

# The zref-clever package implementation\*

Gustavo Barros<sup>†</sup>

2021-09-13

## 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	Reference options variables	8
4.3	Internationalization	9
4.4	Options	14
4.4.1	Auxiliary	14
4.4.2	countertype option	14
4.4.3	counterresetters option	15
4.4.4	counterresetby option	15
4.4.5	ref option	16
4.4.6	typeset option	17
4.4.7	sort option	18
4.4.8	typesort option	18
4.4.9	comp option	18
4.4.10	range option	18
4.4.11	hyperref option	19
4.4.12	nameinlink option	19
4.4.13	cap and capfirst options	20
4.4.14	abbrev and noabbrevfirst options	20
4.4.15	lang option	20
4.4.16	font option	23
4.4.17	note option	23
4.4.18	check option	23
4.4.19	Reference options	24
4.5	\zcsetup	24
4.6	Package options	25

---

\*This file describes v0.1.0-alpha, released 2021-09-13.

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

<b>5</b>	<b>Reference format</b>	<b>25</b>
5.1	<code>\zcRefTypeSetup</code> . . . . .	26
5.2	<code>\zcDeclareTranslations</code> . . . . .	27
<b>6</b>	<b>User interface</b>	<b>29</b>
6.1	<code>\zcref</code> . . . . .	29
6.2	<code>\zcpageref</code> . . . . .	30
<b>7</b>	<b>Sorting</b>	<b>30</b>
<b>8</b>	<b>Typesetting</b>	<b>38</b>
<b>9</b>	<b>Special handling</b>	<b>59</b>
9.1	<code>\appendix</code> . . . . .	59
9.2	<code>\newtheorem</code> . . . . .	59
9.3	<code>enumitem</code> package . . . . .	59
<b>10</b>	<b>Dictionaries</b>	<b>59</b>
10.1	English . . . . .	59
10.2	German . . . . .	63
10.3	French . . . . .	66
10.4	Portuguese . . . . .	70
10.5	Spanish . . . . .	73
	<b>Index</b>	<b>76</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.

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

```

12     \endinput
13 }%
    Identify the package.
14 \ProvidesExplPackage {zref-clever} {2021-09-13} {0.1.0-alpha}
15 {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-counter }
19 \RequirePackage { zref-abspage }
20 \RequirePackage { l3keys2e }

```

## 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 `page` and `counter` properties are respectively provided by modules `zref-base` and `zref-counter`. The `zref-abspage` provides the `abspage` property which gives us a safe and easy way to sort labels for page references.

But 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. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@... prefix`.

```

21 \zref@newprop { zc@thecnt } { \use:c { the \@currentcounter } }
22 \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`.

```

23 \zref@newprop { zc@type }
24 {
25   \prop_if_in:NVTF \l__zrefclever_counter_type_prop \@currentcounter
26   {
27     \exp_args:NNe \prop_item:Nn
28     \l__zrefclever_counter_type_prop { \@currentcounter }
29   }
30   { \@currentcounter }

```

```

31 }
32 \zref@addprop \ZREF@mainlist { zc@type }

```

Since the `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’).

```

33 \zref@newprop { zc@cntval } [0] { \int_use:c { c@ \@currentcounter } }
34 \zref@addprop \ZREF@mainlist { zc@cntval }
35 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
36 \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”, “supercounter” etc.). This information is a little trickier to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\cl@⟨counter⟩` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see section ‘ltcounts.dtx’ in ‘source2e’). Besides, there may be a chain of resetting counters, which must be taken into account: if ‘counterC’ gets reset by ‘counterB’, and ‘counterB’ gets reset by ‘counterA’, stepping the latter affects all three of them.

The procedure below examines a set of counters, those included in `\l__zrefclever_counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\cl@⟨counter⟩`, looking for the counter for which we are trying to set a label (`\@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 `\c1@⟨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_resetters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`\__zrefclever_get_enclosing_counters:n`  
`\__zrefclever_get_enclosing_counters_value:n`

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

```

    \__zrefclever_get_enclosing_counters:n {⟨counter⟩}
    \__zrefclever_get_enclosing_counters_value:n {⟨counter⟩}

37 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
38 {
39   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
40   {
41     { \__zrefclever_counter_reset_by:n {#1} }
42     \__zrefclever_get_enclosing_counters:e
43     { \__zrefclever_counter_reset_by:n {#1} }
44   }
45 }
46 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
47 {
48   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
49   {
50     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
51     \__zrefclever_get_enclosing_counters_value:e
52     { \__zrefclever_counter_reset_by:n {#1} }
53   }
54 }

```

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’).

```

55 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { V , e }
56 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { V , e }

```

(End definition for `\__zrefclever_get_enclosing_counters:n` and `\__zrefclever_get_enclosing_counters_value:n`.)

`\_zrefclever_counter_reset_by:n` Auxiliary function for `\_zrefclever_get_enclosing_counters:n` and `\_zrefclever_get_enclosing_counters_value:n`. They are broken in parts to be able to use the expandable mapping functions. `\_zrefclever_counter_reset_by:n` leaves in the stream the “enclosing counter” which resets  $\langle counter \rangle$ .

```

\__zrefclever_counter_reset_by:n {<counter>}

57 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
58 {
59   \bool_if:nTF
60     { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
61     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
62     {
63       \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
64         { \__zrefclever_counter_reset_by_aux:nn {#1} }
65     }
66 }
67 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
68 {
69   \cs_if_exist:cT { c@ #2 }
70   {
71     \tl_if_empty:cF { cl@ #2 }
72     {
73       \tl_map_tokens:cn { cl@ #2 }
74       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
75     }
76   }
77 }
78 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
79 {
80   \str_if_eq:nnT {#2} {#3}
81   { \tl_map_break:n { \seq_map_break:n {#1} } }
82 }

```

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

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

```

83 \zref@newprop { zc@enclcnt }
84 { \__zrefclever_get_enclosing_counters:V \@currentcounter }
85 \zref@newprop { zc@enclval }
86 { \__zrefclever_get_enclosing_counters_value:V \@currentcounter }
87 \zref@addprop \ZREF@mainlist { zc@enclcnt }
88 \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}`.

```

89 \tl_new:N \g__zrefclever_page_format_tl
90 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
91 \AddToHook { shipout / before }
92 {
93   \group_begin:
94   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
95   \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
96   \group_end:
97 }
98 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
99 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

Still another property which we don’t need to handle at the data provision side, but need to cater for at the retrieval side, is the `url` property (or the equivalent `urluse`) from the `zref-xr` module, which is added to the labels imported from external documents, and needed to construct hyperlinks to them.

## 4 Plumbing

### 4.1 Messages

```

100 \msg_new:nnn { zref-clever } { option-not-type-specific }
101 {
102   Option~'#1'~is-not-type-specific~\msg_line_context:~
103   Set-it-in~'\iow_char:N\zcDeclareTranslations'~before~first~'type'~switch-
104   or~as~package~option.
105 }
106 \msg_new:nnn { zref-clever } { option-only-type-specific }
107 {
108   No~type~specified~for~option~'#1'~\msg_line_context:~
109   Set-it-after~'type'~switch~or~in~'\iow_char:N\zcRefTypeSetup'.
110 }
111 \msg_new:nnn { zref-clever } { key-requires-value }
112 { The~'#1'~key~'#2'~requires~a~value. }
113 \msg_new:nnn { zref-clever } { language-declared }
114 { Language~'#1'~is~already~declared.~Nothing~to~do. }
115 \msg_new:nnn { zref-clever } { alias-declared }
116 { Language~'#1'~is~already~an~alias~to~'#2'.~Nothing~to~do. }
117 \msg_new:nnn { zref-clever } { unknown-language-alias }
118 {
119   Language~'#1'~is~unknown,~cannot~alias~to~it.~See~documentation~for~
120   '\iow_char:N\zcDeclareLanguage'~and~'\iow_char:N\zcDeclareLanguageAlias'.
121 }

```

```

122 \msg_new:nnn { zref-clever } { unknown-language-transl }
123 {
124   Language~'#1'~is~unknown,~cannot~declare~translations~to~it.~
125   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
126   '\iow_char:N\zcDeclareLanguageAlias'.
127 }
128 \msg_new:nnn { zref-clever } { dict-loaded }
129 { Loaded~'#1'~dictionary. }
130 \msg_new:nnn { zref-clever } { dict-not-available }
131 { Dictionary~for~'#1'~not~available. }
132 \msg_new:nnn { zref-clever } { unknown-language-load }
133 {
134   Unable~to~load~dictionary.~Language~'#1'~is~unknown.~See~documentation~for~
135   '\iow_char:N\zcDeclareLanguage'~and~'\iow_char:N\zcDeclareLanguageAlias'.
136 }
137 \msg_new:nnn { zref-clever } { missing-zref-titleref }
138 {
139   Option~'ref=title'~requested~\msg_line_context:~
140   But~package~'zref-titleref'~is~not~loaded,~falling~back~to~default~'ref'.
141 }
142 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
143 {
144   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
145   Use~the~starred~version~of~'\iow_char:N\zceref'~instead.
146 }
147 \msg_new:nnn { zref-clever } { missing-hyperref }
148 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
149 \msg_new:nnn { zref-check } { check-document-only }
150 { Option~'check'~only~available~in~the~document. }
151 \msg_new:nnn { zref-clever } { missing-zref-check }
152 {
153   Option~'check'~requested~\msg_line_context:~
154   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
155 }
156 \msg_new:nnn { zref-clever } { counters-not-nested }
157 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:. }
158 \msg_new:nnn { zref-clever } { missing-type }
159 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
160 \msg_new:nnn { zref-clever } { missing-name }
161 { Name~undefined~for~type~'#1'~\msg_line_context:. }
162 \msg_new:nnn { zref-clever } { single-element-range }
163 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }

