { \l__zrefclever_name_format_tl }
2646             { \l__zrefclever_type_first_label_type_tl }
2647         }
2648     }
2649 }
2650 {
2651     \prop_get:cVNF
2652     {
2653         l__zrefclever_type_
2654         \l__zrefclever_type_first_label_type_tl _options_prop
2655     }
2656     \l__zrefclever_name_format_tl
2657     \l__zrefclever_type_name_tl
2658     {
2659         \prop_get:cVNF
2660         {
2661             l__zrefclever_type_
2662             \l__zrefclever_type_first_label_type_tl _options_prop
2663         }
2664         \l__zrefclever_name_format_fallback_tl
2665         \l__zrefclever_type_name_tl
2666         {
2667             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2668             {
2669                 \tl_put_left:Nn
2670                     \l__zrefclever_name_format_tl { - }
2671                 \tl_put_left:NV \l__zrefclever_name_format_tl
2672                     \l__zrefclever_ref_decl_case_tl

```

```

2673         \tl_put_left:Nn
2674         \l__zrefclever_name_format_fallback_tl { - }
2675     \tl_put_left:NV
2676         \l__zrefclever_name_format_fallback_tl
2677         \l__zrefclever_ref_decl_case_tl
2678     }
2679     \__zrefclever_get_type_transl:xxxNF
2680     { \l__zrefclever_ref_language_tl }
2681     { \l__zrefclever_type_first_label_type_tl }
2682     { \l__zrefclever_name_format_tl }
2683     \l__zrefclever_type_name_tl
2684     {
2685         \__zrefclever_get_type_transl:xxxNF
2686         { \l__zrefclever_ref_language_tl }
2687         { \l__zrefclever_type_first_label_type_tl }
2688         { \l__zrefclever_name_format_fallback_tl }
2689         \l__zrefclever_type_name_tl
2690         {
2691             \tl_clear:N \l__zrefclever_type_name_tl
2692             \msg_warning:nxxx { zref-clever }
2693             { missing-name }
2694             { \l__zrefclever_name_format_tl }
2695             { \l__zrefclever_type_first_label_type_tl }
2696         }
2697     }
2698 }
2699 }
2700 }
2701 }
2702 }
2703
2704 % Signal whether the type name is to be included in the hyperlink or not.
2705 \bool_lazy_any:nTF
2706 {
2707     { ! \l__zrefclever_use_hyperref_bool }
2708     { \l__zrefclever_link_star_bool }
2709     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2710     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2711 }
2712 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2713 {
2714     \bool_lazy_any:nTF
2715     {
2716         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2717         {
2718             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2719             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2720         }
2721         {
2722             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2723             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2724             \l__zrefclever_typeset_last_bool &&
2725             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2726         }
2727     }

```

```

2727     }
2728     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2729     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2730   }
2731 }

```

(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.

```

2732 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2733 {
2734   \zref@ifpropundefined { urluse }
2735   { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2736   {
2737     \zref@ifrefcontainsprop {#1} { urluse }
2738     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2739     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2740   }
2741 }
2742 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for \\_\_zrefclever\_extract\_url\_unexp:n.)

\\_\_zrefclever\_labels\_in\_sequence:nn Auxiliary function to \\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type:. Sets \l\_\_zrefclever\_next\_maybe\_range\_bool to true if <label b> comes in immediate sequence from <label a>. And sets both \l\_\_zrefclever\_next\_maybe\_range\_bool and \l\_\_zrefclever\_next\_is\_same\_bool to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside \\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type:, so this function is expected to be called at its beginning, if compression is enabled.

```

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

2743 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2744 {
2745   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2746   {#1} { externaldocument } { \c_empty_tl }
2747   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2748   {#2} { externaldocument } { \c_empty_tl }
2749
2750   \tl_if_eq:NNT
2751   \l__zrefclever_label_extdoc_a_tl
2752   \l__zrefclever_label_extdoc_b_tl
2753   {
2754     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2755     {
2756       \exp_args:Nxx \tl_if_eq:nnT
2757       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2758       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2759       {
2760         \int_compare:nNnTF

```



```

2761 { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2762 =
2763 { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2764 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2765 {
2766   \int_compare:nNnT
2767     { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2768     =
2769     { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2770     {
2771       \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2772       \bool_set_true:N \l__zrefclever_next_is_same_bool
2773     }
2774   }
2775 }
2776 }
2777 {
2778   \exp_args:Nxx \tl_if_eq:nnT
2779   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2780   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2781   {
2782     \exp_args:Nxx \tl_if_eq:nnT
2783     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2784     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2785     {
2786       \int_compare:nNnTF
2787         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2788         =
2789         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2790         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2791         {
2792           \int_compare:nNnT
2793             { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2794             =
2795             { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2796             {
2797               \bool_set_true:N
2798                 \l__zrefclever_next_maybe_range_bool
2799               \exp_args:Nxx \tl_if_eq:nnT
2800               {
2801                 \__zrefclever_extract_unexp:nvn {#1}
2802                 { l__zrefclever_ref_property_tl } { }
2803               }
2804               {
2805                 \__zrefclever_extract_unexp:nvn {#2}
2806                 { l__zrefclever_ref_property_tl } { }
2807               }
2808               {
2809                 \bool_set_true:N
2810                   \l__zrefclever_next_is_same_bool
2811               }
2812             }
2813           }
2814         }

```

```

2815         }
2816     }
2817 }
2818 }

```

(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 “fallback”). `\_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>}
2819 \cs_new_protected:Npn \_zrefclever_get_ref_string:nN #1#2
2820 {
2821   % First attempt: general options.
2822   \prop_get:NnNF \l_zrefclever_ref_options_prop {#1} #2
2823   {
2824     % If not found, try type specific options.
2825     \bool_lazy_all:nTF
2826     {
2827       { ! \tl_if_empty_p:N \l_zrefclever_label_type_a_tl }
2828       {
2829         \prop_if_exist_p:c
2830         {
2831           l_zrefclever_type_
2832           \l_zrefclever_label_type_a_tl _options_prop
2833         }
2834       }
2835       {
2836         \prop_if_in_p:cn
2837         {
2838           l_zrefclever_type_
2839           \l_zrefclever_label_type_a_tl _options_prop
2840         }
2841         {#1}
2842       }
2843     }
2844     {
2845       \prop_get:cnN
2846       {
2847         l_zrefclever_type_
2848         \l_zrefclever_label_type_a_tl _options_prop
2849       }
2850       {#1} #2
2851     }

```

```

2852     {
2853       % If not found, try type specific translations.
2854       \__zrefclever_get_type_transl:xnNF
2855       { \l__zrefclever_ref_language_tl }
2856       { \l__zrefclever_label_type_a_tl }
2857       {#1} #2
2858       {
2859         % If not found, try default translations.
2860         \__zrefclever_get_default_transl:xnNF
2861         { \l__zrefclever_ref_language_tl }
2862         {#1} #2
2863         {
2864           % If not found, try fallback.
2865           \__zrefclever_get_fallback_transl:nNF {#1} #2
2866           {
2867             \tl_clear:N #2
2868             \msg_warning:nnn { zref-clever }
2869             { missing-string } {#1}
2870           }
2871         }
2872       }
2873     }
2874   }
2875 }

```

(End definition for \\_\_zrefclever\_get\_ref\_string:nN.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2876 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2877 {
2878   % First attempt: general options.
2879   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2880   {
2881     % If not found, try type specific options.
2882     \bool_lazy_and:nnTF
2883     { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2884     {
2885       \prop_if_exist_p:c
2886       {
2887         l__zrefclever_type_
2888         \l__zrefclever_label_type_a_tl _options_prop
2889       }
2890     }
2891     {
2892       \prop_get:cnNF
2893       {
2894         l__zrefclever_type_
2895         \l__zrefclever_label_type_a_tl _options_prop
2896       }
2897       {#1} #2
2898       { \tl_clear:N #2 }
2899     }
2900   } { \tl_clear:N #2 }
2901 }

```

```
2902 }
```

(End definition for `\__zrefclever_get_ref_font:nN`.)

## 9 Compatibility

This section is meant to aggregate any “special handling” needed for L<sup>A</sup>T<sub>E</sub>X 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.

```
2903 \tl_new:N \l__zrefclever_footnote_type_tl
2904 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
2905 \AddToHook { env / minipage / begin }
2906 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
2907 \AddToHook { cmd / @makeintext / before }
2908 {
2909   \__zrefclever_zcsetup:x
2910   { currentcounter = \l__zrefclever_footnote_type_tl }
2911 }
```

### 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.

```
2912 \AddToHook { cmd / appendix / before }
2913 {
2914   \__zrefclever_zcsetup:n
2915   {
```

```

2916     countertype =
2917     {
2918         chapter      = appendix ,
2919         section      = appendix ,
2920         subsection   = appendix ,
2921         subsubsection = appendix ,
2922     }
2923 }
2924 }

```

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

```

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

```

### 9.3 appendix package

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

```

2925 \AddToHook { begindocument }
2926 {
2927     \@ifpackageloaded { appendix }
2928     {
2929         \newcounter { zc@appendix }
2930         \newcounter { zc@save@appendix }
2931         \setcounter { zc@appendix } { 0 }
2932         \setcounter { zc@save@appendix } { 0 }
2933         \cs_if_exist:cTF { chapter }
2934         {
2935             \__zrefclever_zcsetup:n
2936             { counterresetby = { chapter = zc@appendix } }
2937         }
2938         {
2939             \cs_if_exist:cT { section }
2940             {

```

```

2941         \__zrefclever_zcsetup:n
2942         { counterresetby = { section = zc@appendix } }
2943     }
2944 }
2945 \AddToHook { env / appendices / begin }
2946 {
2947     \stepcounter { zc@save@appendix }
2948     \setcounter { zc@appendix } { \value { zc@save@appendix } }
2949     \__zrefclever_zcsetup:n
2950     {
2951         countertype =
2952         {
2953             chapter      = appendix ,
2954             section      = appendix ,
2955             subsection    = appendix ,
2956             subsubsection = appendix ,
2957         }
2958     }
2959 }
2960 \AddToHook { env / appendices / end }
2961 { \setcounter { zc@appendix } { 0 } }
2962 \AddToHook { cmd / appendix / before }
2963 {
2964     \stepcounter { zc@save@appendix }
2965     \setcounter { zc@appendix } { \value { zc@save@appendix } }
2966 }
2967 \AddToHook { env / subappendices / begin }
2968 {
2969     \__zrefclever_zcsetup:n
2970     {
2971         countertype =
2972         {
2973             section      = appendix ,
2974             subsection    = appendix ,
2975             subsubsection = appendix ,
2976         } ,
2977     }
2978 }
2979 \msg_info:nnn { zref-clever } { compat-package } { appendix }
2980 }
2981 {}
2982 }

```

## 9.4 amsmath package

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

```

2983 \AddToHook { begindocument }
2984 {
2985     \@ifpackageloaded { amsmath }
2986     {

```

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...).

```

2987     \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
2988     {
2989         \__zrefclever_orig_ltxlabel:n {#1}
2990         \zref@wrapper@babel \zref@label {#1}
2991     }

```

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

```

2992     \IfFormatAtLeastTF { 2021-11-15 }
2993     {
2994         \@ifpackageloaded { hyperref }
2995         {
2996             \AddToHook { package / nameref / after }
2997             {
2998                 \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2999                 \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3000             }
3001         }
3002         {
3003             \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3004             \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3005         }
3006     }
3007     {
3008         \@ifpackageloaded { hyperref }
3009         {
3010             \@ifpackageloaded { nameref }
3011             {
3012                 \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3013                 \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3014             }
3015             {
3016                 \AddToHook { package / after / nameref }
3017                 {
3018                     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3019                     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n

```

```

3020         }
3021     }
3022 }
3023 {
3024     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3025     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3026 }
3027 }

```

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).

```

3028 \AddToHook { env / subequations / begin }
3029 {
3030     \__zrefclever_zcsetup:x
3031     {
3032         counterresetby =
3033         {
3034             parentequation =
3035             \__zrefclever_counter_reset_by:n { equation } ,
3036             equation = parentequation ,
3037         } ,
3038         currentcounter = parentequation ,
3039         countertype = { parentequation = equation } ,
3040     }
3041 }

```

`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.

```

3042 \clist_map_inline:nn
3043 {
3044     equation ,
3045     equation* ,
3046     align ,
3047     align* ,
3048     alignat ,
3049     alignat* ,

```



```

3050         flalign ,
3051         flalign* ,
3052         xalignat ,
3053         xalignat* ,
3054         gather ,
3055         gather* ,
3056         multiline ,
3057         multiline* ,
3058     }
3059     {
3060         \AddToHook { env / #1 / begin }
3061         { \_zrefclever_zcsetup:n { currentcounter = equation } }
3062     }

```

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

```

3063         \zcRefTypeSetup { equation } { reffont = \upshape }
3064         \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3065     }
3066     {}
3067 }

```

## 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.

```

3068 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3069 \AddToHook { begindocument }
3070 {
3071     \@ifpackageloaded { mathtools }
3072     {
3073         \MH_if_boolean:nT { show_only_refs }
3074         {
3075             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3076             \cs_new_protected:Npn \_zrefclever_mathtools_showonlyrefs:n #1
3077             {
3078                 \@bsphack
3079                 \seq_map_inline:Nn #1
3080                 {
3081                     \exp_args:Nx \tl_if_eq:nnTF
3082                     { \_zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3083                     { equation }
3084                     {
3085                         \protected@write \@auxout { }
3086                         { \string \MT@newlabel {##1} }
3087                     }
3088                 }
3089             }
3090         }
3091     }

```

```

3088         {
3089             \exp_args:Nx \tl_if_eq:nnT
3090             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3091             { parentequation }
3092             {
3093                 \protected@write \@auxout { }
3094                 { \string \MT@newlabel {##1} }
3095             }
3096         }
3097     }
3098     \@esphack
3099 }
3100 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3101 }
3102 {}
3103 {}
3104 }

```

## 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”.

```

3105 \AddToHook { begindocument }
3106 {
3107     \@ifpackageloaded { breqn }
3108     {

```

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.

```

3109     \AddToHook { env / dgroup / begin }
3110     {
3111         \__zrefclever_zcsetup:x
3112         {
3113             counterresetby =
3114             {
3115                 parentequation =
3116                 \__zrefclever_counter_reset_by:n { equation } ,
3117                 equation = parentequation ,
3118             } ,
3119             currentcounter = parentequation ,
3120             countertype = { parentequation = equation } ,
3121         }
3122     }
3123     \clist_map_inline:nn
3124     {
3125         dmath ,
3126         dseries ,

```

```

3127         darray ,
3128     }
3129     {
3130         \AddToHook { env / #1 / begin }
3131             { \__zrefclever_zcsetup:n { currentcounter = equation } }
3132     }
3133 }
3134 {}
3135 }

```

## 9.7 listings package

```

3136 \AddToHook { begindocument }
3137 {
3138     \@ifpackageloaded { listings }
3139     {
3140         \__zrefclever_zcsetup:n
3141         {
3142             countertype =
3143             {
3144                 lstlisting = listing ,
3145                 lstnumber = line ,
3146             } ,
3147             counterresetby = { lstnumber = lstlisting } ,
3148         }
3149         \lst@AddToHook { Init }
3150         {

```

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

```

3151         \tl_if_empty:NF \lst@label
3152         { \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.