```

## 4.2 Reference options variables

```

164 \seq_const_from_clist:Nn
165 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
166 {
167   tpairsep ,
168   tlistsep ,
169   tlastsep ,
170   noteseq ,
171 }
172 \seq_const_from_clist:Nn
173 \c__zrefclever_ref_options_possibly_type_specific_seq

```



```

174 {
175     namesep ,
176     pairsep ,
177     listsep ,
178     lastsep ,
179     rangesep ,
180     refpre ,
181     refpos ,
182     refpre-in ,
183     refpos-in ,
184 }
185 \seq_const_from_clist:Nn
186 \c__zrefclever_ref_options_necessarily_type_specific_seq
187 {
188     Name-sg ,
189     name-sg ,
190     Name-pl ,
191     name-pl ,
192     Name-sg-ab ,
193     name-sg-ab ,
194     Name-pl-ab ,
195     name-pl-ab ,
196 }
197 \seq_new:N \c__zrefclever_ref_options_type_specific_seq
198 \seq_gconcat:NNN \c__zrefclever_ref_options_type_specific_seq
199 \c__zrefclever_ref_options_possibly_type_specific_seq
200 \c__zrefclever_ref_options_necessarily_type_specific_seq
201 \seq_new:N \c__zrefclever_ref_options_not_type_specific_seq
202 \seq_gconcat:NNN \c__zrefclever_ref_options_not_type_specific_seq
203 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
204 \c__zrefclever_ref_options_possibly_type_specific_seq

```

### 4.3 Internationalization

```

205 \prop_new:N \g__zrefclever_language_aliases_prop
206
207 % {<base language>}
208 \NewDocumentCommand \zcDeclareLanguage { m }
209 {
210     \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#1}
211     {
212         \str_if_eq:eeTF {#1}
213         { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
214         { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
215         {
216             \msg_warning:nnxx { zref-clever } { alias-declared } {#1}
217             { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
218         }
219     }
220     { \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1} {#1} }
221 }
222
223 % {<alias>}{<base language>}
224 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
225 {

```

```

226 \tl_if_empty:nF {#2}
227 {
228   \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#2}
229   {
230     \exp_args:NnNx \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1}
231     { \prop_item:Nn \g__zrefclever_language_aliases_prop {#2} }
232   }
233   { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
234 }
235 }

```

```

\l__zrefclever_dict_type_tl
\l__zrefclever_dict_language_tl

```

```

236 \tl_new:N \l__zrefclever_dict_type_tl
237 \tl_new:N \l__zrefclever_dict_language_tl

```

(End definition for \l\_\_zrefclever\_dict\_type\_tl and \l\_\_zrefclever\_dict\_language\_tl.)

```

238 \seq_new:N \g__zrefclever_loaded_dictionaries_seq
239 \tl_new:N \l__zrefclever_dict_file_tl
240 \bool_new:N \l__zrefclever_load_dict_verbose_bool
241
242 % {<language>}
243 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
244 {
245   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
246   \l__zrefclever_dict_language_tl
247   {
248     \seq_if_in:NVF
249     \g__zrefclever_loaded_dictionaries_seq
250     \l__zrefclever_dict_language_tl
251     {
252       \exp_args:Nx \file_get:nnNTF
253       { zref-clever- \l__zrefclever_dict_language_tl .dict }
254       { \ExplSyntaxOn }
255       \l__zrefclever_dict_file_tl
256       {
257         \prop_if_exist:cF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _pro
258         { \prop_new:c { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
259         \tl_clear:N \l__zrefclever_dict_type_tl
260         \exp_args:NnV \keys_set:nn
261         { zref-clever / dictionary } \l__zrefclever_dict_file_tl
262         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
263         \l__zrefclever_dict_language_tl
264         \msg_note:nnx { zref-clever } { dict-loaded }
265         { \l__zrefclever_dict_language_tl }
266       }
267     }
268     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
269     {
270       \msg_warning:nnx { zref-clever } { dict-not-available }
271       { \l__zrefclever_dict_language_tl }
272     }
273   }
274 }
275 }

```

```

276     {
277         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
278         { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
279     }
280 }
281 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }
282
283 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
284 {
285     \group_begin:
286     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
287     \__zrefclever_provide_dictionary:n {#1}
288     \group_end:
289 }
290 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }
291
292 % {<key>}{<translation>}
293 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
294 {
295     \exp_args:Nnx \prop_gput_if_new:cnn
296     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
297     { type- \l__zrefclever_dict_type_tl - #1 } {#2}
298 }
299
300 % {<key>}{<translation>}
301 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
302 {
303     \prop_gput_if_new:cnn
304     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
305     { default- #1 } {#2}
306 }
307
308 % {<language>}{<type>}{<key>}{<translation>}
309 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
310 {
311     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
312     { type- #2 - #3 } {#4}
313 }
314 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
315
316 % {<language>}{<key>}{<translation>}
317 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
318 {
319     \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
320     { default- #2 } {#3}
321 }
322 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }
323 \keys_define:nn { zref-clever / dictionary }
324 {
325     type .code:n =
326     {
327         \tl_if_empty:nTF {#1}
328         { \tl_clear:N \l__zrefclever_dict_type_tl }
329         { \tl_set:Nn \l__zrefclever_dict_type_tl {#1} }

```

```

330     } ,
331 }
332 \seq_map_inline:Nn
333   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
334 {
335   \keys_define:nn { zref-clever / dictionary }
336   {
337     #1 .value_required:n = true ,
338     #1 .code:n =
339     {
340       \tl_if_empty:NTF \l__zrefclever_dict_type_tl
341       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
342       {
343         \msg_info:nnn { zref-clever }
344         { option-not-type-specific } {#1}
345       }
346     } ,
347   }
348 }
349 \seq_map_inline:Nn
350   \c__zrefclever_ref_options_possibly_type_specific_seq
351 {
352   \keys_define:nn { zref-clever / dictionary }
353   {
354     #1 .value_required:n = true ,
355     #1 .code:n =
356     {
357       \tl_if_empty:NTF \l__zrefclever_dict_type_tl
358       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
359       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
360     } ,
361   }
362 }
363 \seq_map_inline:Nn
364   \c__zrefclever_ref_options_necessarily_type_specific_seq
365 {
366   \keys_define:nn { zref-clever / dictionary }
367   {
368     #1 .value_required:n = true ,
369     #1 .code:n =
370     {
371       \tl_if_empty:NTF \l__zrefclever_dict_type_tl
372       {
373         \msg_info:nnn { zref-clever }
374         { option-only-type-specific } {#1}
375       }
376       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
377     } ,
378   }
379 }
380 % {<language>}{<type>}{<key>}<tl var to set>
381 \prg_new_protected_conditional:Npnn \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
382 {
383   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}

```

```

384 \l__zrefclever_dict_language_tl
385 {
386   \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
387   { type- #2 - #3 } #4
388   { \prg_return_true: }
389   { \prg_return_false: }
390 }
391 { \prg_return_false: }
392 }
393 \prg_generate_conditional_variant:Nnn \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }
394
395 % {<language>}{<key>}<tl var to set>
396 \prg_new_protected_conditional:Npnn \__zrefclever_get_default_transl:nnN #1#2#3 { F }
397 {
398   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
399   \l__zrefclever_dict_language_tl
400   {
401     \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
402     { default- #2 } #3
403     { \prg_return_true: }
404     { \prg_return_false: }
405   }
406   { \prg_return_false: }
407 }
408 \prg_generate_conditional_variant:Nnn \__zrefclever_get_default_transl:nnN { xnN } { F }
409
410 % {<key>}<tl var to set>
411 \prg_new_protected_conditional:Npnn \__zrefclever_get_fallback_transl:nN #1#2 { F }
412 {
413   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
414   { #1 } #2
415   { \prg_return_true: }
416   { \prg_return_false: }
417 }

```

All options retrieved with `\__zrefclever_get_option_with_transl:nN` must have their values set for ‘fallback’, even if to empty values, since this is what will be retrieved if `babel` or `polyglossia` is loaded and sets a language which `zref-clever` does not know. On the other hand, type-specific options are not looked for in ‘fallback’.

```

418 \prop_new:N \g__zrefclever_fallback_dict_prop
419 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
420 {
421   namesep = {\nobreakspace} ,
422   pairsep = {,~} ,
423   listsep = {,~} ,
424   lastsep = {,~} ,
425   tpairsep = {,~} ,
426   tlistsep = {,~} ,
427   tlastsep = {,~} ,
428   notesep = {~} ,
429   rangesep = {\textendash} ,
430   refpre = {} ,
431   refpos = {} ,
432   refpre-in = {} ,

```

```

433     refpos-in = {} ,
434 }

```

## 4.4 Options

### 4.4.1 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>}
435 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
436 {
437   \tl_if_empty:nTF {#3}
438     { \prop_remove:Nn #1 {#2} }
439     { \prop_put:Nnn #1 {#2} {#3} }
440 }

```

(End definition for `\_zrefclever_prop_put_non_empty:Nnn`.)

### 4.4.2 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`.

```

441 \prop_new:N \l__zrefclever_counter_type_prop
442 \keys_define:nn { zref-clever / label }
443 {
444   countertype .code:n =
445   {
446     \keyval_parse:nnn
447     {
448       \msg_warning:nnnn { zref-clever }
449       { key-requires-value } { countertype }
450     }
451     {
452       \__zrefclever_prop_put_non_empty:Nnn
453       \l__zrefclever_counter_type_prop
454     }
455     {#1}
456   } ,
457   countertype .value_required:n = true ,
458   countertype .initial:n =
459   {
460     subsection    = section ,
461     subsubsection = section ,
462     subparagraph  = paragraph ,
463     enumi         = item ,
464     enumii        = item ,
465     enumiii       = item ,
466     enumiv        = item ,
467   } ,
468 }

```

#### 4.4.3 counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `\__zrefclever_counter_reset_by:n` to populate the `zc@enclcnt` and `zc@enclval` properties, 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.

```
469 \seq_new:N \l__zrefclever_counter_resetters_seq
470 \keys_define:nn { zref-clever / label }
471 {
472   counterresetters .code:n =
473   {
474     \clist_map_inline:nn {#1}
475     {
476       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
477       {
478         \seq_put_right:Nn
479         \l__zrefclever_counter_resetters_seq {##1}
480       }
481     }
482   } ,
483   counterresetters .initial:n =
484   {
485     part ,
486     chapter ,
487     section ,
488     subsection ,
489     subsubsection ,
490     paragraph ,
491     subparagraph ,
492   },
493   typesort .value_required:n = true ,
494 }
```

#### 4.4.4 counterresetby option

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

```
495 \prop_new:N \l__zrefclever_counter_resetby_prop
496 \keys_define:nn { zref-clever / label }
497 {
498   counterresetby .code:n =
499   {
500     \keyval_parse:nnn
501     {
502       \msg_warning:nnn { zref-clever }
```

```

503         { key-requires-value } { counterresetby }
504     }
505     {
506         \__zrefclever_prop_put_non_empty:Nnn
507         \l__zrefclever_counter_resetby_prop
508     }
509     {#1}
510 } ,
511 counterresetby .value_required:n = true ,
512 counterresetby .initial:n =
513 {

```

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.

```

514     enumii = enumi ,
515     enumiii = enumii ,
516     enumiv = enumiii ,
517 } ,
518 }

```

#### 4.4.5 ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these two (or three) alternatives – `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 default counter. If more flexibility is granted, one thing *must* be handled at this point: the existence of the property itself, as far as `zref` is concerned. This because typesetting relies on the check `\zref@ifrefcontainsprop`, which *presumes* the property is defined and silently expands the *true* branch if it is not (see <https://github.com/ho-tex/zref/issues/13>, thanks Ulrike Fischer). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door.

```

519 \tl_new:N \l__zrefclever_ref_property_tl
520 \keys_define:nn { zref-clever / reference }
521 {
522     ref .choice: ,
523     ref / zc@thecnt .code:n =
524     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
525     ref / page .code:n =
526     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
527     ref / title .code:n =
528     {
529         \AddToHook { begindocument }
530         {
531             \@ifpackageloaded { zref-titleref }
532             { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
533             {
534                 \msg_warning:nn { zref-clever } { missing-zref-titleref }
535                 \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
536             }
537         }
538     }

```



```

538     } ,
539     ref .initial:n = zc@thecnt ,
540     ref .value_required:n = true ,
541     page .meta:n = { ref = page },
542     page .value_forbidden:n = true ,
543 }
544 \AddToHook { begindocument }
545 {
546   \@ifpackageloaded { zref-titleref }
547   {
548     \keys_define:nn { zref-clever / reference }
549     {
550       ref / title .code:n =
551       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
552     }
553   }
554   {
555     \keys_define:nn { zref-clever / reference }
556     {
557       ref / title .code:n =
558       {
559         \msg_warning:nn { zref-clever } { missing-zref-titleref }
560         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
561       }
562     }
563   }
564 }

```

#### 4.4.6 typeset option

```

565 \bool_new:N \l__zrefclever_typeset_ref_bool
566 \bool_new:N \l__zrefclever_typeset_name_bool
567 \keys_define:nn { zref-clever / reference }
568 {
569   typeset .choice: ,
570   typeset / both .code:n =
571   {
572     \bool_set_true:N \l__zrefclever_typeset_ref_bool
573     \bool_set_true:N \l__zrefclever_typeset_name_bool
574   } ,
575   typeset / ref .code:n =
576   {
577     \bool_set_true:N \l__zrefclever_typeset_ref_bool
578     \bool_set_false:N \l__zrefclever_typeset_name_bool
579   } ,
580   typeset / name .code:n =
581   {
582     \bool_set_false:N \l__zrefclever_typeset_ref_bool
583     \bool_set_true:N \l__zrefclever_typeset_name_bool
584   } ,
585   typeset .initial:n = both ,
586   typeset .value_required:n = true ,
587
588   noname .meta:n = { typeset = ref },

```

```

589     noname .value_forbidden:n = true ,
590 }

```

#### 4.4.7 sort option

```

591 \bool_new:N \l__zrefclever_typeset_sort_bool
592 \keys_define:nn { zref-clever / reference }
593 {
594     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
595     sort .initial:n = true ,
596     sort .default:n = true ,
597     nosort .meta:n = { sort = false },
598     nosort .value_forbidden:n = true ,
599 }

```

#### 4.4.8 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.

```

600 \seq_new:N \l__zrefclever_typesort_seq
601 \keys_define:nn { zref-clever / reference }
602 {
603     typesort .code:n =
604     {
605         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
606         \seq_reverse:N \l__zrefclever_typesort_seq
607     } ,
608     typesort .initial:n =
609     { part , chapter , section , paragraph },
610     typesort .value_required:n = true ,
611     notypesort .code:n =
612     { \seq_clear:N \l__zrefclever_typesort_seq } ,
613     notypesort .value_forbidden:n = true ,
614 }

```

#### 4.4.9 comp option

```

615 \bool_new:N \l__zrefclever_typeset_compress_bool
616 \keys_define:nn { zref-clever / reference }
617 {
618     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
619     comp .initial:n = true ,
620     comp .default:n = true ,
621     nocomp .meta:n = { comp = false },
622     nocomp .value_forbidden:n = true ,
623 }

```

#### 4.4.10 range option

```

624 \bool_new:N \l__zrefclever_typeset_range_bool
625 \keys_define:nn { zref-clever / reference }
626 {
627     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
628     range .initial:n = false ,

```

```

629     range .default:n = true ,
630 }

```

#### 4.4.11 hyperref option

```

631 \bool_new:N \l__zrefclever_use_hyperref_bool
632 \bool_new:N \l__zrefclever_warn_hyperref_bool
633 \keys_define:nn { zref-clever / reference }
634 {
635     hyperref .choice: ,
636     hyperref / auto .code:n =
637     {
638         \bool_set_true:N \l__zrefclever_use_hyperref_bool
639         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
640     } ,
641     hyperref / true .code:n =
642     {
643         \bool_set_true:N \l__zrefclever_use_hyperref_bool
644         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
645     } ,
646     hyperref / false .code:n =
647     {
648         \bool_set_false:N \l__zrefclever_use_hyperref_bool
649         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
650     } ,
651     hyperref .initial:n = auto ,
652     hyperref .default:n = auto
653 }
654 \AddToHook { begindocument }
655 {
656     \@ifpackageloaded { hyperref }
657     {
658         \bool_if:NT \l__zrefclever_use_hyperref_bool
659         { \RequirePackage { zref-hyperref } }
660     }
661     {
662         \bool_if:NT \l__zrefclever_warn_hyperref_bool
663         { \msg_warning:nn { zref-clever } { missing-hyperref } }
664         \bool_set_false:N \l__zrefclever_use_hyperref_bool
665     }
666     \keys_define:nn { zref-clever / reference }
667     {
668         hyperref .code:n =
669         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
670     }
671 }

```

#### 4.4.12 nameinlink option

```

672 \str_new:N \l__zrefclever_nameinlink_str
673 \keys_define:nn { zref-clever / reference }
674 {
675     nameinlink .choice: ,
676     nameinlink / true .code:n =
677     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
678     nameinlink / false .code:n =

```

```

679     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
680     nameinlink / single .code:n =
681     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
682     nameinlink / tsingle .code:n =
683     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
684     nameinlink .initial:n = tsingle ,
685     nameinlink .default:n = true ,
686 }

```

#### 4.4.13 cap and capfirst options

```

687 \bool_new:N \l__zrefclever_capitalize_bool
688 \bool_new:N \l__zrefclever_capitalize_first_bool
689 \keys_define:nn { zref-clever / reference }
690 {
691     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
692     cap .initial:n = false ,
693     cap .default:n = true ,
694     nocap .meta:n = { cap = false },
695     nocap .value_forbidden:n = true ,
696
697     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
698     capfirst .initial:n = false ,
699     capfirst .default:n = true ,
700
701     C .meta:n =
702     { capfirst = true , noabbrevfirst = true },
703     C .value_forbidden:n = true ,
704 }

```

#### 4.4.14 abbrev and noabbrevfirst options

```

705 \bool_new:N \l__zrefclever_abbrev_bool
706 \bool_new:N \l__zrefclever_noabbrev_first_bool
707 \keys_define:nn { zref-clever / reference }
708 {
709     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
710     abbrev .initial:n = false ,
711     abbrev .default:n = true ,
712     noabbrev .meta:n = { abbrev = false },
713     noabbrev .value_forbidden:n = true ,
714
715     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
716     noabbrevfirst .initial:n = false ,
717     noabbrevfirst .default:n = true ,
718 }

```

#### 4.4.15 lang option

`\l__zrefclever_current_language_tl` is an internal alias for babel’s `\language` or polyglossia’s `\mainbabelname`, 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 are set. 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`.