```

3153         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
3154     }
3155     \msg_info:nnn { zref-clever } { compat-package } { listings }
3156 }
3157 {}
3158 }

```

## 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.

```

3159 \AddToHook { begindocument }
3160 {
3161   \@ifpackageloaded { enumitem }
3162   {
3163     \int_set:Nn \l_tmpa_int { 5 }
3164     \bool_while_do:nn
3165     {
3166       \cs_if_exist_p:c
3167       { c@ enum \int_to_roman:n { \l_tmpa_int } }
3168     }
3169     {
3170       \__zrefclever_zcsetup:x
3171       {
3172         counterresetby =
3173         {
3174           enum \int_to_roman:n { \l_tmpa_int } =
3175           enum \int_to_roman:n { \l_tmpa_int - 1 }
3176         } ,
3177         countertype =
3178         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3179       }
3180       \int_incr:N \l_tmpa_int
3181     }
3182     \int_compare:nNnT { \l_tmpa_int } > { 5 }
3183     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3184   }
3185   {}
3186 }
3187 \end{package}

```

## 10 Dictionaries

### 10.1 English

```

3188 <package>\zcDeclareLanguage { english }
3189 <package>\zcDeclareLanguageAlias { american } { english }
3190 <package>\zcDeclareLanguageAlias { australian } { english }
3191 <package>\zcDeclareLanguageAlias { british } { english }
3192 <package>\zcDeclareLanguageAlias { canadian } { english }
3193 <package>\zcDeclareLanguageAlias { newzealand } { english }
3194 <package>\zcDeclareLanguageAlias { UKenglish } { english }
3195 <package>\zcDeclareLanguageAlias { USenglish } { english }
3196 <*dict-english>

```

```

3197 namesep   = {\nobreakspace} ,
3198 pairsep    = {\~and\nobreakspace} ,
3199 listsep     = {\~,~} ,
3200 lastsep     = {\~and\nobreakspace} ,
3201 tpairsep    = {\~and\nobreakspace} ,
3202 tlistsep    = {\~,~} ,
3203 tlastsep    = {\~,~and\nobreakspace} ,
3204 notesep     = {\~} ,
3205 rangesep    = {\~to\nobreakspace} ,
3206
3207 type = part ,
3208     Name-sg = Part ,
3209     name-sg = part ,
3210     Name-pl = Parts ,
3211     name-pl = parts ,
3212
3213 type = chapter ,
3214     Name-sg = Chapter ,
3215     name-sg = chapter ,
3216     Name-pl = Chapters ,
3217     name-pl = chapters ,
3218
3219 type = section ,
3220     Name-sg = Section ,
3221     name-sg = section ,
3222     Name-pl = Sections ,
3223     name-pl = sections ,
3224
3225 type = paragraph ,
3226     Name-sg = Paragraph ,
3227     name-sg = paragraph ,
3228     Name-pl = Paragraphs ,
3229     name-pl = paragraphs ,
3230     Name-sg-ab = Par. ,
3231     name-sg-ab = par. ,
3232     Name-pl-ab = Par. ,
3233     name-pl-ab = par. ,
3234
3235 type = appendix ,
3236     Name-sg = Appendix ,
3237     name-sg = appendix ,
3238     Name-pl = Appendices ,
3239     name-pl = appendices ,
3240
3241 type = subappendix ,
3242     Name-sg = Appendix ,
3243     name-sg = appendix ,
3244     Name-pl = Appendices ,
3245     name-pl = appendices ,
3246
3247 type = page ,
3248     Name-sg = Page ,
3249     name-sg = page ,
3250     Name-pl = Pages ,

```

```

3251     name-pl = pages ,
3252     name-sg-ab = p. ,
3253     name-pl-ab = pp. ,
3254
3255     type = line ,
3256     Name-sg = Line ,
3257     name-sg = line ,
3258     Name-pl = Lines ,
3259     name-pl = lines ,
3260
3261     type = figure ,
3262     Name-sg = Figure ,
3263     name-sg = figure ,
3264     Name-pl = Figures ,
3265     name-pl = figures ,
3266     Name-sg-ab = Fig. ,
3267     name-sg-ab = fig. ,
3268     Name-pl-ab = Figs. ,
3269     name-pl-ab = figs. ,
3270
3271     type = table ,
3272     Name-sg = Table ,
3273     name-sg = table ,
3274     Name-pl = Tables ,
3275     name-pl = tables ,
3276
3277     type = item ,
3278     Name-sg = Item ,
3279     name-sg = item ,
3280     Name-pl = Items ,
3281     name-pl = items ,
3282
3283     type = footnote ,
3284     Name-sg = Footnote ,
3285     name-sg = footnote ,
3286     Name-pl = Footnotes ,
3287     name-pl = footnotes ,
3288
3289     type = note ,
3290     Name-sg = Note ,
3291     name-sg = note ,
3292     Name-pl = Notes ,
3293     name-pl = notes ,
3294
3295     type = equation ,
3296     Name-sg = Equation ,
3297     name-sg = equation ,
3298     Name-pl = Equations ,
3299     name-pl = equations ,
3300     Name-sg-ab = Eq. ,
3301     name-sg-ab = eq. ,
3302     Name-pl-ab = Eqs. ,
3303     name-pl-ab = eqs. ,
3304     refpre-in = { ( } ,

```

```

3305     refpos-in = {} } ,
3306
3307 type = theorem ,
3308     Name-sg = Theorem ,
3309     name-sg = theorem ,
3310     Name-pl = Theorems ,
3311     name-pl = theorems ,
3312
3313 type = lemma ,
3314     Name-sg = Lemma ,
3315     name-sg = lemma ,
3316     Name-pl = Lemmas ,
3317     name-pl = lemmas ,
3318
3319 type = corollary ,
3320     Name-sg = Corollary ,
3321     name-sg = corollary ,
3322     Name-pl = Corollaries ,
3323     name-pl = corollaries ,
3324
3325 type = proposition ,
3326     Name-sg = Proposition ,
3327     name-sg = proposition ,
3328     Name-pl = Propositions ,
3329     name-pl = propositions ,
3330
3331 type = definition ,
3332     Name-sg = Definition ,
3333     name-sg = definition ,
3334     Name-pl = Definitions ,
3335     name-pl = definitions ,
3336
3337 type = proof ,
3338     Name-sg = Proof ,
3339     name-sg = proof ,
3340     Name-pl = Proofs ,
3341     name-pl = proofs ,
3342
3343 type = result ,
3344     Name-sg = Result ,
3345     name-sg = result ,
3346     Name-pl = Results ,
3347     name-pl = results ,
3348
3349 type = remark ,
3350     Name-sg = Remark ,
3351     name-sg = remark ,
3352     Name-pl = Remarks ,
3353     name-pl = remarks ,
3354
3355 type = example ,
3356     Name-sg = Example ,
3357     name-sg = example ,
3358     Name-pl = Examples ,

```

```

3359   name-pl = examples ,
3360
3361   type = algorithm ,
3362     Name-sg = Algorithm ,
3363     name-sg = algorithm ,
3364     Name-pl = Algorithms ,
3365     name-pl = algorithms ,
3366
3367   type = listing ,
3368     Name-sg = Listing ,
3369     name-sg = listing ,
3370     Name-pl = Listings ,
3371     name-pl = listings ,
3372
3373   type = exercise ,
3374     Name-sg = Exercise ,
3375     name-sg = exercise ,
3376     Name-pl = Exercises ,
3377     name-pl = exercises ,
3378
3379   type = solution ,
3380     Name-sg = Solution ,
3381     name-sg = solution ,
3382     Name-pl = Solutions ,
3383     name-pl = solutions ,
3384 </dict-english>

```

## 10.2 German

```

3385 <package>\zcDeclareLanguage [ N , A , D , G ] { german }
3386 <package>\zcDeclareLanguageAlias { austrian      } { german }
3387 <package>\zcDeclareLanguageAlias { germanb       } { german }
3388 <package>\zcDeclareLanguageAlias { ngerman       } { german }
3389 <package>\zcDeclareLanguageAlias { naustrian     } { german }
3390 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
3391 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
3392 <*dict-german>
3393 namesep = {\nobreakspace} ,
3394 pairsep = {\~und\nobreakspace} ,
3395 listsep = {\~,~} ,
3396 lastsep = {\~und\nobreakspace} ,
3397 tpairsep = {\~und\nobreakspace} ,
3398 tlistsep = {\~,~} ,
3399 tlastsep = {\~und\nobreakspace} ,
3400 notesep = {\~} ,
3401 rangesep = {\~bis\nobreakspace} ,
3402
3403 type = part ,
3404   case = N ,
3405     Name-sg = Teil ,
3406     name-sg = Teil ,
3407     Name-pl = Teile ,
3408     name-pl = Teile ,
3409   case = A ,

```



```

3410     Name-sg = Teil ,
3411     name-sg = Teil ,
3412     Name-pl = Teile ,
3413     name-pl = Teile ,
3414     case = D ,
3415     Name-sg = Teil ,
3416     name-sg = Teil ,
3417     Name-pl = Teilen ,
3418     name-pl = Teilen ,
3419     case = G ,
3420     Name-sg = Teiles ,
3421     name-sg = Teiles ,
3422     Name-pl = Teile ,
3423     name-pl = Teile ,
3424
3425 type = chapter ,
3426     case = N ,
3427     Name-sg = Kapitel ,
3428     name-sg = Kapitel ,
3429     Name-pl = Kapitel ,
3430     name-pl = Kapitel ,
3431     case = A ,
3432     Name-sg = Kapitel ,
3433     name-sg = Kapitel ,
3434     Name-pl = Kapitel ,
3435     name-pl = Kapitel ,
3436     case = D ,
3437     Name-sg = Kapitel ,
3438     name-sg = Kapitel ,
3439     Name-pl = Kapiteln ,
3440     name-pl = Kapiteln ,
3441     case = G ,
3442     Name-sg = Kapiteln ,
3443     name-sg = Kapiteln ,
3444     Name-pl = Kapitel ,
3445     name-pl = Kapitel ,
3446
3447 type = section ,
3448     case = N ,
3449     Name-sg = Abschnitt ,
3450     name-sg = Abschnitt ,
3451     Name-pl = Abschnitte ,
3452     name-pl = Abschnitte ,
3453     case = A ,
3454     Name-sg = Abschnitt ,
3455     name-sg = Abschnitt ,
3456     Name-pl = Abschnitte ,
3457     name-pl = Abschnitte ,
3458     case = D ,
3459     Name-sg = Abschnitt ,
3460     name-sg = Abschnitt ,
3461     Name-pl = Abschnitten ,
3462     name-pl = Abschnitten ,
3463     case = G ,

```

```

3464     Name-sg = Abschnitts ,
3465     name-sg = Abschnitts ,
3466     Name-pl = Abschnitte ,
3467     name-pl = Abschnitte ,
3468
3469 type = paragraph ,
3470     case = N ,
3471     Name-sg = Absatz ,
3472     name-sg = Absatz ,
3473     Name-pl = Absätze ,
3474     name-pl = Absätze ,
3475     case = A ,
3476     Name-sg = Absatz ,
3477     name-sg = Absatz ,
3478     Name-pl = Absätze ,
3479     name-pl = Absätze ,
3480     case = D ,
3481     Name-sg = Absatz ,
3482     name-sg = Absatz ,
3483     Name-pl = Absätzen ,
3484     name-pl = Absätzen ,
3485     case = G ,
3486     Name-sg = Absatzes ,
3487     name-sg = Absatzes ,
3488     Name-pl = Absätze ,
3489     name-pl = Absätze ,
3490
3491 type = appendix ,
3492     case = N ,
3493     Name-sg = Anhang ,
3494     name-sg = Anhang ,
3495     Name-pl = Anhänge ,
3496     name-pl = Anhänge ,
3497     case = A ,
3498     Name-sg = Anhang ,
3499     name-sg = Anhang ,
3500     Name-pl = Anhänge ,
3501     name-pl = Anhänge ,
3502     case = D ,
3503     Name-sg = Anhang ,
3504     name-sg = Anhang ,
3505     Name-pl = Anhängen ,
3506     name-pl = Anhängen ,
3507     case = G ,
3508     Name-sg = Anhangs ,
3509     name-sg = Anhangs ,
3510     Name-pl = Anhänge ,
3511     name-pl = Anhänge ,
3512
3513 type = subappendix ,
3514     case = N ,
3515     Name-sg = Anhang ,
3516     name-sg = Anhang ,
3517     Name-pl = Anhänge ,

```