```

719 \tl_new:N \l__zrefclever_ref_language_tl
720 \tl_new:N \l__zrefclever_main_language_tl
721 \tl_new:N \l__zrefclever_current_language_tl
722 \AddToHook { begindocument }
723 {
724   \@ifpackageloaded { babel }
725   {
726     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
727     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
728   }
729   {
730     \@ifpackageloaded { polyglossia }
731     {
732       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
733       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
734     }
735     {
736       \tl_set:Nn \l__zrefclever_current_language_tl { english }
737       \tl_set:Nn \l__zrefclever_main_language_tl { english }
738     }
739   }

```

Provide default value for `\l__zrefclever_ref_language_tl` corresponding to option `main`, but do so outside of the `l3keys` machinery, so that we are able to distinguish when the user actually gave the option, in which case, the dictionary loading is done verbosely.

```

740   \tl_set:Nn \l__zrefclever_ref_language_tl { \l__zrefclever_main_language_tl }
741 }
742 \keys_define:nn { zref-clever / reference }
743 {
744   lang .code:n =
745   {
746     \AddToHook { begindocument }
747     {
748       \str_case:nnF {#1}

```

```

749         {
750             { main }
751             {
752                 \tl_set:Nn \l__zrefclever_ref_language_tl
753                 { \l__zrefclever_main_language_tl }
754                 \__zrefclever_provide_dictionary_verbose:x
755                 { \l__zrefclever_ref_language_tl }
756             }
757
758             { current }
759             {
760                 \tl_set:Nn \l__zrefclever_ref_language_tl
761                 { \l__zrefclever_current_language_tl }
762                 \__zrefclever_provide_dictionary_verbose:x
763                 { \l__zrefclever_ref_language_tl }
764             }
765         }
766         {
767             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
768             \__zrefclever_provide_dictionary_verbose:x
769             { \l__zrefclever_ref_language_tl }
770         }
771     }
772 } ,
773 lang .value_required:n = true ,
774 }
775 \AddToHook { begindocument / before }
776 {
777     \AddToHook { begindocument }
778     {

```

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.

```

779     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body.

```

780     \keys_define:nn { zref-clever / reference }
781     {
782         lang .code:n =
783         {
784             \str_case:nnF {#1}
785             {
786                 { main }
787                 {
788                     \tl_set:Nn \l__zrefclever_ref_language_tl
789                     { \l__zrefclever_main_language_tl }
790                     \__zrefclever_provide_dictionary_verbose:x
791                     { \l__zrefclever_ref_language_tl }
792                 }
793
794                 { current }
795                 {
796                     \tl_set:Nn \l__zrefclever_ref_language_tl
797                     { \l__zrefclever_current_language_tl }

```

```

798         \__zrefclever_provide_dictionary_verbose:x
799         { \l__zrefclever_ref_language_tl }
800     }
801 }
802 {
803     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
804     \__zrefclever_provide_dictionary_verbose:x
805     { \l__zrefclever_ref_language_tl }
806 }
807 } ,
808 lang .value_required:n = true ,
809 }
810 }
811 }

```

#### 4.4.16 font option

```

812 \tl_new:N \l__zrefclever_ref_typeset_font_tl
813 \keys_define:nn { zref-clever / reference }
814 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

#### 4.4.17 note option

```

815 \tl_new:N \l__zrefclever_zceref_note_tl
816 \keys_define:nn { zref-clever / reference }
817 {
818     note .tl_set:N = \l__zrefclever_zceref_note_tl ,
819     note .value_required:n = true ,
820 }

```

#### 4.4.18 check option

Integration with zref-check.

```

821 \bool_new:N \l__zrefclever_zrefcheck_available_bool
822 \bool_new:N \l__zrefclever_zceref_with_check_bool
823 \keys_define:nn { zref-clever / reference }
824 {
825     check .code:n =
826     { \msg_warning:nn { zref-clever } { check-document-only } } ,
827 }
828 \AddToHook { begindocument }
829 {
830     \@ifpackageloaded { zref-check }
831     {
832         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
833         \keys_define:nn { zref-clever / reference }
834         {
835             check .code:n =
836             {
837                 \bool_set_true:N \l__zrefclever_zceref_with_check_bool
838                 \keys_set:nn { zref-check / zcheck } {#1}
839             }
840         }
841     }
842 }

```

```

843     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
844     \keys_define:nn { zref-clever / reference }
845     {
846         check .code:n =
847             { \msg_warning:nn { zref-clever } { missing-zref-check } }
848     }
849 }
850 }

```

#### 4.4.19 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_option_with_transl:nN` and `\__zrefclever_get_option_plain: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 5.1.

```

851 \prop_new:N \l__zrefclever_ref_options_prop
852 \seq_map_inline:Nn
853   \c__zrefclever_ref_options_not_type_specific_seq
854   {
855     \keys_define:nn { zref-clever / reference }
856     {
857       #1 .default:V = \c_novalue_tl ,
858       #1 .code:n =
859       {
860         \tl_if_novalue:nTF {##1}
861         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
862         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
863       } ,
864     }
865   }

```

#### 4.5 \zcsetup

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 the `\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.

```

866 \keys_define:nn { }
867 {
868   zref-clever / zcsetup .inherit:n = zref-clever / label ,
869   zref-clever / zcsetup .inherit:n = zref-clever / reference ,
870 }

```



```

\zcsetup Provide \zcsetup.
871 \NewDocumentCommand \zcsetup { m }
872 { \keys_set:nn { zref-clever / zcsetup } {#1} }

(End definition for \zcsetup.)

```

## 4.6 Package options

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

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

## 5 Reference format

Formatting how the reference is to be typeset is, quite naturally, a big part of the user interface of `zref-clever`. In this area, we tried to balance “flexibility” and “user friendliness”. But the former does place a big toll overall, since there are indeed many places where tweaking may be desired, and the settings may depend on at least two important dimensions of variation: the reference type and the language. Combination of those necessarily makes for a large set of possibilities. Hence, the attempt here is to provide a rich set of “handles” for fine tuning the reference format but, at the same time, do not *require* detailed setup by the users, unless they really want it.

With that in mind, we have settled with an user interface for reference formatting which allows settings to be done in different scopes, with more or less overarching effects, and some precedence rules to regulate the relation of settings given in each of these scopes. There are four scopes in which reference formatting can be specified by the user, in the following precedence order: i) as general *options*; ii) as *type-specific options*; iii) as *language-specific and type-specific translations*; and iv) as *default translations* (that is, language-specific but not type-specific). These precedence rules are handled / enforced in `\__zrefclever_get_option_with_transl:nN` and `\__zrefclever_get_option_plain:nN`, which are the basic functions to retrieve proper values for reference format settings.

General “options” (i) can be given by the user in the optional argument of `\zceref`, but just as well in `\zcsetup` or as package options at load-time (see Section 4.4.19). “Type-specific options” (ii) are handled by `\zcRefTypeSetup`. “Language-specific translations”, be they “type-specific” (iii) or “default” (iv) have their user interface in `\zcDeclareTranslations`, and have their values populated by the package’s dictionaries.

Not all reference format specifications can be given in all of these scopes. Some of them can’t be type-specific, others must be type-specific, so the set available in each scope depends on the pertinence of the case.

The package itself places the default setup for reference formatting at low precedence levels, and the users can easily and conveniently override them as desired. Indeed, I expect most of the users’ needs to be normally achievable with the general options and type-specific options, since references will normally be typeset in a single language (the document’s main language) and, hence, multiple translations don’t need to be provided.

```

\l__zrefclever_setup_type_tl Store type and language in use in \zcRefTypeSetup and \zcDeclareTranslations.
\l__zrefclever_setup_language_tl
874 \tl_new:N \l__zrefclever_setup_type_tl
875 \tl_new:N \l__zrefclever_setup_language_tl

(End definition for \l__zrefclever_setup_type_tl and \l__zrefclever_setup_language_tl.)

```

## 5.1 \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 `\zcDeclareTranslations` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The  $\langle options \rangle$  should be given in the usual `key=val` format. The  $\langle type \rangle$  does not need to pre-exist, the property list variable to store the properties for the type gets created if need be.

```
\zcRefTypeSetup      \zcRefTypeSetup {\langle type \rangle} {\langle options \rangle}
876 \NewDocumentCommand \zcRefTypeSetup { m m }
877 {
878   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
879   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
880   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
881   \keys_set:nn { zref-clever / typesetup } {#2}
882 }
```

(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 4.4.19), 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>.