```

3518     name-pl = Anhänge ,
3519 case = A ,
3520     Name-sg = Anhang ,
3521     name-sg = Anhang ,
3522     Name-pl = Anhänge ,
3523     name-pl = Anhänge ,
3524 case = D ,
3525     Name-sg = Anhang ,
3526     name-sg = Anhang ,
3527     Name-pl = Anhängen ,
3528     name-pl = Anhängen ,
3529 case = G ,
3530     Name-sg = Anhangs ,
3531     name-sg = Anhangs ,
3532     Name-pl = Anhänge ,
3533     name-pl = Anhänge ,
3534
3535 type = page ,
3536 case = N ,
3537     Name-sg = Seite ,
3538     name-sg = Seite ,
3539     Name-pl = Seiten ,
3540     name-pl = Seiten ,
3541 case = A ,
3542     Name-sg = Seite ,
3543     name-sg = Seite ,
3544     Name-pl = Seiten ,
3545     name-pl = Seiten ,
3546 case = D ,
3547     Name-sg = Seite ,
3548     name-sg = Seite ,
3549     Name-pl = Seiten ,
3550     name-pl = Seiten ,
3551 case = G ,
3552     Name-sg = Seite ,
3553     name-sg = Seite ,
3554     Name-pl = Seiten ,
3555     name-pl = Seiten ,
3556
3557 type = line ,
3558 case = N ,
3559     Name-sg = Zeile ,
3560     name-sg = Zeile ,
3561     Name-pl = Zeilen ,
3562     name-pl = Zeilen ,
3563 case = A ,
3564     Name-sg = Zeile ,
3565     name-sg = Zeile ,
3566     Name-pl = Zeilen ,
3567     name-pl = Zeilen ,
3568 case = D ,
3569     Name-sg = Zeile ,
3570     name-sg = Zeile ,
3571     Name-pl = Zeilen ,

```

```

3572     name-pl = Zeilen ,
3573     case = G ,
3574     Name-sg = Zeile ,
3575     name-sg = Zeile ,
3576     Name-pl = Zeilen ,
3577     name-pl = Zeilen ,
3578
3579 type = figure ,
3580     case = N ,
3581     Name-sg = Abbildung ,
3582     name-sg = Abbildung ,
3583     Name-pl = Abbildungen ,
3584     name-pl = Abbildungen ,
3585     Name-sg-ab = Abb. ,
3586     name-sg-ab = Abb. ,
3587     Name-pl-ab = Abb. ,
3588     name-pl-ab = Abb. ,
3589     case = A ,
3590     Name-sg = Abbildung ,
3591     name-sg = Abbildung ,
3592     Name-pl = Abbildungen ,
3593     name-pl = Abbildungen ,
3594     Name-sg-ab = Abb. ,
3595     name-sg-ab = Abb. ,
3596     Name-pl-ab = Abb. ,
3597     name-pl-ab = Abb. ,
3598     case = D ,
3599     Name-sg = Abbildung ,
3600     name-sg = Abbildung ,
3601     Name-pl = Abbildungen ,
3602     name-pl = Abbildungen ,
3603     Name-sg-ab = Abb. ,
3604     name-sg-ab = Abb. ,
3605     Name-pl-ab = Abb. ,
3606     name-pl-ab = Abb. ,
3607     case = G ,
3608     Name-sg = Abbildung ,
3609     name-sg = Abbildung ,
3610     Name-pl = Abbildungen ,
3611     name-pl = Abbildungen ,
3612     Name-sg-ab = Abb. ,
3613     name-sg-ab = Abb. ,
3614     Name-pl-ab = Abb. ,
3615     name-pl-ab = Abb. ,
3616
3617 type = table ,
3618     case = N ,
3619     Name-sg = Tabelle ,
3620     name-sg = Tabelle ,
3621     Name-pl = Tabellen ,
3622     name-pl = Tabellen ,
3623     case = A ,
3624     Name-sg = Tabelle ,
3625     name-sg = Tabelle ,

```

```

3626     Name-pl = Tabellen ,
3627     name-pl = Tabellen ,
3628     case = D ,
3629     Name-sg = Tabelle ,
3630     name-sg = Tabelle ,
3631     Name-pl = Tabellen ,
3632     name-pl = Tabellen ,
3633     case = G ,
3634     Name-sg = Tabelle ,
3635     name-sg = Tabelle ,
3636     Name-pl = Tabellen ,
3637     name-pl = Tabellen ,
3638
3639 type = item ,
3640     case = N ,
3641     Name-sg = Punkt ,
3642     name-sg = Punkt ,
3643     Name-pl = Punkte ,
3644     name-pl = Punkte ,
3645     case = A ,
3646     Name-sg = Punkt ,
3647     name-sg = Punkt ,
3648     Name-pl = Punkte ,
3649     name-pl = Punkte ,
3650     case = D ,
3651     Name-sg = Punkt ,
3652     name-sg = Punkt ,
3653     Name-pl = Punkten ,
3654     name-pl = Punkten ,
3655     case = G ,
3656     Name-sg = Punktes ,
3657     name-sg = Punktes ,
3658     Name-pl = Punkte ,
3659     name-pl = Punkte ,
3660
3661 type = footnote ,
3662     case = N ,
3663     Name-sg = Fußnote ,
3664     name-sg = Fußnote ,
3665     Name-pl = Fußnoten ,
3666     name-pl = Fußnoten ,
3667     case = A ,
3668     Name-sg = Fußnote ,
3669     name-sg = Fußnote ,
3670     Name-pl = Fußnoten ,
3671     name-pl = Fußnoten ,
3672     case = D ,
3673     Name-sg = Fußnote ,
3674     name-sg = Fußnote ,
3675     Name-pl = Fußnoten ,
3676     name-pl = Fußnoten ,
3677     case = G ,
3678     Name-sg = Fußnote ,
3679     name-sg = Fußnote ,

```

```

3680     Name-pl = Fußnoten ,
3681     name-pl = Fußnoten ,
3682
3683 type = note ,
3684     case = N ,
3685     Name-sg = Anmerkung ,
3686     name-sg = Anmerkung ,
3687     Name-pl = Anmerkungen ,
3688     name-pl = Anmerkungen ,
3689     case = A ,
3690     Name-sg = Anmerkung ,
3691     name-sg = Anmerkung ,
3692     Name-pl = Anmerkungen ,
3693     name-pl = Anmerkungen ,
3694     case = D ,
3695     Name-sg = Anmerkung ,
3696     name-sg = Anmerkung ,
3697     Name-pl = Anmerkungen ,
3698     name-pl = Anmerkungen ,
3699     case = G ,
3700     Name-sg = Anmerkung ,
3701     name-sg = Anmerkung ,
3702     Name-pl = Anmerkungen ,
3703     name-pl = Anmerkungen ,
3704
3705 type = equation ,
3706     case = N ,
3707     Name-sg = Gleichung ,
3708     name-sg = Gleichung ,
3709     Name-pl = Gleichungen ,
3710     name-pl = Gleichungen ,
3711     case = A ,
3712     Name-sg = Gleichung ,
3713     name-sg = Gleichung ,
3714     Name-pl = Gleichungen ,
3715     name-pl = Gleichungen ,
3716     case = D ,
3717     Name-sg = Gleichung ,
3718     name-sg = Gleichung ,
3719     Name-pl = Gleichungen ,
3720     name-pl = Gleichungen ,
3721     case = G ,
3722     Name-sg = Gleichung ,
3723     name-sg = Gleichung ,
3724     Name-pl = Gleichungen ,
3725     name-pl = Gleichungen ,
3726     refpre-in = {()} ,
3727     refpos-in = {} ,
3728
3729 type = theorem ,
3730     case = N ,
3731     Name-sg = Theorem ,
3732     name-sg = Theorem ,
3733     Name-pl = Theoreme ,

```

```

3734     name-pl = Theoreme ,
3735 case = A ,
3736     Name-sg = Theorem ,
3737     name-sg = Theorem ,
3738     Name-pl = Theoreme ,
3739     name-pl = Theoreme ,
3740 case = D ,
3741     Name-sg = Theorem ,
3742     name-sg = Theorem ,
3743     Name-pl = Theoremen ,
3744     name-pl = Theoremen ,
3745 case = G ,
3746     Name-sg = Theorems ,
3747     name-sg = Theorems ,
3748     Name-pl = Theoreme ,
3749     name-pl = Theoreme ,
3750
3751 type = lemma ,
3752 case = N ,
3753     Name-sg = Lemma ,
3754     name-sg = Lemma ,
3755     Name-pl = Lemmata ,
3756     name-pl = Lemmata ,
3757 case = A ,
3758     Name-sg = Lemma ,
3759     name-sg = Lemma ,
3760     Name-pl = Lemmata ,
3761     name-pl = Lemmata ,
3762 case = D ,
3763     Name-sg = Lemma ,
3764     name-sg = Lemma ,
3765     Name-pl = Lemmata ,
3766     name-pl = Lemmata ,
3767 case = G ,
3768     Name-sg = Lemmas ,
3769     name-sg = Lemmas ,
3770     Name-pl = Lemmata ,
3771     name-pl = Lemmata ,
3772
3773 type = corollary ,
3774 case = N ,
3775     Name-sg = Korollar ,
3776     name-sg = Korollar ,
3777     Name-pl = Korollare ,
3778     name-pl = Korollare ,
3779 case = A ,
3780     Name-sg = Korollar ,
3781     name-sg = Korollar ,
3782     Name-pl = Korollare ,
3783     name-pl = Korollare ,
3784 case = D ,
3785     Name-sg = Korollar ,
3786     name-sg = Korollar ,
3787     Name-pl = Korollaren ,

```

```

3788     name-pl = Korollaren ,
3789     case = G ,
3790     Name-sg = Korollars ,
3791     name-sg = Korollars ,
3792     Name-pl = Korollare ,
3793     name-pl = Korollare ,
3794
3795 type = proposition ,
3796     case = N ,
3797     Name-sg = Satz ,
3798     name-sg = Satz ,
3799     Name-pl = Sätze ,
3800     name-pl = Sätze ,
3801     case = A ,
3802     Name-sg = Satz ,
3803     name-sg = Satz ,
3804     Name-pl = Sätze ,
3805     name-pl = Sätze ,
3806     case = D ,
3807     Name-sg = Satz ,
3808     name-sg = Satz ,
3809     Name-pl = Sätzen ,
3810     name-pl = Sätzen ,
3811     case = G ,
3812     Name-sg = Satzes ,
3813     name-sg = Satzes ,
3814     Name-pl = Sätze ,
3815     name-pl = Sätze ,
3816
3817 type = definition ,
3818     case = N ,
3819     Name-sg = Definition ,
3820     name-sg = Definition ,
3821     Name-pl = Definitionen ,
3822     name-pl = Definitionen ,
3823     case = A ,
3824     Name-sg = Definition ,
3825     name-sg = Definition ,
3826     Name-pl = Definitionen ,
3827     name-pl = Definitionen ,
3828     case = D ,
3829     Name-sg = Definition ,
3830     name-sg = Definition ,
3831     Name-pl = Definitionen ,
3832     name-pl = Definitionen ,
3833     case = G ,
3834     Name-sg = Definition ,
3835     name-sg = Definition ,
3836     Name-pl = Definitionen ,
3837     name-pl = Definitionen ,
3838
3839 type = proof ,
3840     case = N ,
3841     Name-sg = Beweis ,

```



```

3842     name-sg = Beweis ,
3843     Name-pl = Beweise ,
3844     name-pl = Beweise ,
3845     case = A ,
3846     Name-sg = Beweis ,
3847     name-sg = Beweis ,
3848     Name-pl = Beweise ,
3849     name-pl = Beweise ,
3850     case = D ,
3851     Name-sg = Beweis ,
3852     name-sg = Beweis ,
3853     Name-pl = Beweisen ,
3854     name-pl = Beweisen ,
3855     case = G ,
3856     Name-sg = Beweises ,
3857     name-sg = Beweises ,
3858     Name-pl = Beweise ,
3859     name-pl = Beweise ,
3860
3861 type = result ,
3862     case = N ,
3863     Name-sg = Ergebnis ,
3864     name-sg = Ergebnis ,
3865     Name-pl = Ergebnisse ,
3866     name-pl = Ergebnisse ,
3867     case = A ,
3868     Name-sg = Ergebnis ,
3869     name-sg = Ergebnis ,
3870     Name-pl = Ergebnisse ,
3871     name-pl = Ergebnisse ,
3872     case = D ,
3873     Name-sg = Ergebnis ,
3874     name-sg = Ergebnis ,
3875     Name-pl = Ergebnissen ,
3876     name-pl = Ergebnissen ,
3877     case = G ,
3878     Name-sg = Ergebnisses ,
3879     name-sg = Ergebnisses ,
3880     Name-pl = Ergebnisse ,
3881     name-pl = Ergebnisse ,
3882
3883 type = remark ,
3884     case = N ,
3885     Name-sg = Bemerkung ,
3886     name-sg = Bemerkung ,
3887     Name-pl = Bemerkungen ,
3888     name-pl = Bemerkungen ,
3889     case = A ,
3890     Name-sg = Bemerkung ,
3891     name-sg = Bemerkung ,
3892     Name-pl = Bemerkungen ,
3893     name-pl = Bemerkungen ,
3894     case = D ,
3895     Name-sg = Bemerkung ,

```

```

3896     name-sg = Bemerkung ,
3897     Name-pl = Bemerkungen ,
3898     name-pl = Bemerkungen ,
3899     case = G ,
3900     Name-sg = Bemerkung ,
3901     name-sg = Bemerkung ,
3902     Name-pl = Bemerkungen ,
3903     name-pl = Bemerkungen ,
3904
3905 type = example ,
3906     case = N ,
3907     Name-sg = Beispiel ,
3908     name-sg = Beispiel ,
3909     Name-pl = Beispiele ,
3910     name-pl = Beispiele ,
3911     case = A ,
3912     Name-sg = Beispiel ,
3913     name-sg = Beispiel ,
3914     Name-pl = Beispiele ,
3915     name-pl = Beispiele ,
3916     case = D ,
3917     Name-sg = Beispiel ,
3918     name-sg = Beispiel ,
3919     Name-pl = Beispielen ,
3920     name-pl = Beispielen ,
3921     case = G ,
3922     Name-sg = Beispiels ,
3923     name-sg = Beispiels ,
3924     Name-pl = Beispiele ,
3925     name-pl = Beispiele ,
3926
3927 type = algorithm ,
3928     case = N ,
3929     Name-sg = Algorithmus ,
3930     name-sg = Algorithmus ,
3931     Name-pl = Algorithmen ,
3932     name-pl = Algorithmen ,
3933     case = A ,
3934     Name-sg = Algorithmus ,
3935     name-sg = Algorithmus ,
3936     Name-pl = Algorithmen ,
3937     name-pl = Algorithmen ,
3938     case = D ,
3939     Name-sg = Algorithmus ,
3940     name-sg = Algorithmus ,
3941     Name-pl = Algorithmen ,
3942     name-pl = Algorithmen ,
3943     case = G ,
3944     Name-sg = Algorithmus ,
3945     name-sg = Algorithmus ,
3946     Name-pl = Algorithmen ,
3947     name-pl = Algorithmen ,
3948
3949 type = listing ,

```

```

3950 case = N ,
3951     Name-sg = Listing ,
3952     name-sg = Listing ,
3953     Name-pl = Listings ,
3954     name-pl = Listings ,
3955 case = A ,
3956     Name-sg = Listing ,
3957     name-sg = Listing ,
3958     Name-pl = Listings ,
3959     name-pl = Listings ,
3960 case = D ,
3961     Name-sg = Listing ,
3962     name-sg = Listing ,
3963     Name-pl = Listings ,
3964     name-pl = Listings ,
3965 case = G ,
3966     Name-sg = Listings ,
3967     name-sg = Listings ,
3968     Name-pl = Listings ,
3969     name-pl = Listings ,
3970
3971 type = exercise ,
3972     case = N ,
3973         Name-sg = Übungsaufgabe ,
3974         name-sg = Übungsaufgabe ,
3975         Name-pl = Übungsaufgaben ,
3976         name-pl = Übungsaufgaben ,
3977     case = A ,
3978         Name-sg = Übungsaufgabe ,
3979         name-sg = Übungsaufgabe ,
3980         Name-pl = Übungsaufgaben ,
3981         name-pl = Übungsaufgaben ,
3982     case = D ,
3983         Name-sg = Übungsaufgabe ,
3984         name-sg = Übungsaufgabe ,
3985         Name-pl = Übungsaufgaben ,
3986         name-pl = Übungsaufgaben ,
3987     case = G ,
3988         Name-sg = Übungsaufgabe ,
3989         name-sg = Übungsaufgabe ,
3990         Name-pl = Übungsaufgaben ,
3991         name-pl = Übungsaufgaben ,
3992
3993 type = solution ,
3994     case = N ,
3995         Name-sg = Lösung ,
3996         name-sg = Lösung ,
3997         Name-pl = Lösungen ,
3998         name-pl = Lösungen ,
3999     case = A ,
4000         Name-sg = Lösung ,
4001         name-sg = Lösung ,
4002         Name-pl = Lösungen ,
4003         name-pl = Lösungen ,

```

```

4004 case = D ,
4005     Name-sg = Lösung ,
4006     name-sg = Lösung ,
4007     Name-pl = Lösungen ,
4008     name-pl = Lösungen ,
4009 case = G ,
4010     Name-sg = Lösung ,
4011     name-sg = Lösung ,
4012     Name-pl = Lösungen ,
4013     name-pl = Lösungen ,
4014 </dict-german>

```

### 10.3 French

```

4015 <package>\zcDeclareLanguage { french }
4016 <package>\zcDeclareLanguageAlias { acadian } { french }
4017 <package>\zcDeclareLanguageAlias { canadien } { french }
4018 <package>\zcDeclareLanguageAlias { francais } { french }
4019 <package>\zcDeclareLanguageAlias { frenchb } { french }
4020 <*dict-french>

4021 namesep = {\nobreakspace} ,
4022 pairsep = {\~et\nobreakspace} ,
4023 listsep = {\~,~} ,
4024 lastsep = {\~et\nobreakspace} ,
4025 tpairsep = {\~et\nobreakspace} ,
4026 tlistsep = {\~,~} ,
4027 tlastsep = {\~et\nobreakspace} ,
4028 notesep = {\~} ,
4029 rangesep = {\~à\nobreakspace} ,
4030
4031 type = part ,
4032     Name-sg = Partie ,
4033     name-sg = partie ,
4034     Name-pl = Parties ,
4035     name-pl = parties ,
4036
4037 type = chapter ,
4038     Name-sg = Chapitre ,
4039     name-sg = chapitre ,
4040     Name-pl = Chapitres ,
4041     name-pl = chapitres ,
4042
4043 type = section ,
4044     Name-sg = Section ,
4045     name-sg = section ,
4046     Name-pl = Sections ,
4047     name-pl = sections ,
4048
4049 type = paragraph ,
4050     Name-sg = Paragraphe ,
4051     name-sg = paragraphe ,
4052     Name-pl = Paragraphes ,
4053     name-pl = paragraphes ,
4054

```

```

4055 type = appendix ,
4056     Name-sg = Annexe ,
4057     name-sg = annexe ,
4058     Name-pl = Annexes ,
4059     name-pl = annexes ,
4060
4061 type = subappendix ,
4062     Name-sg = Annexe ,
4063     name-sg = annexe ,
4064     Name-pl = Annexes ,
4065     name-pl = annexes ,
4066
4067 type = page ,
4068     Name-sg = Page ,
4069     name-sg = page ,
4070     Name-pl = Pages ,
4071     name-pl = pages ,
4072
4073 type = line ,
4074     Name-sg = Ligne ,
4075     name-sg = ligne ,
4076     Name-pl = Lignes ,
4077     name-pl = lignes ,
4078
4079 type = figure ,
4080     Name-sg = Figure ,
4081     name-sg = figure ,
4082     Name-pl = Figures ,
4083     name-pl = figures ,
4084
4085 type = table ,
4086     Name-sg = Table ,
4087     name-sg = table ,
4088     Name-pl = Tables ,
4089     name-pl = tables ,
4090
4091 type = item ,
4092     Name-sg = Point ,
4093     name-sg = point ,
4094     Name-pl = Points ,
4095     name-pl = points ,
4096
4097 type = footnote ,
4098     Name-sg = Note ,
4099     name-sg = note ,
4100     Name-pl = Notes ,
4101     name-pl = notes ,
4102
4103 type = note ,
4104     Name-sg = Note ,
4105     name-sg = note ,
4106     Name-pl = Notes ,
4107     name-pl = notes ,
4108

```

```

4109 type = equation ,
4110     Name-sg = Équation ,
4111     name-sg = équation ,
4112     Name-pl = Équations ,
4113     name-pl = équations ,
4114     refpre-in = {()} ,
4115     refpos-in = {} } ,
4116
4117 type = theorem ,
4118     Name-sg = Théorème ,
4119     name-sg = théorème ,
4120     Name-pl = Théorèmes ,
4121     name-pl = théorèmes ,
4122
4123 type = lemma ,
4124     Name-sg = Lemme ,
4125     name-sg = lemme ,
4126     Name-pl = Lemmes ,
4127     name-pl = lemmes ,
4128
4129 type = corollary ,
4130     Name-sg = Corollaire ,
4131     name-sg = corollaire ,
4132     Name-pl = Corollaires ,
4133     name-pl = corollaires ,
4134
4135 type = proposition ,
4136     Name-sg = Proposition ,
4137     name-sg = proposition ,
4138     Name-pl = Propositions ,
4139     name-pl = propositions ,
4140
4141 type = definition ,
4142     Name-sg = Définition ,
4143     name-sg = définition ,
4144     Name-pl = Définitions ,
4145     name-pl = définitions ,
4146
4147 type = proof ,
4148     Name-sg = Démonstration ,
4149     name-sg = démonstration ,
4150     Name-pl = Démonstrations ,
4151     name-pl = démonstrations ,
4152
4153 type = result ,
4154     Name-sg = Résultat ,
4155     name-sg = résultat ,
4156     Name-pl = Résultats ,
4157     name-pl = résultats ,
4158
4159 type = remark ,
4160     Name-sg = Remarque ,
4161     name-sg = remarque ,
4162     Name-pl = Remarques ,

```

```

4163   name-pl = remarques ,
4164
4165   type = example ,
4166     Name-sg = Exemple ,
4167     name-sg = exemple ,
4168     Name-pl = Exemples ,
4169     name-pl = exemples ,
4170
4171   type = algorithm ,
4172     Name-sg = Algorithme ,
4173     name-sg = algorithme ,
4174     Name-pl = Algorithmes ,
4175     name-pl = algorithmes ,
4176
4177   type = listing ,
4178     Name-sg = Liste ,
4179     name-sg = liste ,
4180     Name-pl = Listes ,
4181     name-pl = listes ,
4182
4183   type = exercise ,
4184     Name-sg = Exercice ,
4185     name-sg = exercice ,
4186     Name-pl = Exercices ,
4187     name-pl = exercices ,
4188
4189   type = solution ,
4190     Name-sg = Solution ,
4191     name-sg = solution ,
4192     Name-pl = Solutions ,
4193     name-pl = solutions ,
4194 </dict-french>

```

## 10.4 Portuguese

```

4195 <package>\zcDeclareLanguage { portuguese }
4196 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
4197 <package>\zcDeclareLanguageAlias { brazil   } { portuguese }
4198 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
4199 <*dict-portuguese>

4200 namesep = {\nobreakspace} ,
4201 pairsep  = {\~e\nobreakspace} ,
4202 listsep  = {,~} ,
4203 lastsep  = {\~e\nobreakspace} ,
4204 tpairsep = {\~e\nobreakspace} ,
4205 tlistsep = {,~} ,
4206 tlastsep = {\~e\nobreakspace} ,
4207 notesep  = {\~} ,
4208 rangesep = {\~a\nobreakspace} ,
4209
4210 type = part ,
4211     Name-sg = Parte ,
4212     name-sg = parte ,
4213     Name-pl = Partes ,

```

```

4214     name-pl = partes ,
4215
4216 type = chapter ,
4217     Name-sg = Capítulo ,
4218     name-sg = capítulo ,
4219     Name-pl = Capítulos ,
4220     name-pl = capítulos ,
4221
4222 type = section ,
4223     Name-sg = Seção ,
4224     name-sg = seção ,
4225     Name-pl = Seções ,
4226     name-pl = seções ,
4227
4228 type = paragraph ,
4229     Name-sg = Parágrafo ,
4230     name-sg = parágrafo ,
4231     Name-pl = Parágrafos ,
4232     name-pl = parágrafos ,
4233     Name-sg-ab = Par. ,
4234     name-sg-ab = par. ,
4235     Name-pl-ab = Par. ,
4236     name-pl-ab = par. ,
4237
4238 type = appendix ,
4239     Name-sg = Apêndice ,
4240     name-sg = apêndice ,
4241     Name-pl = Apêndices ,
4242     name-pl = apêndices ,
4243
4244 type = subappendix ,
4245     Name-sg = Apêndice ,
4246     name-sg = apêndice ,
4247     Name-pl = Apêndices ,
4248     name-pl = apêndices ,
4249
4250 type = page ,
4251     Name-sg = Página ,
4252     name-sg = página ,
4253     Name-pl = Páginas ,
4254     name-pl = páginas ,
4255     name-sg-ab = p. ,
4256     name-pl-ab = pp. ,
4257
4258 type = line ,
4259     Name-sg = Linha ,
4260     name-sg = linha ,
4261     Name-pl = Linhas ,
4262     name-pl = linhas ,
4263
4264 type = figure ,
4265     Name-sg = Figura ,
4266     name-sg = figura ,
4267     Name-pl = Figuras ,

```



```

4268     name-pl = figuras ,
4269     Name-sg-ab = Fig. ,
4270     name-sg-ab = fig. ,
4271     Name-pl-ab = Figs. ,
4272     name-pl-ab = figs. ,
4273
4274     type = table ,
4275     Name-sg = Tabela ,
4276     name-sg = tabela ,
4277     Name-pl = Tabelas ,
4278     name-pl = tabelas ,
4279
4280     type = item ,
4281     Name-sg = Item ,
4282     name-sg = item ,
4283     Name-pl = Itens ,
4284     name-pl = itens ,
4285
4286     type = footnote ,
4287     Name-sg = Nota ,
4288     name-sg = nota ,
4289     Name-pl = Notas ,
4290     name-pl = notas ,
4291
4292     type = note ,
4293     Name-sg = Nota ,
4294     name-sg = nota ,
4295     Name-pl = Notas ,
4296     name-pl = notas ,
4297
4298     type = equation ,
4299     Name-sg = Equação ,
4300     name-sg = equação ,
4301     Name-pl = Equações ,
4302     name-pl = equações ,
4303     Name-sg-ab = Eq. ,
4304     name-sg-ab = eq. ,
4305     Name-pl-ab = Eqs. ,
4306     name-pl-ab = eqs. ,
4307     refpre-in = { ( } ,
4308     refpos-in = { ) } ,
4309
4310     type = theorem ,
4311     Name-sg = Teorema ,
4312     name-sg = teorema ,
4313     Name-pl = Teoremas ,
4314     name-pl = teoremas ,
4315
4316     type = lemma ,
4317     Name-sg = Lema ,
4318     name-sg = lema ,
4319     Name-pl = Lemas ,
4320     name-pl = lemas ,
4321

```