```
883 \seq_map_inline:Nn
884   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
885   {
886     \keys_define:nn { zref-clever / typesetup }
887     {
888       #1 .code:n =
889       {
890         \msg_warning:nnn { zref-clever }
891         { option-not-type-specific } {#1}
892       } ,
893     }
894   }
895 \seq_map_inline:Nn
896   \c__zrefclever_ref_options_type_specific_seq
897   {
898     \keys_define:nn { zref-clever / typesetup }
899     {
900       #1 .default:V = \c_novalue_tl ,
```

```

901     #1 .code:n =
902     {
903         \tl_if_novalue:nTF {##1}
904         {
905             \prop_remove:cn
906             {
907                 l__zrefclever_type_
908                 \l__zrefclever_setup_type_tl _options_prop
909             }
910             {#1}
911         }
912         {
913             \prop_put:cnn
914             {
915                 l__zrefclever_type_
916                 \l__zrefclever_setup_type_tl _options_prop
917             }
918             {#1} {##1}
919         }
920     } ,
921 }
922 }

```

## 5.2 \zcDeclareTranslations

`\zcDeclareTranslations` 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 `\zcDeclareTranslations`, 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.

```

\zcDeclareTranslations      \zcDeclareTranslations {<language>} {<options>}
923 \NewDocumentCommand \zcDeclareTranslations { m m }
924 {
925     \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
926     \l__zrefclever_setup_language_tl
927     {
928         \tl_clear:N \l__zrefclever_setup_type_tl
929         \keys_set:nn { zref-clever / translations } {#2}
930     }
931     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
932 }

```

(End definition for `\zcDeclareTranslations`.)

```

933 \keys_define:nn { zref-clever / translations }
934 {
935     type .code:n =
936     {
937         \tl_if_empty:nTF {#1}
938         { \tl_clear:N \l__zrefclever_setup_type_tl }
939         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }

```

```

940     } ,
941 }
942 \seq_map_inline:Nn
943 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
944 {
945   \keys_define:nn { zref-clever / translations }
946   {
947     #1 .value_required:n = true ,
948     #1 .code:n =
949     {
950       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
951       {
952         \__zrefclever_declare_default_transl:Vnn
953         \l__zrefclever_setup_language_tl
954         {#1} {##1}
955       }
956       {
957         \msg_warning:nnn { zref-clever }
958         { option-not-type-specific } {#1}
959       }
960     } ,
961   }
962 }
963 \seq_map_inline:Nn
964 \c__zrefclever_ref_options_possibly_type_specific_seq
965 {
966   \keys_define:nn { zref-clever / translations }
967   {
968     #1 .value_required:n = true ,
969     #1 .code:n =
970     {
971       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
972       {
973         \__zrefclever_declare_default_transl:Vnn
974         \l__zrefclever_setup_language_tl
975         {#1} {##1}
976       }
977       {
978         \__zrefclever_declare_type_transl:VVnn
979         \l__zrefclever_setup_language_tl
980         \l__zrefclever_setup_type_tl
981         {#1} {##1}
982       }
983     } ,
984   }
985 }
986 \seq_map_inline:Nn
987 \c__zrefclever_ref_options_necessarily_type_specific_seq
988 {
989   \keys_define:nn { zref-clever / translations }
990   {
991     #1 .value_required:n = true ,
992     #1 .code:n =

```

```

993         {
994             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
995             {
996                 \msg_warning:nnn { zref-clever }
997                 { option-only-type-specific } {#1}
998             }
999             {
1000                 \__zrefclever_declare_type_transl:VWnn
1001                 \l__zrefclever_setup_language_tl
1002                 \l__zrefclever_setup_type_tl
1003                 {#1} {##1}
1004             }
1005         } ,
1006     }
1007 }

```

## 6 User interface

### 6.1 \zcref

```

\zcref          \zcref<*>[<options>]{<labels>}
1008 \NewDocumentCommand \zcref { s O { } m }
1009 { \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>}
1010 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1011 {
1012     \group_begin:

```

Set options.

```

1013     \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1014     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1015     \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.

```

1016     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with zref-check.

```

1017     \bool_lazy_and:nnT
1018     { \l__zrefclever_zrefcheck_available_bool }
1019     { \l__zrefclever_zcref_with_check_bool }
1020     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1021     \bool_lazy_or:nnT
1022     { \l__zrefclever_typeset_sort_bool }
1023     { \l__zrefclever_typeset_range_bool }
1024     { \l__zrefclever_sort_labels: }

```

Typeset the references.

```

1025     \l__zrefclever_typeset_refs:

```

Typeset note.

```

1026     \l__zrefclever_notesep_tl
1027     \l__zrefclever_zcref_note_tl

```

Integration with zref-check.

```

1028     \bool_lazy_and:nnT
1029     { \l__zrefclever_zrefcheck_available_bool }
1030     { \l__zrefclever_zcref_with_check_bool }
1031     {
1032         \zrefcheck_zcref_end_label_maybe:
1033         \zrefcheck_zcref_run_checks_on_labels:n
1034         { \l__zrefclever_zcref_labels_seq }
1035     }
1036     \group_end:
1037 }

```

(End definition for \l\_\_zrefclever\_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1038 \seq_new:N \l__zrefclever_zcref_labels_seq
1039 \bool_new:N \l__zrefclever_link_star_bool

```

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

## 6.2 \zcpageref

```

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

1040 \NewDocumentCommand \zcpageref { s O { } m }
1041 {
1042     \IfBooleanTF {#1}
1043     { \zcref*[#2, ref = page] {#3} }
1044     { \zcref [ #2, ref = page] {#3} }
1045 }

```

(End definition for \zcpageref.)

## 7 Sorting

Aux variables, for use in sorting and typesetting. I could probably let go some of them in favor of tmpa/tmpb, but they do improve code readability.

```

\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclcnt_b_tl
\l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl

1046 \tl_new:N \l__zrefclever_label_a_tl
1047 \tl_new:N \l__zrefclever_label_b_tl
1048 \tl_new:N \l__zrefclever_label_type_a_tl
1049 \tl_new:N \l__zrefclever_label_type_b_tl
1050 \tl_new:N \l__zrefclever_label_enclcnt_a_tl

```

```

1051 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1052 \tl_new:N \l__zrefclever_label_enclval_a_tl
1053 \tl_new:N \l__zrefclever_label_enclval_b_tl

(End definition for \l__zrefclever_label_a_tl and others.)

1054 \int_new:N \l__zrefclever_sort_prior_a_int
1055 \int_new:N \l__zrefclever_sort_prior_b_int

```

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

```

1056 \bool_new:N \l__zrefclever_sort_decided_bool

(End definition for \l__zrefclever_sort_decided_bool.)
Variant not provided by the kernel.

1057 \cs_generate_variant:Nn \tl_reverse_items:n { V }

```

`\__zrefclever_label_type_put_new_right:n` Auxiliary function used to store “new” label types (in order) as the sorting proceeds. It is expected to be run inside `\__zrefclever_sort_labels:`, and stores new types in `\l__zrefclever_label_types_seq`.

```

\__zrefclever_label_type_put_new_right:n {<label>}

1058 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1059 {
1060   \tl_set:Nx \l__zrefclever_label_type_a_tl
1061   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1062   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1063   {
1064     \seq_if_in:NVF
1065     \l__zrefclever_label_types_seq
1066     \l__zrefclever_label_type_a_tl
1067     {
1068       \seq_put_right:NV \l__zrefclever_label_types_seq
1069       \l__zrefclever_label_type_a_tl
1070     }
1071   }
1072 }

(End definition for \__zrefclever_label_type_put_new_right:n.)

```

`\l__zrefclever_label_types_seq` Stores the order in which reference types appear in the label list supplied by the user in `\zcref`. This order is required as a “last resort” sort criterion between the reference types, for use in `\__zrefclever_sort_default:nn`.

```

1073 \seq_new:N \l__zrefclever_label_types_seq

(End definition for \l__zrefclever_label_types_seq.)

```

`\__zrefclever_sort_labels:` The main sorting function. It does not receive arguments, but it is expected to be run inside `\__zrefclever_zcref:nnnn` where a number of environment variables are to be set appropriately. In particular, `\l__zrefclever_zcref_labels_seq` should contain the labels received as argument to `\zcref`, and the function performs its task by sorting this variable.

```

1074 \cs_new_protected:Npn \__zrefclever_sort_labels:
1075 {

```

Store label types sequence.

```

1076 \seq_clear:N \l__zrefclever_label_types_seq
1077 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1078 {
1079   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1080   \__zrefclever_label_type_put_new_right:n
1081 }

```

Sort.

```

1082 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1083 {
1084   \zref@ifrefundefined {##1}
1085   {
1086     \zref@ifrefundefined {##2}
1087     {
1088       % Neither label is defined.
1089       \sort_return_same:
1090     }
1091     {
1092       % The second label is defined, but the first isn't, leave the
1093       % undefined first (to be more visible).
1094       \sort_return_same:
1095     }
1096   }
1097   {
1098     \zref@ifrefundefined {##2}
1099     {
1100       % The first label is defined, but the second isn't, bring the
1101       % second forward.
1102       \sort_return_swapped:
1103     }
1104     {
1105       % The interesting case: both labels are defined. The
1106       % reference to the "default" property/counter or to the page
1107       % are quite different from our perspective, they rely on
1108       % different fields and even use different information for
1109       % sorting, so we branch them here to specialized functions.
1110       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1111       { \__zrefclever_sort_page:nn {##1} {##2} }
1112       { \__zrefclever_sort_default:nn {##1} {##2} }
1113     }
1114   }
1115 }
1116 }

```

(End definition for \\_\_zrefclever\_sort\_labels:.)