```

4322 type = corollary ,
4323     Name-sg = Corolário ,
4324     name-sg = corolário ,
4325     Name-pl = Corolários ,
4326     name-pl = corolários ,
4327
4328 type = proposition ,
4329     Name-sg = Proposição ,
4330     name-sg = proposição ,
4331     Name-pl = Proposições ,
4332     name-pl = proposições ,
4333
4334 type = definition ,
4335     Name-sg = Definição ,
4336     name-sg = definição ,
4337     Name-pl = Definições ,
4338     name-pl = definições ,
4339
4340 type = proof ,
4341     Name-sg = Demonstração ,
4342     name-sg = demonstração ,
4343     Name-pl = Demonstrações ,
4344     name-pl = demonstrações ,
4345
4346 type = result ,
4347     Name-sg = Resultado ,
4348     name-sg = resultado ,
4349     Name-pl = Resultados ,
4350     name-pl = resultados ,
4351
4352 type = remark ,
4353     Name-sg = Observação ,
4354     name-sg = observação ,
4355     Name-pl = Observações ,
4356     name-pl = observações ,
4357
4358 type = example ,
4359     Name-sg = Exemplo ,
4360     name-sg = exemplo ,
4361     Name-pl = Exemplos ,
4362     name-pl = exemplos ,
4363
4364 type = algorithm ,
4365     Name-sg = Algoritmo ,
4366     name-sg = algoritmo ,
4367     Name-pl = Algoritmos ,
4368     name-pl = algoritmos ,
4369
4370 type = listing ,
4371     Name-sg = Listagem ,
4372     name-sg = listagem ,
4373     Name-pl = Listagens ,
4374     name-pl = listagens ,
4375

```

```

4376 type = exercise ,
4377   Name-sg = Exercício ,
4378   name-sg = exercício ,
4379   Name-pl = Exercícios ,
4380   name-pl = exercícios ,
4381
4382 type = solution ,
4383   Name-sg = Solução ,
4384   name-sg = solução ,
4385   Name-pl = Soluções ,
4386   name-pl = soluções ,
4387 </dict-portuguese>

```

## 10.5 Spanish

```

4388 <package>\zcDeclareLanguage { spanish }
4389 <*dict-spanish>
4390 namesep = {\nobreakspace} ,
4391 pairsep = {\~y\nobreakspace} ,
4392 listsep = {,~} ,
4393 lastsep = {\~y\nobreakspace} ,
4394 tpairsep = {\~y\nobreakspace} ,
4395 tlistsep = {,~} ,
4396 tlastsep = {\~y\nobreakspace} ,
4397 notesep = {\~} ,
4398 rangesep = {\~a\nobreakspace} ,
4399
4400 type = part ,
4401   Name-sg = Parte ,
4402   name-sg = parte ,
4403   Name-pl = Partes ,
4404   name-pl = partes ,
4405
4406 type = chapter ,
4407   Name-sg = Capítulo ,
4408   name-sg = capítulo ,
4409   Name-pl = Capítulos ,
4410   name-pl = capítulos ,
4411
4412 type = section ,
4413   Name-sg = Sección ,
4414   name-sg = sección ,
4415   Name-pl = Secciones ,
4416   name-pl = secciones ,
4417
4418 type = paragraph ,
4419   Name-sg = Párrafo ,
4420   name-sg = párrafo ,
4421   Name-pl = Párrafos ,
4422   name-pl = párrafos ,
4423
4424 type = appendix ,
4425   Name-sg = Apéndice ,
4426   name-sg = apéndice ,

```

```

4427     Name-pl = Apéndices ,
4428     name-pl = apéndices ,
4429
4430     type = subappendix ,
4431     Name-sg = Apéndice ,
4432     name-sg = apéndice ,
4433     Name-pl = Apéndices ,
4434     name-pl = apéndices ,
4435
4436     type = page ,
4437     Name-sg = Página ,
4438     name-sg = página ,
4439     Name-pl = Páginas ,
4440     name-pl = páginas ,
4441
4442     type = line ,
4443     Name-sg = Línea ,
4444     name-sg = línea ,
4445     Name-pl = Líneas ,
4446     name-pl = líneas ,
4447
4448     type = figure ,
4449     Name-sg = Figura ,
4450     name-sg = figura ,
4451     Name-pl = Figuras ,
4452     name-pl = figuras ,
4453
4454     type = table ,
4455     Name-sg = Cuadro ,
4456     name-sg = cuadro ,
4457     Name-pl = Cuadros ,
4458     name-pl = cuadros ,
4459
4460     type = item ,
4461     Name-sg = Punto ,
4462     name-sg = punto ,
4463     Name-pl = Puntos ,
4464     name-pl = puntos ,
4465
4466     type = footnote ,
4467     Name-sg = Nota ,
4468     name-sg = nota ,
4469     Name-pl = Notas ,
4470     name-pl = notas ,
4471
4472     type = note ,
4473     Name-sg = Nota ,
4474     name-sg = nota ,
4475     Name-pl = Notas ,
4476     name-pl = notas ,
4477
4478     type = equation ,
4479     Name-sg = Ecuación ,
4480     name-sg = ecuación ,

```

```

4481   Name-pl = Ecuaciones ,
4482   name-pl = ecuaciones ,
4483   refpre-in = {()} ,
4484   refpos-in = {} } ,
4485
4486   type = theorem ,
4487   Name-sg = Teorema ,
4488   name-sg = teorema ,
4489   Name-pl = Teoremas ,
4490   name-pl = teoremas ,
4491
4492   type = lemma ,
4493   Name-sg = Lema ,
4494   name-sg = lema ,
4495   Name-pl = Lemas ,
4496   name-pl = lemas ,
4497
4498   type = corollary ,
4499   Name-sg = Corolario ,
4500   name-sg = corolario ,
4501   Name-pl = Corolarios ,
4502   name-pl = corolarios ,
4503
4504   type = proposition ,
4505   Name-sg = Proposición ,
4506   name-sg = proposición ,
4507   Name-pl = Proposiciones ,
4508   name-pl = proposiciones ,
4509
4510   type = definition ,
4511   Name-sg = Definición ,
4512   name-sg = definición ,
4513   Name-pl = Definiciones ,
4514   name-pl = definiciones ,
4515
4516   type = proof ,
4517   Name-sg = Demostración ,
4518   name-sg = demostración ,
4519   Name-pl = Demostraciones ,
4520   name-pl = demostraciones ,
4521
4522   type = result ,
4523   Name-sg = Resultado ,
4524   name-sg = resultado ,
4525   Name-pl = Resultados ,
4526   name-pl = resultados ,
4527
4528   type = remark ,
4529   Name-sg = Observación ,
4530   name-sg = observación ,
4531   Name-pl = Observaciones ,
4532   name-pl = observaciones ,
4533
4534   type = example ,

```

```

4535 Name-sg = Ejemplo ,
4536 name-sg = ejemplo ,
4537 Name-pl = Ejemplos ,
4538 name-pl = ejemplos ,
4539
4540 type = algorithm ,
4541 Name-sg = Algoritmo ,
4542 name-sg = algoritmo ,
4543 Name-pl = Algoritmos ,
4544 name-pl = algoritmos ,
4545
4546 type = listing ,
4547 Name-sg = Listado ,
4548 name-sg = listado ,
4549 Name-pl = Listados ,
4550 name-pl = listados ,
4551
4552 type = exercise ,
4553 Name-sg = Ejercicio ,
4554 name-sg = ejercicio ,
4555 Name-pl = Ejercicios ,
4556 name-pl = ejercicios ,
4557
4558 type = solution ,
4559 Name-sg = Solución ,
4560 name-sg = solución ,
4561 Name-pl = Soluciones ,
4562 name-pl = soluciones ,
4563 </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		B	
<code>\</code>	112, 118, 127, 128, 133, 134, 139, 140, 145, 146, 181, 189, 190, 200	<code>\babelname</code>	792
<code>\{</code>	181	<code>\babelprovide</code>	14, 26
<code>\}</code>	181	<code>\begin</code>	80
† internal commands:		bool commands:	
<code>\_zrefclever_current_counter_tl</code>	5	<code>\bool_case_true:</code>	2
<b>A</b>		<code>\bool_if:NTF</code>	390, 401, 750, 754, 1434, 1851, 1946, 2076, 2098, 2129, 2175, 2224, 2247, 2251, 2257, 2267, 2273, 2430, 2592, 2594
<code>\AddToHook</code>	100, 587, 602, 746, 782, 807, 845, 847, 897, 1040, 1061, 2905, 2907, 2912, 2925, 2945, 2960, 2962, 2967, 2983, 2996, 3016, 3028, 3060, 3069, 3105, 3109, 3130, 3136, 3159	<code>\bool_if:nTF</code>	68, 1519, 1528, 1537, 1592, 1602, 1626, 1643, 1658, 1723, 1731, 1865, 1873, 2110, 2117, 2124, 2382, 2503
<code>\appendix</code>	76, 77	<code>\bool_lazy_all:nTF</code>	2196, 2604, 2825
<code>\appendixname</code>	76	<code>\bool_lazy_and:nTF</code>	1408, 1426, 2577, 2882

\bool_lazy_any:nTF	2705, 2714	\exp_args:NNnx	328
\bool_lazy_or:nnTF	1412, 2565	\exp_args:NNo	231, 237
\bool_new:N	337, 623, 624, 649, 673, 682, 689, 690, 703, 704, 723, 724, 975, 976, 977, 978, 1054, 1055, 1442, 1455, 1763, 1764, 1774, 1780, 1781, 3068	\exp_args:NNx	367, 907, 914, 1261
\bool_set:Nn	1405	\exp_args:Nnx	418
\bool_set_false:N	636, 640, 731, 740, 741, 756, 985, 989, 996, 1004, 1005, 1077, 1584, 1820, 1857, 1871, 1885, 2088, 2222, 2223, 2712, 2729	\exp_args:No	237
\bool_set_true:N	411, 630, 631, 635, 641, 730, 735, 736, 983, 990, 995, 1011, 1013, 1016, 1017, 1065, 1070, 1598, 1608, 1612, 1634, 1649, 1664, 1688, 1828, 1852, 1858, 1862, 1889, 1895, 2728, 2764, 2771, 2772, 2790, 2797, 2809, 3075	\exp_args:Nx	349, 3081, 3089
\bool_until_do:Nn	1624, 1821	\exp_args:Nxx	1555, 2756, 2778, 2782, 2799
\bool_while_do:nn	3164	\exp_not:N	64, 2131, 2134, 2145, 2148, 2151, 2388, 2391, 2394, 2403, 2405, 2408, 2411, 2417, 2419, 2437, 2447, 2450, 2452, 2455, 2462, 2469, 2471, 2475, 2478, 2481, 2493, 2496, 2509, 2512, 2515, 2531, 2533, 2536, 2539, 2546, 2548
<b>C</b>		\exp_not:n	238, 1975, 1991, 2003, 2008, 2031, 2045, 2049, 2061, 2065, 2099, 2100, 2132, 2144, 2149, 2150, 2287, 2300, 2307, 2331, 2343, 2347, 2357, 2361, 2389, 2390, 2392, 2398, 2401, 2404, 2409, 2410, 2412, 2413, 2416, 2418, 2448, 2449, 2451, 2453, 2454, 2456, 2457, 2461, 2473, 2474, 2479, 2480, 2482, 2490, 2494, 2495, 2497, 2510, 2511, 2513, 2525, 2529, 2532, 2537, 2538, 2540, 2541, 2545, 2547
clist commands:		\ExplSyntaxOn	14, 351
\clist_map_inline:nn	1006, 1120, 3042, 3123	<b>F</b>	
\counterwithin	4	file commands:	
\cref	79	\file_get:nnNTF	349
cs commands:		\fmtversion	3
\cs_generate_variant:Nn	65, 234, 240, 407, 415, 1200, 1288, 1294, 2424, 2742	\footnote	76
\cs_if_exist:NTF	25, 28, 46, 49, 58, 78, 2933, 2939	<b>G</b>	
\cs_if_exist_p:N	3166	group commands:	
\cs_new:Npn	56, 66, 76, 87, 235, 241, 2378, 2425, 2732	\group_begin:	102, 340, 410, 1250, 1402, 1416, 2131, 2148, 2388, 2391, 2408, 2411, 2447, 2452, 2455, 2471, 2478, 2493, 2509, 2512, 2536, 2539
\cs_new_eq:NN	2998, 3003, 3012, 3018, 3024	\group_end:	105, 405, 413, 1280, 1419, 1439, 2145, 2151, 2403, 2405, 2417, 2419, 2450, 2462, 2469, 2475, 2481, 2496, 2531, 2533, 2546, 2548
\cs_new_protected:Npn	229, 338, 408, 416, 422, 569, 905, 1198, 1283, 1289, 1400, 1459, 1501, 1512, 1571, 1701, 1753, 1797, 1953, 2218, 2374, 2376, 2555, 2743, 2819, 2876, 3076	<b>I</b>	
\cs_new_protected:Npx	99	\IfBooleanTF	1445
\cs_set_eq:NN	103, 2999, 3004, 3013, 3019, 3025	\ifdraft	988
\cs_set_nopar:Npn	2987	\IfFormatAtLeastTF	3, 4, 2992
<b>E</b>		\ifoptionfinal	994
\endinput	12	\input	14
exp commands:		int commands:	
\exp_args:NNe	35, 38	\int_case:nnTF	1956, 1984, 2016, 2178, 2280, 2319
\exp_args:NNNo	231		