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

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



```

1117 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1118 {
1119   \tl_set:Nx \l__zrefclever_label_type_a_tl
1120   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1121   \tl_set:Nx \l__zrefclever_label_type_b_tl
1122   { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1123
1124   \bool_if:nTF
1125   {
1126     % The second label has a type, but the first doesn't, leave the
1127     % undefined first (to be more visible).
1128     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1129     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1130   }
1131   { \sort_return_same: }
1132   {
1133     \bool_if:nTF
1134     {
1135       % The first label has a type, but the second doesn't, bring the
1136       % second forward.
1137       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1138       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1139     }
1140     { \sort_return_swapped: }
1141     {
1142       \bool_if:nTF
1143       {
1144         % The interesting case: both labels have a type...
1145         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1146         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1147       }
1148       {
1149         % Here we send this to a couple of auxiliary functions for no
1150         % other reason than to keep this long function a little less
1151         % unreadable.
1152         \tl_if_eq:NNTF
1153         \l__zrefclever_label_type_a_tl
1154         \l__zrefclever_label_type_b_tl
1155         {
1156           % ...and it's the same type.
1157           \__zrefclever_sort_default_same_type:nn {#1} {#2}
1158         }
1159         {
1160           % ...and they are different types.
1161           \__zrefclever_sort_default_different_types:nn {#1} {#2}
1162         }
1163       }
1164     }
1165     {
1166       % Neither of the labels has a type. We can't do much of
1167       % meaningful here, but if it's the same counter, compare it.
1168       \exp_args:Nxx \tl_if_eq:nnTF
1169       { \zref@extractdefault {#1} { counter } { } }
1170       { \zref@extractdefault {#2} { counter } { } }
1171       {

```

```

1171         \int_compare:nNnTF
1172         { \zref@extractdefault {#1} { zc@cntval } {-1} }
1173         >
1174         { \zref@extractdefault {#2} { zc@cntval } {-1} }
1175         { \sort_return_swapped: }
1176         { \sort_return_same: }
1177     }
1178     { \sort_return_same: }
1179 }
1180 }
1181 }
1182 }

```

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

\\_zrefclever\_sort\_default\_same\_type:nn

```

1183 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1184 {
1185     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1186     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1187     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1188     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1189     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1190     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1191     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1192     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1193     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1194     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1195     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1196     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1197     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1198     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1199     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1200     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1201
1202     \bool_set_false:N \l__zrefclever_sort_decided_bool
1203     % CHECK should I replace the tmp variables here?
1204     \tl_clear:N \l_tmpa_tl
1205     \tl_clear:N \l_tmpb_tl
1206     \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1207     {
1208         \tl_set:Nx \l_tmpa_tl
1209         { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1210         \tl_set:Nx \l_tmpb_tl
1211         { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1212
1213         \bool_if:nTF
1214         {
1215             % Both are empty, meaning: neither labels have any (further)
1216             % ‘‘enclosing counters’’ (left).
1217             \tl_if_empty_p:V \l_tmpa_tl &&
1218             \tl_if_empty_p:V \l_tmpb_tl
1219         }
1220         {

```

```

1221 \exp_args:Nxx \tl_if_eq:nnTF
1222 { \zref@extractdefault {#1} { counter } { } }
1223 { \zref@extractdefault {#2} { counter } { } }
1224 {
1225   \bool_set_true:N \l__zrefclever_sort_decided_bool
1226   \int_compare:nNnTF
1227     { \zref@extractdefault {#1} { zc@cntval } {-1} }
1228     >
1229     { \zref@extractdefault {#2} { zc@cntval } {-1} }
1230     { \sort_return_swapped: }
1231     { \sort_return_same: }
1232 }
1233 {
1234   \msg_warning:nnnn { zref-clever }
1235   { counters-not-nested } {#1} {#2}
1236   \bool_set_true:N \l__zrefclever_sort_decided_bool
1237   \sort_return_same:
1238 }
1239 }
1240 {
1241   \bool_if:nTF
1242   {
1243     % 'a' is empty (and 'b' is not), meaning: 'b' is (possibly)
1244     % nested in 'a'.
1245     \tl_if_empty_p:V \l_tmpa_tl
1246   }
1247   {
1248     \tl_set:Nx \l_tmpa_tl
1249     { {\zref@extractdefault {#1} { counter } { } } }
1250     \exp_args:NNx \tl_if_in:NnTF
1251     \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1252     {
1253       \bool_set_true:N \l__zrefclever_sort_decided_bool
1254       \sort_return_same:
1255     }
1256     {
1257       \msg_warning:nnnn { zref-clever }
1258       { counters-not-nested } {#1} {#2}
1259       \bool_set_true:N \l__zrefclever_sort_decided_bool
1260       \sort_return_same:
1261     }
1262   }
1263 }
1264 \bool_if:nTF
1265 {
1266   % 'b' is empty (and 'a' is not), meaning: 'a' is
1267   % (possibly) nested in 'b'.
1268   \tl_if_empty_p:V \l_tmpb_tl
1269 }
1270 {
1271   \tl_set:Nx \l_tmpb_tl
1272   { {\zref@extractdefault {#2} { counter } { } } }
1273   \exp_args:NNx \tl_if_in:NnTF
1274   \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }

```

```

1275 {
1276   \bool_set_true:N \l__zrefclever_sort_decided_bool
1277   \sort_return_swapped:
1278 }
1279 {
1280   \msg_warning:nnnn { zref-clever }
1281   { counters-not-nested } {#1} {#2}
1282   \bool_set_true:N \l__zrefclever_sort_decided_bool
1283   \sort_return_same:
1284 }
1285 }
1286 {
1287   % Neither is empty, meaning: we can (possibly) compare the
1288   % values of the current enclosing counter in the loop, if
1289   % they are equal, we are still in the loop, if they are
1290   % not, a sorting decision can be made directly.
1291   \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
1292   {
1293     \int_compare:nNnTF
1294       { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1295       =
1296       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1297       {
1298         \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1299           { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1300         \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1301           { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1302         \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1303           { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1304         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1305           { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1306       }
1307       {
1308         \bool_set_true:N \l__zrefclever_sort_decided_bool
1309         \int_compare:nNnTF
1310           { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1311           >
1312           { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1313           { \sort_return_swapped: }
1314           { \sort_return_same: }
1315       }
1316   }
1317   {
1318     \msg_warning:nnnn { zref-clever }
1319     { counters-not-nested } {#1} {#2}
1320     \bool_set_true:N \l__zrefclever_sort_decided_bool
1321     \sort_return_same:
1322   }
1323 }
1324 }
1325 }
1326 }
1327 }

```

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

\_zrefclever\_sort\_default\_different\_types:nn

```

1328 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1329 {
1330   \int_zero:N \l__zrefclever_sort_prior_a_int
1331   \int_zero:N \l__zrefclever_sort_prior_b_int
1332   % \cs{l__zrefclever_typesort_seq} was stored in reverse sequence, and
1333   % we compute the sort priorities in the negative range, so that we can
1334   % implicitly rely on '0' being the "last value".
1335   \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1336   {
1337     \tl_if_eq:nnTF {##2} {{othertypes}}
1338     {
1339       \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1340       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1341       \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1342       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1343     }
1344     {
1345       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1346       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1347       {
1348         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1349         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1350       }
1351     }
1352   }
1353   \bool_if:nTF
1354   {
1355     \int_compare_p:nNn
1356     { \l__zrefclever_sort_prior_a_int } <
1357     { \l__zrefclever_sort_prior_b_int }
1358   }
1359   { \sort_return_same: }
1360   {
1361     \bool_if:nTF
1362     {
1363       \int_compare_p:nNn
1364       { \l__zrefclever_sort_prior_a_int } >
1365       { \l__zrefclever_sort_prior_b_int }
1366     }
1367     { \sort_return_swapped: }
1368     {
1369       % Sort priorities are equal for different types: the type that
1370       % occurs first in 'labels', as given by the user, is kept (or
1371       % brought) forward.
1372       \seq_map_inline:Nn \l__zrefclever_label_types_seq
1373       {
1374         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1375         { \seq_map_break:n { \sort_return_same: } }
1376         {
1377           \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1378           { \seq_map_break:n { \sort_return_swapped: } }
1379         }
1380       }
1381     }
1382   }

```

```

1381     }
1382   }
1383 }

```

(End definition for `\_zrefclever_sort_default_different_types:nn`.)

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

```

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

1384 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1385 {
1386   \int_compare:nNnTF
1387     { \zref@extractdefault {#1} { abspage } {-1} }
1388     >
1389     { \zref@extractdefault {#2} { abspage } {-1} }
1390     { \sort_return_swapped: }
1391     { \sort_return_same:   }
1392 }

```

(End definition for `\_zrefclever_sort_page:nn`.)

## 8 Typesetting

About possible alternatives to signal compression inhibition for individual labels, see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be to receive an optional argument with the label(s) not to be compressed. This would be a repetition, but would keep the syntax “clean”. All in all, and rethinking this here, probably the best is simply to not allow individual inhibition of compression. We can already control compression of each individual call of `\zcref` with existing options, this should be enough. I don’t think the small extra flexibility this would grant is worth the syntax disruption it entails. Anyway, I have kept a “handle” to deal with this in case the need arises, in the form of `\l__zrefclever_range_inhibit_next_bool`, which is currently no-op, but is in place.