`\int_compare:nNnTF` ..... 1559, 1635, 1650, 1665,  
1677, 1689, 1709, 1711, 1755, 1917,  
1971, 2005, 2167, 2169, 2235, 2260,  
2304, 2760, 2766, 2786, 2792, 3182  
`\int_compare_p:nNn` ..... 1725,  
1733, 2200, 2569, 2580, 2609, 2725  
`\int_eval:n` ..... 99  
`\int_incr:N` ..... 2213, 2250, 2252,  
2266, 2268, 2272, 2274, 2372, 3180  
`\int_new:N` .....  
.. 1456, 1457, 1765, 1766, 1777, 1778  
`\int_set:Nn` 1710, 1712, 1716, 1719, 3163  
`\int_to_roman:n` 3167, 3174, 3175, 3178  
`\int_use:N` ..... 47, 50, 54, 60  
`\int_zero:N` .....  
... 1703, 1704, 1806, 1807, 1808,  
1809, 2212, 2214, 2215, 2367, 2368  
`\l_tmpa_int` ..... 3163,  
3167, 3174, 3175, 3178, 3180, 3182  
iow commands:  
`\iow_char:N` .....  
.... 112, 118, 127, 128, 133, 134,  
139, 140, 145, 146, 181, 189, 190, 200  
  

### K

keys commands:  
`\keys_define:nn` 37, 428, 459, 476,  
490, 576, 606, 613, 625, 650, 659,  
674, 683, 691, 705, 717, 725, 758,  
765, 803, 850, 892, 899, 979, 1033,  
1035, 1042, 1049, 1056, 1066, 1078,  
1087, 1116, 1142, 1166, 1176, 1187,  
1211, 1223, 1295, 1326, 1347, 1370  
`\keys_set:nn` ..... 14, 28, 36,  
41, 383, 1071, 1199, 1206, 1277, 1403  
keyval commands:  
`\keyval_parse:nnn` ..... 1091, 1146  
  

### L

`\label` ..... 78, 79, 82  
`\labelformat` ..... 3  
`\language` ..... 25, 786  
  

### M

`\mainbabelname` ..... 25, 793  
`\MessageBreak` ..... 10  
MH commands:  
`\MH_if_boolean:nTF` ..... 3073  
msg commands:  
`\msg_info:nnn` .....  
467, 497, 2979, 3064, 3100, 3155, 3183  
`\msg_info:nnnn` ..... 440, 447  
`\msg_line_context:` . 111, 117, 121,  
123, 126, 132, 138, 144, 150, 155,  
160, 165, 171, 176, 179, 185, 188,  
194, 199, 206, 211, 215, 217, 220, 224  
`\msg_new:nnn` ..... 109,  
115, 120, 122, 124, 130, 136, 142,  
148, 153, 158, 163, 168, 173, 178,  
180, 182, 184, 186, 192, 197, 202,  
204, 209, 214, 216, 218, 223, 225, 227  
`\msg_note:nnn` ..... 386  
`\msg_warning:nn` .....  
. 592, 617, 755, 761, 1045, 1082, 2202  
`\msg_warning:nnn` .....  
.... 307, 332, 392, 402, 833, 878,  
895, 1021, 1148, 1215, 1279, 1338,  
1377, 1910, 2083, 2598, 2614, 2868  
`\msg_warning:nnnn` ..... 928, 945,  
958, 968, 1093, 1307, 1314, 2644, 2692  
  

### N

`\newcounter` ..... 4, 2929, 2930  
`\NewDocumentCommand` .....  
302, 322, 1196, 1201, 1248, 1398, 1443  
`\nobreakspace` ..... 518,  
3197, 3198, 3200, 3201, 3203, 3205,  
3393, 3394, 3396, 3397, 3399, 3401,  
4021, 4022, 4024, 4025, 4027, 4029,  
4200, 4201, 4203, 4204, 4206, 4208,  
4390, 4391, 4393, 4394, 4396, 4398  
  

### P

`\PackageError` ..... 7  
`\pagenumbering` ..... 7  
`\pageref` ..... 42  
prg commands:  
`\prg_generate_conditional_-`  
variant:Nnn ..... 542, 558  
`\prg_new_protected_conditional:Npnn`  
..... 528, 544, 561  
`\prg_return_false:` .....  
..... 538, 540, 554, 556, 567  
`\prg_return_true:` ..... 537, 553, 566  
`\ProcessKeysOptions` ..... 1195  
prop commands:  
`\prop_get:NnN` ..... 2845  
`\prop_get:NnNTF` ..... 342,  
531, 534, 547, 550, 564, 907, 1251,  
2623, 2651, 2659, 2822, 2879, 2892  
`\prop_gput:Nnn` .. 309, 329, 1285, 1291  
`\prop_gput_if_new:Nnn` ..... 418, 424  
`\prop_gset_from_keyval:Nn` ..... 512  
`\prop_if_exist:NTF` .....  
..... 312, 354, 911, 1203, 1255  
`\prop_if_exist_p:N` ..... 2829, 2885  
`\prop_if_in:NnTF` 35, 306, 326, 828, 873  
`\prop_if_in_p:Nn` ..... 69, 2836  
`\prop_item:Nn` 38, 70, 330, 370, 917, 1264



<code>\prop_new:N</code> . . . . .	301, 314, 360, 511, 1086, 1141, 1172, 1204, 1258
<code>\prop_put:Nnn</code> . . .	315, 573, 1183, 1238
<code>\prop_remove:Nn</code> . . . . .	572, 1182, 1230
<code>\providecommand</code> . . . . .	3
<code>\ProvidesExplPackage</code> . . . . .	14
<code>\ProvidesFile</code> . . . . .	14

  

<b>R</b>	
<code>\refstepcounter</code> . . . . .	3, 76, 80, 83
<code>\renewlist</code> . . . . .	83, 84
<code>\RequirePackage</code> . . . . .	16, 17, 18, 19, 20, 751, 1037, 1058

  

<b>S</b>	
<code>\scantokens</code> . . . . .	77
seq commands:	
<code>\seq_clear:N</code> . . . . .	670, 1461
<code>\seq_const_from_clist:Nn</code> . . . . .	247, 255, 268, 280
<code>\seq_gconcat:NNN</code> . . .	288, 291, 295, 298
<code>\seq_get_left:NN</code> . . . . .	380, 449, 938, 949, 1274, 1316, 1831
<code>\seq_gput_right:Nn</code> . . . . .	384, 395
<code>\seq_if_empty:NTF</code> . . . . .	377, 438, 924, 1271, 1305, 1825
<code>\seq_if_in:NnTF</code> . . . . .	345, 444, 942, 1122, 1311, 1505
<code>\seq_map_break:n</code> . . . . .	90, 1744, 1747
<code>\seq_map_function:NN</code> . . . . .	1464
<code>\seq_map_indexed_inline:Nn</code> . . .	23, 1705
<code>\seq_map_inline:Nn</code> . . . . .	456, 473, 487, 1173, 1208, 1220, 1323, 1344, 1367, 1741, 3079
<code>\seq_map_tokens:Nn</code> . . . . .	72
<code>\seq_new:N</code> . . . . .	246, 287, 294, 336, 658, 1115, 1441, 1458, 1762
<code>\seq_pop_left:NN</code> . . . . .	1823
<code>\seq_put_right:Nn</code> . . . . .	1124, 1508
<code>\seq_reverse:N</code> . . . . .	664
<code>\seq_set_eq:NN</code> . . . . .	1799
<code>\seq_set_from_clist:Nn</code> . . . . .	367, 663, 914, 1261, 1404
<code>\seq_sort:Nn</code> . . . . .	44, 1467
<code>\setcounter</code> . . . . .	2931, 2932, 2948, 2961, 2965
sort commands:	
<code>\sort_return_same:</code> . . . . .	45, 50, 1474, 1479, 1526, 1564, 1566, 1599, 1619, 1640, 1655, 1669, 1694, 1729, 1744, 1760
<code>\sort_return_swapped:</code> . . .	45, 50, 1487, 1535, 1563, 1609, 1618, 1639, 1654, 1670, 1693, 1737, 1747, 1759
<code>\stepcounter</code> . . . . .	2947, 2964

  

str commands:	
<code>\str_case:nnTF</code> . . . . .	809, 854, 1008
<code>\str_compare:nNnTF</code> . . . . .	1615
<code>\str_if_eq:nnTF</code> . . . . .	89
<code>\str_if_eq_p:nn</code> . . . . .	2710, 2716, 2718, 2722
<code>\str_new:N</code> . . . . .	764
<code>\str_set:Nn</code> . . . . .	769, 771, 773, 775
<code>\string</code> . . . . .	3086, 3094

  

<b>T</b>	
<code>\tag</code> . . . . .	80, 82
T <sub>E</sub> X and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:	
<code>\@Alph</code> . . . . .	76
<code>\@addtoreset</code> . . . . .	4
<code>\@auxout</code> . . . . .	3085, 3093
<code>\@bsphack</code> . . . . .	341, 3078
<code>\@chapapp</code> . . . . .	76
<code>\@currentcounter</code> . . . . .	3, 5, 34, 76, 80, 83, 28, 29, 49, 50, 1170
<code>\@currentlabel</code> . . . . .	3, 80, 83
<code>\@elt</code> . . . . .	5
<code>\@esphack</code> . . . . .	404, 3098
<code>\@ifl@t@r</code> . . . . .	3
<code>\@ifpackageloaded</code> . . .	589, 604, 748, 784, 790, 1063, 2927, 2985, 2994, 3008, 3010, 3071, 3107, 3138, 3161
<code>\@onlypreamble</code> . . . . .	321, 335, 1282
<code>\bbl@loaded</code> . . . . .	26
<code>\bbl@main@language</code> . . . . .	25, 787
<code>\c@</code> . . . . .	4
<code>\c@enumN</code> . . . . .	84
<code>\c@lstnumber</code> . . . . .	83
<code>\c@page</code> . . . . .	7, 103
<code>\c@l@</code> . . . . .	5
<code>\hyper@link</code> . . . . .	64, 2134, 2394, 2437, 2515
<code>\lst@AddToHook</code> . . . . .	3149
<code>\lst@label</code> . . . . .	3151, 3152
<code>\ltx@gobble</code> . . . . .	79
<code>\ltx@label</code> . . . . .	79, 2998, 2999, 3003, 3004, 3012, 3013, 3018, 3019, 3024, 3025
<code>\MT@newlabel</code> . . . . .	3086, 3094
<code>\p@.</code> . . . . .	3
<code>\protected@write</code> . . . . .	3085, 3093
<code>\zref@addprop</code> . . . . .	22, 32, 43, 53, 55, 97, 108
<code>\zref@default</code> . . . . .	64, 2375, 2377
<code>\zref@extractdefault</code> . . . . .	10, 72, 232, 238, 242
<code>\zref@ifpropundefined</code> . . . . .	21, 2734
<code>\zref@ifrefcontainsprop</code> . . . . .	21, 2380, 2432, 2499, 2737
<code>\zref@ifrefundefined</code> . . . . .	1469, 1471, 1483, 1854, 1856, 1861, 1905, 2080, 2089, 2226, 2427, 2557
<code>\zref@label</code> . . . . .	79, 2990

<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>	..... 1904
<code>\zref@wrapper@babel</code>	41, 79, 1399, 2990
<code>\textendash</code>	..... 522
<code>\textup</code>	..... 81
<code>\the</code>	..... 3
<code>\thechapter</code>	..... 76
<code>\thelstnumber</code>	..... 83
<code>\thepage</code>	..... 6, 7, 104
<code>\thesection</code>	..... 76
tl commands:	
<code>\c_empty_tl</code>	..... 1504, 1515, 1517, 1574, 1577, 1580, 1582, 1842, 1844, 2735, 2738, 2739, 2746, 2748
<code>\c_novalue_tl</code>	..... 1178, 1225
<code>\tl_clear:N</code>	.... 366, 378, 433, 932, 961, 971, 1254, 1272, 1300, 1801, 1802, 1803, 1804, 1805, 1827, 2208, 2209, 2210, 2211, 2249, 2558, 2561, 2589, 2643, 2691, 2867, 2898, 2900
<code>\tl_gset:Nn</code>	..... 104
<code>\tl_head:N</code>	..... 1653, 1666, 1678, 1680, 1690, 1692
<code>\tl_if_empty:NTF</code>	80, 464, 481, 495, 501, 926, 936, 956, 966, 1331, 1352, 1375, 1381, 1420, 1908, 2078, 2487, 2574, 2596, 2621, 2631, 2667, 3151
<code>\tl_if_empty:nTF</code>	..... 304, 310, 324, 432, 571, 1299, 2027, 2043, 2059, 2298, 2329, 2341, 2355, 2560
<code>\tl_if_empty_p:N</code>	..... 1523, 1524, 1532, 1533, 1540, 1541, 1868, 1869, 1876, 1878, 2607, 2709, 2719, 2723, 2827, 2883
<code>\tl_if_empty_p:n</code>	..... 1594, 1595, 1604, 1605, 1630, 1631, 1646, 1661
<code>\tl_if_eq:NNTF</code>	1544, 1588, 1881, 2750
<code>\tl_if_eq:NnTF</code>	..... 1462, 1494, 1715, 1718, 1743, 1746, 1835, 2754
<code>\tl_if_eq:nnTF</code>	..... 1555, 1707, 2756, 2778, 2782, 2799, 3081, 3089
<code>\tl_if_novalue:nTF</code>	..... 1181, 1228
<code>\tl_map_break:n</code>	..... 90
<code>\tl_map_tokens:Nn</code>	..... 82
<code>\tl_new:N</code>	..... 98, 243, 244, 245, 575, 779, 780, 781, 891, 1032, 1048, 1165, 1449, 1450, 1451, 1452, 1453, 1454, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1775, 1776, 1779, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 2903
<code>\tl_put_left:Nn</code>	..... 2113, 2120, 2160, 2633, 2634, 2669, 2671, 2673, 2675
<code>\tl_put_right:Nn</code>	..... 1973, 1989, 1998, 2029, 2040, 2056, 2285, 2296, 2327, 2339, 2353, 2575, 2576, 2587
<code>\tl_reverse:N</code>	..... 1575, 1578
<code>\tl_set:Nn</code>	..... 231, 434, 445, 580, 582, 584, 590, 593, 609, 618, 786, 787, 792, 793, 796, 797, 800, 813, 821, 830, 835, 858, 866, 875, 880, 1205, 1301, 1312, 1682, 1684, 1837, 1838, 1962, 1964, 2096, 2127, 2239, 2241, 2264, 2571, 2572, 2585, 2904, 2906
<code>\tl_set_eq:NN</code>	..... 2206
<code>\tl_tail:N</code>	..... 1683, 1685
<code>\l_tmpa_tl</code>	..... 352, 383, 1422, 1423
U	
<code>\upshape</code>	..... 3063
use commands:	
<code>\use:N</code>	..... 26, 29
V	
<code>\value</code>	..... 2948, 2965
Z	
<code>\zcDeclareLanguage</code>	..... 12, 302, 3188, 3385, 4015, 4195, 4388
<code>\zcDeclareLanguageAlias</code>	..... 13, 322, 3189, 3190, 3191, 3192, 3193, 3194, 3195, 3386, 3387, 3388, 3389, 3390, 3391, 4016, 4017, 4018, 4019, 4196, 4197, 4198
<code>\zcLanguageSetup</code>	11, 13, 14, 16, 36–39, 1248
<code>\zcpageref</code>	..... 42, 1443
<code>\zceref</code>	..... 27, 31, 35, 41–44, 51, 53, 79, 81, 1398, 1446, 1447
<code>\zcRefTypeSetup</code>	..... 11, 36, 1201, 3063
<code>\zcsetup</code>	..... 25, 31, 35, 36, 1196
<code>\zlabel</code>	..... 78–80, 82, 83, 3152
zrefcheck commands:	
<code>\zrefcheck_zceref_beg_label:</code>	.. 1411
<code>\zrefcheck_zceref_end_label_–</code>	maybe: ..... 1430
<code>\zrefcheck_zceref_run_checks_on_–</code>	labels:n ..... 1431
zrefclever internal commands:	
<code>\l_zrefclever_abbrev_bool</code>	..... 703, 707, 2578
<code>\l_zrefclever_capitalize_bool</code>	..... 689, 693, 2566
<code>\l_zrefclever_capitalize_first_–</code>	bool ..... 690, 699, 2568
<code>\_zrefclever_counter_reset_by:n</code>	..... 6, 33, 34, 58, 60, 62, 66, 3035, 3116

`\__zrefclever_counter_reset_by-`  
`aux:nn` ..... 73, 76  
`\__zrefclever_counter_reset_by-`  
`aux:nnn` ..... 83, 87  
`\l_zrefclever_counter_resetby-`  
`prop` ..... 5, 34, 69, 70, 1141, 1153  
`\l_zrefclever_counter_resettters-`  
`seq` . 5, 33, 34, 72, 1115, 1122, 1125  
`\l_zrefclever_counter_type_prop`  
..... 4, 32, 35, 38, 1086, 1098  
`\l_zrefclever_current_counter-`  
`tl` ..... 3, 34, 21, 25,  
26, 36, 39, 41, 46, 47, 95, 1165, 1168  
`\l_zrefclever_current_language-`  
`tl` .. 25, 781, 786, 792, 796, 822, 867  
`\__zrefclever_declare_default-`  
`transl:nnn` ... 38, 1283, 1333, 1354  
`\__zrefclever_declare_type-`  
`transl:nnnn` .....  
..... 38, 1283, 1359, 1383, 1389  
`\__zrefclever_def_extract:Nnnn` ..  
..... 10,  
229, 1503, 1514, 1516, 1573, 1576,  
1579, 1581, 1841, 1843, 2745, 2747  
`\g_zrefclever_dict_⟨language⟩_prop`  
..... 14  
`\l_zrefclever_dict_decl_case_tl`  
.... 243, 378, 381, 445, 450, 501,  
505, 1272, 1275, 1312, 1317, 1381, 1392  
`\l_zrefclever_dict_declension-`  
`seq` ..... 243, 368, 377, 380,  
438, 444, 449, 915, 924, 938, 942,  
949, 1262, 1271, 1274, 1305, 1311, 1316  
`\l_zrefclever_dict_language_tl` .  
. 243, 343, 347, 350, 357, 363, 373,  
385, 387, 393, 396, 419, 425, 441,  
448, 532, 535, 548, 551, 909, 912,  
920, 1252, 1256, 1259, 1267, 1308,  
1315, 1334, 1355, 1360, 1384, 1390  
`\__zrefclever_extract:nnn` .....  
.. 10, 241, 1560, 1562, 1636, 1638,  
1651, 1668, 1756, 1758, 2761, 2763,  
2767, 2769, 2787, 2789, 2793, 2795  
`\__zrefclever_extract_unexp:nnn` .  
..... 10, 72, 235,  
1556, 1557, 2140, 2396, 2399, 2414,  
2443, 2458, 2521, 2526, 2542, 2735,  
2738, 2739, 2757, 2758, 2779, 2780,  
2783, 2784, 2801, 2805, 3082, 3090  
`\__zrefclever_extract_url-`  
`unexp:n` 2136, 2395, 2439, 2517, 2732  
`\g_zrefclever_fallback_dict-`  
`prop` ..... 10, 511, 512, 564  
`\l_zrefclever_footnote_type_tl` .  
..... 2903, 2904, 2906, 2910  
`\__zrefclever_get_default-`  
`transl:nnN` ..... 11, 545, 559  
`\__zrefclever_get_default-`  
`transl:nnNTF` ..... 19, 544, 2860  
`\__zrefclever_get_enclosing-`  
`counters_value:n` . 5, 6, 56, 61, 94  
`\__zrefclever_get_fallback-`  
`transl:nN` ..... 562  
`\__zrefclever_get_fallback-`  
`transl:nNTF` ..... 20, 560, 2865  
`\__zrefclever_get_ref:n` .....  
..... 64, 65, 1976, 1992,  
2004, 2009, 2032, 2046, 2050, 2062,  
2066, 2101, 2121, 2288, 2301, 2308,  
2332, 2344, 2348, 2358, 2362, 2378  
`\__zrefclever_get_ref_first:` ...  
..... 64, 68, 2114, 2161, 2425  
`\__zrefclever_get_ref_font:nN` 10,  
18, 35, 74, 75, 1937, 1939, 1941, 2876  
`\__zrefclever_get_ref_string:nN` .  
... 10, 11, 18, 35, 74, 1422, 1812,  
1814, 1816, 1919, 1921, 1923, 1925,  
1927, 1929, 1931, 1933, 1935, 2819  
`\__zrefclever_get_type_transl:nnnN`  
..... 11, 529, 543  
`\__zrefclever_get_type_transl:nnnNTF`  
..... 19, 528, 2637, 2679, 2685, 2854  
`\l_zrefclever_label_a_tl` .....  
. 50, 1767, 1824, 1842, 1854, 1904,  
1905, 1911, 1963, 1976, 1992, 2009,  
2050, 2066, 2094, 2101, 2226, 2230,  
2240, 2265, 2288, 2309, 2348, 2362  
`\l_zrefclever_label_b_tl` .....  
..... 50, 1767,  
1827, 1832, 1844, 1856, 1861, 2230  
`\l_zrefclever_label_count_int` ..  
..... 51, 1765, 1806,  
1917, 1956, 2212, 2235, 2372, 2610  
`\l_zrefclever_label_enclval_a-`  
`tl` ..... 1449, 1573, 1575, 1630,  
1646, 1666, 1678, 1682, 1683, 1690  
`\l_zrefclever_label_enclval_b-`  
`tl` ..... 1449, 1576, 1578, 1631,  
1653, 1661, 1680, 1684, 1685, 1692  
`\l_zrefclever_label_extdoc_a_tl`  
..... 1449, 1579,  
1589, 1594, 1604, 1617, 2745, 2751  
`\l_zrefclever_label_extdoc_b_tl`  
..... 1449, 1581,  
1590, 1595, 1605, 1616, 2747, 2752  
`\l_zrefclever_label_type_a_tl` ..  
..... 74, 1449, 1504, 1506,

1509, 1515, 1523, 1532, 1540, 1545,  
 1715, 1743, 1837, 1841, 1868, 1876,  
 1882, 1908, 1965, 2242, 2827, 2832,  
 2839, 2848, 2856, 2883, 2888, 2895  
 \l\_zrefclever\_label\_type\_b\_tl ..  
     ..... 1449,  
     1517, 1524, 1533, 1541, 1546, 1718,  
     1746, 1838, 1843, 1869, 1878, 1883  
 \\_zrefclever\_label\_type\_put\_-  
   new\_right:n .... 43, 45, 1465, 1501  
 \l\_zrefclever\_label\_types\_seq ..  
     .... 44, 1458, 1461, 1505, 1508, 1741  
 \\_zrefclever\_labels\_in\_sequence:nn  
     ..... 51, 72, 2092, 2229, 2743  
 \g\_zrefclever\_languages\_prop ...  
     ..... 13, 301, 306, 309, 326, 329,  
     330, 342, 531, 547, 828, 873, 907, 1251  
 \l\_zrefclever\_last\_of\_type\_bool  
     ..... 51, 1762, 1852, 1857, 1858,  
     1862, 1871, 1886, 1890, 1896, 1946  
 \l\_zrefclever\_lastsep\_tl . 1782,  
     1928, 1991, 2008, 2031, 2049, 2061  
 \l\_zrefclever\_link\_star\_bool ...  
     ..... 1405, 1441, 2385, 2506, 2708  
 \l\_zrefclever\_listsep\_tl .....  
     ... 1782, 1926, 2003, 2045, 2287,  
     2300, 2307, 2331, 2343, 2347, 2357  
 \l\_zrefclever\_load\_dict\_-  
   verbose\_bool ... 337, 390, 401, 411  
 \g\_zrefclever\_loaded\_dictionaries\_-  
   seq ..... 336, 346, 384, 395  
 \\_zrefclever\_ltxlabel:n .....  
     79, 2987, 2999, 3004, 3013, 3019, 3025  
 \l\_zrefclever\_main\_language\_tl .  
     ..... 25, 780,  
     787, 793, 797, 801, 814, 836, 859, 881  
 \\_zrefclever\_mathtools\_showonlyrefs:n  
     ..... 1436, 3076  
 \l\_zrefclever\_mathtools\_-  
   showonlyrefs\_bool 1434, 3068, 3075  
 \\_zrefclever\_name\_default: ....  
     ..... 2374, 2489  
 \l\_zrefclever\_name\_format\_-  
   fallback\_tl ..... 1773, 2585,  
     2589, 2621, 2664, 2674, 2676, 2688  
 \l\_zrefclever\_name\_format\_tl ...  
     ... 1773, 2571, 2572, 2575, 2576,  
     2586, 2587, 2628, 2633, 2634, 2640,  
     2645, 2656, 2670, 2671, 2682, 2694  
 \l\_zrefclever\_name\_in\_link\_bool  
     ..... 66,  
     68, 1773, 2129, 2430, 2712, 2728, 2729  
 \l\_zrefclever\_namefont\_tl 1782,  
     1938, 2132, 2149, 2448, 2479, 2494  
 \l\_zrefclever\_nameinlink\_str ...  
     ..... 764, 769,  
     771, 773, 775, 2710, 2716, 2718, 2722  
 \l\_zrefclever\_namesep\_tl .....  
     ... 1782, 1920, 2451, 2482, 2490, 2497  
 \l\_zrefclever\_next\_is\_same\_bool  
     ..... 51, 72, 1777,  
     2223, 2251, 2267, 2273, 2772, 2810  
 \l\_zrefclever\_next\_maybe\_range\_-  
   bool .....  
     ... 51, 72, 1777, 2088, 2098, 2222,  
     2247, 2257, 2764, 2771, 2790, 2798  
 \l\_zrefclever\_noabbrev\_first\_-  
   bool ..... 704, 713, 2582  
 \l\_zrefclever\_nudge\_comptosing\_-  
   bool ... 977, 1005, 1013, 1017, 2606  
 \l\_zrefclever\_nudge\_enabled\_-  
   bool ..... 975,  
     983, 985, 989, 990, 995, 996, 2198, 2592  
 \l\_zrefclever\_nudge\_multitype\_-  
   bool ... 976, 1004, 1011, 1016, 2199  
 \l\_zrefclever\_nudge\_singular\_-  
   bool ..... 978, 1028, 2594  
 \\_zrefclever\_orig\_ltxlabel:n ...  
     ... 2989, 2998, 3003, 3012, 3018, 3024  
 \\_zrefclever\_page\_format\_aux: ..  
     ..... 99, 103  
 \g\_zrefclever\_page\_format\_tl ...  
     ..... 7, 98, 104, 107  
 \l\_zrefclever\_pairsep\_tl .....  
     ..... 1782, 1924, 1975, 2099  
 \\_zrefclever\_prop\_put\_non\_-  
   empty:Nnn .... 20, 569, 1097, 1152  
 \\_zrefclever\_provide\_dict\_-  
   default\_transl:nn 16, 416, 465, 482  
 \\_zrefclever\_provide\_dict\_type\_-  
   transl:nn ... 16, 416, 483, 502, 504  
 \\_zrefclever\_provide\_dictionary:n  
     ..... 11, 14, 16, 17,  
     41, 338, 412, 849, 860, 868, 883, 1406  
 \\_zrefclever\_provide\_dictionary\_-  
   verbose:n ... 16, 408, 815, 823, 838  
 \l\_zrefclever\_range\_beg\_label\_-  
   tl ..... 51, 1777, 1805,  
     2004, 2027, 2033, 2043, 2047, 2059,  
     2063, 2211, 2249, 2264, 2298, 2302,  
     2329, 2333, 2341, 2345, 2355, 2359  
 \l\_zrefclever\_range\_count\_int ..  
     ..... 51,  
     1777, 1808, 1984, 2018, 2214, 2250,  
     2261, 2266, 2272, 2280, 2321, 2367  
 \l\_zrefclever\_range\_same\_count\_-  
   int ..... 51,