### Variables

`\l__zrefclever_typeset_last_bool` Auxiliary variables for `\_zrefclever_typeset_refs:`. `\l__zrefclever_typeset_last_bool` signals if the label list is over so that we can leave the loop. `\l__zrefclever_last_of_type_bool` signals if we are processing the last label of the current reference type.

```

1393 \bool_new:N \l__zrefclever_typeset_last_bool
1394 \bool_new:N \l__zrefclever_last_of_type_bool

```

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

```

\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl

```

Auxiliary variables for `\__zrefclever_typeset_refs:`. They store, respectively the “previous” and the “current” reference type information while they are being processed, since we cannot typeset them directly, given we can only know certain things when the (next) type list is over. The “queue” stores all references but the first of the type, and they are stored ready to be typeset. The “first\_label” stores the *label* of the first reference for the type, because the name can only be determined at the end, and its (potential) hyperlink must be handled at that point.

```

1395 \seq_new:N \l__zrefclever_typeset_labels_seq
1396 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1397 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1398 \tl_new:N \l__zrefclever_type_first_label_tl
1399 \tl_new:N \l__zrefclever_type_first_label_type_tl

```

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

```

\l_zrefclever_label_count_int
\l_zrefclever_type_count_int

```

Main counters for `\__zrefclever_typeset_refs:`. They track the state of the parsing of the labels list. `\l__zrefclever_label_count_int` is stepped for every reference/label in the list, and reset at the start of a new type. `\l__zrefclever_type_count_int` is stepped at every reference type change.

```

1400 \int_new:N \l__zrefclever_label_count_int
1401 \int_new:N \l__zrefclever_type_count_int

```

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

```

\l_zrefclever_range_count_int
\l_zrefclever_range_same_count_int
\l_zrefclever_range_beg_label_tl
\l_zrefclever_next_maybe_range_bool
\l_zrefclever_next_is_same_bool
\l_zrefclever_range_inhibit_next_bool

```

Range related auxiliary variables for `\__zrefclever_typeset_refs:`. `\l__zrefclever_range_count_int` counts how many references/labels are in the current ongoing range. `\l__zrefclever_range_same_count_int` counts how many of the references in the current ongoing range are repeated ones. `\l__zrefclever_range_beg_label_tl` stores the label of the reference that starts a range. `\l__zrefclever_next_maybe_range_bool` signals whether the next element is in sequence to the current one. `\l__zrefclever_next_is_same_bool` signals whether the next element repeats the current one. `\l__zrefclever_range_inhibit_next_bool` allows to control/track compression inhibition of the next label.

```

1402 \int_new:N \l__zrefclever_range_count_int
1403 \int_new:N \l__zrefclever_range_same_count_int
1404 \tl_new:N \l__zrefclever_range_beg_label_tl
1405 \bool_new:N \l__zrefclever_next_maybe_range_bool
1406 \bool_new:N \l__zrefclever_next_is_same_bool
1407 \bool_new:N \l__zrefclever_range_inhibit_next_bool

```

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

Aux variables for `\__zrefclever_typeset_refs:`. Store separators and `refpre/pos` options.

```

1408 \tl_new:N \l__zrefclever_namefont_tl
1409 \tl_new:N \l__zrefclever_reffont_out_tl
1410 \tl_new:N \l__zrefclever_reffont_in_tl
1411
1412 \tl_new:N \l__zrefclever_namesep_tl
1413 \tl_new:N \l__zrefclever_rangesep_tl
1414 \tl_new:N \l__zrefclever_pairsep_tl
1415 \tl_new:N \l__zrefclever_listsep_tl
1416 \tl_new:N \l__zrefclever_lastsep_tl

```

```

1417 \tl_new:N \l__zrefclever_tpairsep_tl
1418 \tl_new:N \l__zrefclever_tlistsep_tl
1419 \tl_new:N \l__zrefclever_tlastsep_tl
1420 \tl_new:N \l__zrefclever_notesep_tl
1421 \tl_new:N \l__zrefclever_refpre_out_tl
1422 \tl_new:N \l__zrefclever_refpos_out_tl
1423 \tl_new:N \l__zrefclever_refpre_in_tl
1424 \tl_new:N \l__zrefclever_refpos_in_tl

```

(End definition for .)

\l\_\_zrefclever\_type\_name\_tl Auxiliary variables for \\_\_zrefclever\_get\_ref\_first: and \\_\_zrefclever\_type\_name\_setup:.

```

\l__zrefclever_name_in_link_bool
\l__zrefclever_name_format_tl
\l__zrefclever_name_format_fallback_tl
1425 \tl_new:N \l__zrefclever_type_name_tl
1426 \bool_new:N \l__zrefclever_name_in_link_bool
1427 \tl_new:N \l__zrefclever_name_format_tl
1428 \tl_new:N \l__zrefclever_name_format_fallback_tl

```

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

## Main functions

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

```

1429 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1430 {
1431   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zceref_labels_seq
1432   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1433   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1434   \tl_clear:N \l__zrefclever_type_first_label_tl
1435   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1436   \tl_clear:N \l__zrefclever_range_beg_label_tl
1437   \int_zero:N \l__zrefclever_label_count_int
1438   \int_zero:N \l__zrefclever_type_count_int
1439   \int_zero:N \l__zrefclever_range_count_int
1440   \int_zero:N \l__zrefclever_range_same_count_int
1441
1442   % Get not-type-specific separators and refpre/pos options.
1443   \__zrefclever_get_option_with_transl:nN {tpairsep} \l__zrefclever_tpairsep_tl
1444   \__zrefclever_get_option_with_transl:nN {tlistsep} \l__zrefclever_tlistsep_tl
1445   \__zrefclever_get_option_with_transl:nN {tlastsep} \l__zrefclever_tlastsep_tl
1446   \__zrefclever_get_option_with_transl:nN {notesep} \l__zrefclever_notesep_tl
1447
1448   % Set the font option for this zceref call.
1449   \l__zrefclever_ref_typeset_font_tl
1450
1451   % Loop over the label list in sequence.
1452   \bool_set_false:N \l__zrefclever_typeset_last_bool
1453   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1454   {
1455     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl
1456     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1457     {
1458       \tl_clear:N \l__zrefclever_label_b_tl
1459       \bool_set_true:N \l__zrefclever_typeset_last_bool

```



```

1460     }
1461     { \seq_get_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_b_tl }
1462
1463 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1464 {
1465     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1466     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1467 }
1468 {
1469     \tl_set:Nx \l__zrefclever_label_type_a_tl
1470     {
1471         \zref@extractdefault
1472         { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1473     }
1474     \tl_set:Nx \l__zrefclever_label_type_b_tl
1475     {
1476         \zref@extractdefault
1477         { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1478     }
1479 }
1480
1481 % First, we establish whether the ‘current label’ (i.e. ‘a’) is the
1482 % last one of its type. This can happen because the ‘next label’
1483 % (i.e. ‘b’) is of a different type (or different definition status),
1484 % or because we are at the end of the list.
1485 \bool_if:NTF \l__zrefclever_typeset_last_bool
1486 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1487 {
1488     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1489     {
1490         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1491         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1492         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1493     }
1494     {
1495         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1496         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1497         {
1498             % Neither is undefined, we must check the types.
1499             \bool_if:nTF
1500             % Both empty: same ‘type’.
1501             {
1502                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1503                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1504             }
1505             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1506             {
1507                 \bool_if:nTF
1508                 % Neither empty: compare types.
1509                 {
1510                     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1511                     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1512                 }
1513                 {

```

```

1514         \tl_if_eq:NNTF
1515         \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1516         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1517         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1518     }
1519     % One empty, the other not: different ‘types’.
1520     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1521 }
1522 }
1523 }
1524 }
1525
1526 % Handle warnings in case of reference or type undefined.
1527 \zref@refused { \l__zrefclever_label_a_tl }
1528 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1529 {}
1530 {
1531     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1532     {
1533         \msg_warning:nxx { zref-clever } { missing-type }
1534         { \l__zrefclever_label_a_tl }
1535     }
1536 }
1537
1538 % Get type-specific separators, refpre/pos and font options, once per
1539 % type.
1540 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1541 {
1542     \__zrefclever_get_option_plain:nN {namefont}          \l__zrefclever_namefont_tl
1543     \__zrefclever_get_option_plain:nN {reffont}          \l__zrefclever_reffont_out_tl
1544     \__zrefclever_get_option_plain:nN {reffont-in}       \l__zrefclever_reffont_in_tl
1545     \__zrefclever_get_option_with_transl:nN {namesep}     \l__zrefclever_namesep_tl
1546     \__zrefclever_get_option_with_transl:nN {rangesep}   \l__zrefclever_rangesep_tl
1547     \__zrefclever_get_option_with_transl:nN {pairsep}    \l__zrefclever_pairsep_tl
1548     \__zrefclever_get_option_with_transl:nN {listsep}    \l__zrefclever_listsep_tl
1549     \__zrefclever_get_option_with_transl:nN {lastsep}    \l__zrefclever_lastsep_tl
1550     \__zrefclever_get_option_with_transl:nN {refpre}     \l__zrefclever_refpre_out_tl
1551     \__zrefclever_get_option_with_transl:nN {refpos}     \l__zrefclever_refpos_out_tl
1552     \__zrefclever_get_option_with_transl:nN {refpre-in}  \l__zrefclever_refpre_in_tl
1553     \__zrefclever_get_option_with_transl:nN {refpos-in}  \l__zrefclever_refpos_in_tl
1554 }
1555
1556 % Here we send this to a couple of auxiliary functions for no other
1557 % reason than to keep this long function a little less unreadable.
1558 \bool_if:NTF \l__zrefclever_last_of_type_bool
1559 {
1560     % There exists no next label of the same type as the current.
1561     \__zrefclever_typeset_refs_aux_last_of_type:
1562 }
1563 {
1564     % There exists a next label of the same type as the current.
1565     \__zrefclever_typeset_refs_aux_not_last_of_type:
1566 }
1567 }