[1777](#), [1809](#), [1971](#), [2006](#), [2019](#), [2215](#),  
[2252](#), [2268](#), [2274](#), [2305](#), [2322](#), [2368](#)  
\l\_zrefclever\_rangeseq\_tl .....  
..... [1782](#), [1922](#), [2065](#), [2100](#), [2361](#)  
\l\_zrefclever\_ref\_decl\_case\_tl .  
..... [28](#),  
[891](#), [901](#), [926](#), [931](#), [932](#), [936](#), [939](#),  
[943](#), [947](#), [950](#), [956](#), [960](#), [961](#), [966](#),  
[969](#), [971](#), [2631](#), [2635](#), [2667](#), [2672](#), [2677](#)  
\\_zrefclever\_ref\_default: .....  
..... [2374](#), [2422](#), [2428](#), [2483](#), [2551](#)  
\l\_zrefclever\_ref\_language\_tl ..  
..... [25](#), [26](#), [28](#),  
[779](#), [800](#), [813](#), [816](#), [821](#), [824](#), [830](#),  
[835](#), [839](#), [849](#), [858](#), [861](#), [866](#), [869](#),  
[875](#), [880](#), [884](#), [908](#), [930](#), [948](#), [959](#),  
[970](#), [1406](#), [2638](#), [2680](#), [2686](#), [2855](#), [2861](#)  
\c\_zrefclever\_ref\_options\_font-  
seq ..... [11](#), [18](#), [247](#)  
\c\_zrefclever\_ref\_options-  
necessarily\_not\_type\_specific-  
seq ..... [18](#), [247](#), [457](#), [1209](#), [1324](#)  
\c\_zrefclever\_ref\_options-  
possibly\_type\_specific\_seq ..  
..... [18](#), [247](#), [474](#), [1345](#)  
\l\_zrefclever\_ref\_options\_prop .  
. [35](#), [36](#), [1172](#), [1182](#), [1183](#), [2822](#), [2879](#)  
\c\_zrefclever\_ref\_options-  
reference\_seq ..... [247](#), [1174](#)  
\c\_zrefclever\_ref\_options\_type-  
names\_seq ..... [247](#), [488](#), [1368](#)  
\c\_zrefclever\_ref\_options-  
typesetup\_seq ..... [247](#), [1221](#)  
\l\_zrefclever\_ref\_property\_tl ..  
..... [21](#), [575](#), [580](#),  
[582](#), [584](#), [590](#), [593](#), [609](#), [618](#), [1462](#),  
[1494](#), [1835](#), [2380](#), [2434](#), [2501](#), [2754](#)  
\l\_zrefclever\_ref\_typeset\_font-  
tl ..... [1032](#), [1034](#), [1417](#)  
\l\_zrefclever\_reffont\_in\_tl [1782](#),  
[1942](#), [2392](#), [2412](#), [2456](#), [2513](#), [2540](#)  
\l\_zrefclever\_reffont\_out\_tl ...  
..... [1782](#), [1940](#),  
[2389](#), [2409](#), [2453](#), [2473](#), [2510](#), [2537](#)  
\l\_zrefclever\_refpos\_in\_tl [1782](#),  
[1936](#), [2401](#), [2416](#), [2461](#), [2529](#), [2545](#)  
\l\_zrefclever\_refpos\_out\_tl [1782](#),  
[1932](#), [2404](#), [2418](#), [2474](#), [2532](#), [2547](#)  
\l\_zrefclever\_refpre\_in\_tl [1782](#),  
[1934](#), [2398](#), [2413](#), [2457](#), [2525](#), [2541](#)  
\l\_zrefclever\_refpre\_out\_tl [1782](#),  
[1930](#), [2390](#), [2410](#), [2454](#), [2511](#), [2538](#)  
\l\_zrefclever\_setup\_type\_tl ...  
..... [16](#), [243](#),  
[366](#), [420](#), [433](#), [434](#), [464](#), [481](#), [495](#),  
[1205](#), [1233](#), [1241](#), [1254](#), [1300](#), [1301](#),  
[1331](#), [1352](#), [1361](#), [1375](#), [1385](#), [1391](#)  
\l\_zrefclever\_sort\_decided\_bool  
..... [1455](#), [1584](#), [1598](#), [1608](#),  
[1612](#), [1624](#), [1634](#), [1649](#), [1664](#), [1688](#)  
\\_zrefclever\_sort\_default:nn ...  
..... [45](#), [1496](#), [1512](#)  
\\_zrefclever\_sort\_default-  
different\_types:nn .....  
..... [22](#), [43](#), [49](#), [1550](#), [1701](#)  
\\_zrefclever\_sort\_default\_same-  
type:nn ..... [43](#), [46](#), [1548](#), [1571](#)  
\\_zrefclever\_sort\_labels: .....  
..... [43-45](#), [50](#), [1415](#), [1459](#)  
\\_zrefclever\_sort\_page:nn .....  
..... [50](#), [1495](#), [1753](#)  
\l\_zrefclever\_sort\_prior\_a\_int .  
..... [1456](#),  
[1703](#), [1709](#), [1710](#), [1716](#), [1726](#), [1734](#)  
\l\_zrefclever\_sort\_prior\_b\_int .  
..... [1456](#),  
[1704](#), [1711](#), [1712](#), [1719](#), [1727](#), [1735](#)  
\l\_zrefclever\_tlastsep\_tl .....  
..... [1782](#), [1817](#), [2192](#)  
\l\_zrefclever\_tlistsep\_tl .....  
..... [1782](#), [1815](#), [2170](#)  
\l\_zrefclever\_tpairsep\_tl .....  
..... [1782](#), [1813](#), [2186](#)  
\l\_zrefclever\_type\_<type>-  
options\_prop ..... [36](#)  
\l\_zrefclever\_type\_count\_int ...  
. [51](#), [68](#), [1765](#), [1807](#), [2167](#), [2169](#),  
[2178](#), [2200](#), [2213](#), [2569](#), [2581](#), [2725](#)  
\l\_zrefclever\_type\_first\_label-  
tl [51](#), [66](#), [1767](#), [1803](#), [1962](#), [2080](#),  
[2089](#), [2093](#), [2121](#), [2137](#), [2141](#), [2209](#),  
[2239](#), [2427](#), [2433](#), [2440](#), [2444](#), [2459](#),  
[2500](#), [2518](#), [2522](#), [2527](#), [2543](#), [2557](#)  
\l\_zrefclever\_type\_first\_label-  
type\_tl ..... [51](#),  
[68](#), [1767](#), [1804](#), [1964](#), [2084](#), [2210](#),  
[2241](#), [2560](#), [2600](#), [2616](#), [2626](#), [2639](#),  
[2646](#), [2654](#), [2662](#), [2681](#), [2687](#), [2695](#)  
\\_zrefclever\_type\_name\_setup: ..  
..... [10](#), [11](#), [66](#), [2109](#), [2555](#)  
\l\_zrefclever\_type\_name\_tl .....  
..... [66](#), [68](#),  
[1773](#), [2144](#), [2150](#), [2449](#), [2480](#), [2487](#),  
[2495](#), [2558](#), [2561](#), [2629](#), [2641](#), [2643](#),  
[2657](#), [2665](#), [2683](#), [2689](#), [2691](#), [2709](#)  
\l\_zrefclever\_typeset\_compress-  
bool ..... [673](#), [676](#), [2224](#)

<code>\l_zrefclever_typeset_labels_-</code>	<code>\l_zrefclever_typeset_sort_bool</code>
seq <a href="#">50</a> , <a href="#">1762</a> , <a href="#">1799</a> , <a href="#">1823</a> , <a href="#">1825</a> , <a href="#">1831</a>	..... <a href="#">649</a> , <a href="#">652</a> , <a href="#">1413</a>
<code>\l_zrefclever_typeset_last_bool</code>	<code>\l_zrefclever_typesort_seq</code> ....
..... <a href="#">51</a> , <a href="#">1762</a> ,	.... <a href="#">22</a> , <a href="#">49</a> , <a href="#">658</a> , <a href="#">663</a> , <a href="#">664</a> , <a href="#">670</a> , <a href="#">1705</a>
<a href="#">1820</a> , <a href="#">1821</a> , <a href="#">1828</a> , <a href="#">1851</a> , <a href="#">2175</a> , <a href="#">2724</a>	<code>\l_zrefclever_use_hyperref_bool</code>
<code>\l_zrefclever_typeset_name_bool</code>	..... <a href="#">723</a> , <a href="#">730</a> ,
..... <a href="#">624</a> , <a href="#">631</a> , <a href="#">636</a> , <a href="#">641</a> , <a href="#">2111</a> , <a href="#">2125</a>	<a href="#">735</a> , <a href="#">740</a> , <a href="#">750</a> , <a href="#">756</a> , <a href="#">2384</a> , <a href="#">2505</a> , <a href="#">2707</a>
<code>\l_zrefclever_typeset_queue_-</code>	<code>\_zrefclever_validate_decl_-</code>
curr_tl ..... <a href="#">51</a> , <a href="#">64</a> ,	option: ..... <a href="#">905</a> , <a href="#">1407</a>
<a href="#">68</a> , <a href="#">1767</a> , <a href="#">1802</a> , <a href="#">1973</a> , <a href="#">1989</a> , <a href="#">1998</a> ,	<code>\l_zrefclever_warn_hyperref_-</code>
<a href="#">2029</a> , <a href="#">2040</a> , <a href="#">2056</a> , <a href="#">2078</a> , <a href="#">2096</a> , <a href="#">2113</a> ,	bool ..... <a href="#">724</a> , <a href="#">731</a> , <a href="#">736</a> , <a href="#">741</a> , <a href="#">754</a>
<a href="#">2120</a> , <a href="#">2127</a> , <a href="#">2160</a> , <a href="#">2182</a> , <a href="#">2187</a> , <a href="#">2193</a> ,	<code>\_zrefclever_zcref:nnn</code> <a href="#">28</a> , <a href="#">1399</a> , <a href="#">1400</a>
<a href="#">2207</a> , <a href="#">2208</a> , <a href="#">2285</a> , <a href="#">2296</a> , <a href="#">2327</a> , <a href="#">2339</a> ,	<code>\_zrefclever_zcref:nnnn</code> <a href="#">41</a> , <a href="#">43</a> , <a href="#">1400</a>
<a href="#">2353</a> , <a href="#">2574</a> , <a href="#">2596</a> , <a href="#">2607</a> , <a href="#">2719</a> , <a href="#">2723</a>	<code>\l_zrefclever_zcref_labels_seq</code> .
<code>\l_zrefclever_typeset_queue_-</code>	..... <a href="#">43</a> , <a href="#">44</a> , <a href="#">1404</a> ,
prev_tl . <a href="#">51</a> , <a href="#">1767</a> , <a href="#">1801</a> , <a href="#">2171</a> , <a href="#">2206</a>	<a href="#">1432</a> , <a href="#">1437</a> , <a href="#">1441</a> , <a href="#">1464</a> , <a href="#">1467</a> , <a href="#">1800</a>
<code>\l_zrefclever_typeset_range_-</code>	<code>\l_zrefclever_zcref_note_tl</code> ...
bool ..... <a href="#">682</a> , <a href="#">685</a> , <a href="#">1414</a> , <a href="#">2076</a>	..... <a href="#">1048</a> , <a href="#">1051</a> , <a href="#">1420</a> , <a href="#">1424</a>
<code>\l_zrefclever_typeset_ref_bool</code> .	<code>\l_zrefclever_zcref_with_check_-</code>
..... <a href="#">623</a> , <a href="#">630</a> , <a href="#">635</a> , <a href="#">640</a> , <a href="#">2111</a> , <a href="#">2118</a>	bool ..... <a href="#">1055</a> , <a href="#">1070</a> , <a href="#">1410</a> , <a href="#">1428</a>
<code>\_zrefclever_typeset_refs: ....</code>	<code>\_zrefclever_zcsetup:n</code> ..... ..... <a href="#">36</a> , <a href="#">1197</a> , <a href="#">1198</a> , <a href="#">2909</a> ,
..... <a href="#">50</a> , <a href="#">52</a> , <a href="#">1418</a> , <a href="#">1797</a>	<a href="#">2914</a> , <a href="#">2935</a> , <a href="#">2941</a> , <a href="#">2949</a> , <a href="#">2969</a> , <a href="#">3030</a> ,
<code>\_zrefclever_typeset_refs_last_-</code>	<a href="#">3061</a> , <a href="#">3111</a> , <a href="#">3131</a> , <a href="#">3140</a> , <a href="#">3153</a> , <a href="#">3170</a>
of_type: . <a href="#">56</a> , <a href="#">64</a> , <a href="#">66</a> , <a href="#">68</a> , <a href="#">1948</a> , <a href="#">1953</a>	<code>\l_zrefclever_zrefcheck_-</code>
<code>\_zrefclever_typeset_refs_not_-</code>	available_bool ..... ..... <a href="#">1054</a> , <a href="#">1065</a> , <a href="#">1077</a> , <a href="#">1409</a> , <a href="#">1427</a>
last_of_type: ..... ..... <a href="#">51</a> , <a href="#">56</a> , <a href="#">64</a> , <a href="#">72</a> , <a href="#">1950</a> , <a href="#">2218</a>	