```

```

1568 }

(End definition for \_zrefclever_typeset_refs:.)

\_zrefclever_typeset_refs_aux_last_of_type: Handles typesetting of when the current label is the last of its type.
1569 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_last_of_type:
1570 {
1571   % Process the current label to the current queue.
1572   \int_case:nnF { \l__zrefclever_label_count_int }
1573   {
1574     % It is the last label of its type, but also the first one, and that's
1575     % what matters here: just store it.
1576     { 0 }
1577     {
1578       \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1579       \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1580     }
1581
1582     % The last is the second: we have a pair (if not repeated).
1583     { 1 }
1584     {
1585       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1586       {
1587         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1588         {
1589           \exp_not:V \l__zrefclever_pairsep_tl
1590           \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1591         }
1592       }
1593     }
1594   }
1595   % If neither the first, nor the second: we have the last label
1596   % on the current type list (if not repeated).
1597   {
1598     \int_case:nnF { \l__zrefclever_range_count_int }
1599     {
1600       % There was no range going on.
1601       {0}
1602       {
1603         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1604         {
1605           \exp_not:V \l__zrefclever_lastsep_tl
1606           \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1607         }
1608       }
1609       % Last in the range is also the second in it.
1610       {1}
1611       {
1612         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1613         {
1614           % We know 'range_beg_label' is not empty, since this is the
1615           % second element in the range, but the third or more in the
1616           % type list.
1617           \exp_not:V \l__zrefclever_listsep_tl

```

```

1618         \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1619         \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1620         {
1621             \exp_not:V \l__zrefclever_lastsep_tl
1622             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1623         }
1624     }
1625 }
1626 }
1627 % Last in the range is third or more in it.
1628 {
1629     \int_case:nnF
1630     { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1631     {
1632         % Repetition, not a range.
1633         {0}
1634         {
1635             % If 'range_beg_label' is empty, it means it was also the
1636             % first of the type, and hence was already handled.
1637             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1638             {
1639                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1640                 {
1641                     \exp_not:V \l__zrefclever_lastsep_tl
1642                     \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1643                 }
1644             }
1645         }
1646         % A 'range', but with no skipped value, treat as list.
1647         {1}
1648         {
1649             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1650             {
1651                 % Ditto.
1652                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1653                 {
1654                     \exp_not:V \l__zrefclever_listsep_tl
1655                     \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1656                 }
1657                 \exp_not:V \l__zrefclever_lastsep_tl
1658                 \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1659             }
1660         }
1661     }
1662 }
1663 % An actual range.
1664 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1665 {
1666     % Ditto.
1667     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1668     {
1669         \exp_not:V \l__zrefclever_lastsep_tl
1670         \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1671     }

```

```

1672         \exp_not:V \l__zrefclever_rangesep_tl
1673         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1674     }
1675 }
1676 }
1677 }
1678
1679 % Handle ‘‘range’’ option. The idea is simple: if the queue is not empty,
1680 % we replace it with the end of the range (or pair). We can still
1681 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1682 % be processing the last label of its type at this point.
1683 \bool_if:NT \l__zrefclever_typeset_range_bool
1684 {
1685     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1686     {
1687         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1688         { }
1689         {
1690             \msg_warning:nxx { zref-clever } { single-element-range }
1691             { \l__zrefclever_type_first_label_type_tl }
1692         }
1693     }
1694     {
1695         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1696         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1697         { }
1698         {
1699             \__zrefclever_labels_in_sequence:nn
1700             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
1701         }
1702         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1703         {
1704             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1705             { \exp_not:V \l__zrefclever_pairsep_tl }
1706             { \exp_not:V \l__zrefclever_rangesep_tl }
1707             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1708         }
1709     }
1710 }
1711
1712 % Now that the type is finished, we can add the name and the first ref to
1713 % the queue. Or, if ‘‘typset’’ option is not ‘‘both’’, handle it here
1714 % too.
1715 \__zrefclever_type_name_setup:
1716 \bool_if:NTF
1717 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1718 {
1719     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1720     { \__zrefclever_get_ref_first: }
1721 }
1722 {
1723     \bool_if:NTF
1724     { \l__zrefclever_typeset_ref_bool }
1725     {

```

```

1726 \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1727 { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1728 }
1729 {
1730 \bool_if:NTF
1731 { \l__zrefclever_typeset_name_bool }
1732 {
1733 \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1734 {
1735 \bool_if:NTF \l__zrefclever_name_in_link_bool
1736 {
1737 \exp_not:N \group_begin:
1738 \exp_not:V \l__zrefclever_namefont_tl
1739 % It's two '@s', but escaped for DocStrip.
1740 \exp_not:N \hyper@@link
1741 {
1742 \zref@ifrefcontainsprop
1743 { \l__zrefclever_type_first_label_tl } { urluse }
1744 {
1745 \zref@extractdefault
1746 { \l__zrefclever_type_first_label_tl }
1747 { urluse } {}
1748 }
1749 {
1750 \zref@extractdefault
1751 { \l__zrefclever_type_first_label_tl }
1752 { url } {}
1753 }
1754 }
1755 {
1756 \zref@extractdefault
1757 { \l__zrefclever_type_first_label_tl } { anchor } {}
1758 }
1759 { \exp_not:V \l__zrefclever_type_name_tl }
1760 \exp_not:N \group_end:
1761 }
1762 {
1763 \exp_not:N \group_begin:
1764 \exp_not:V \l__zrefclever_namefont_tl
1765 \exp_not:V \l__zrefclever_type_name_tl
1766 \exp_not:N \group_end:
1767 }
1768 }
1769 }
1770 {
1771 % This case would correspond to "typeset=none" but should not
1772 % happen, given the options are set up to typeset at least one
1773 % of "ref" or "name", but a sensible fallback, equal to the
1774 % behavior of ‘‘both’’.
1775 \tl_put_left:Nx
1776 \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1777 }
1778 }
1779 }

```

```

1780
1781 % Typeset the previous type, if there is one.
1782 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1783 {
1784   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1785   { \l__zrefclever_tlistsep_tl }
1786   \l__zrefclever_typeset_queue_prev_tl
1787 }
1788
1789 % Wrap up loop, or prepare for next iteration.
1790 \bool_if:NTF \l__zrefclever_typeset_last_bool
1791 {
1792   % We are finishing, typeset the current queue.
1793   \int_case:nnF { \l__zrefclever_type_count_int }
1794   {
1795     % Single type.
1796     { 0 }
1797     { \l__zrefclever_typeset_queue_curr_tl }
1798     % Pair of types.
1799     { 1 }
1800     {
1801       \l__zrefclever_tpairsep_tl
1802       \l__zrefclever_typeset_queue_curr_tl
1803     }
1804   }
1805   {
1806     % Last in list of types.
1807     \l__zrefclever_tlastsep_tl
1808     \l__zrefclever_typeset_queue_curr_tl
1809   }
1810 }
1811 {
1812   % There are further labels, set variables for next iteration.
1813   \tl_set_eq:NN
1814   \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1815   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1816   \tl_clear:N \l__zrefclever_type_first_label_tl
1817   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1818   \tl_clear:N \l__zrefclever_range_beg_label_tl
1819   \int_zero:N \l__zrefclever_label_count_int
1820   \int_incr:N \l__zrefclever_type_count_int
1821   \int_zero:N \l__zrefclever_range_count_int
1822   \int_zero:N \l__zrefclever_range_same_count_int
1823 }
1824 }

```

(End definition for `\__zrefclever_typeset_refs_aux_last_of_type:`)

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

```

1825 \cs_new_protected:Npn \__zrefclever_typeset_refs_aux_not_last_of_type:
1826 {
1827   % Signal if next label may form a range with the current one (of
1828   % course, only considered if compression is enabled in the first
1829   % place).

```

```

1830 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1831 \bool_set_false:N \l__zrefclever_next_is_same_bool
1832 \bool_lazy_and:nnT
1833 { \l__zrefclever_typeset_compress_bool }
1834 % Currently no-op, but kept as ‘handle’ to inhibit compression of
1835 % individual labels.
1836 { ! \l__zrefclever_range_inhibit_next_bool }
1837 {
1838   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1839   { }
1840   {
1841     \__zrefclever_labels_in_sequence:nn
1842     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1843   }
1844 }
1845
1846 % Process the current label to the current queue.
1847 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1848 {
1849   % Current label is the first of its type (also not the last, but it
1850   % doesn’t matter here): just store the label.
1851   \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1852   \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1853
1854   % If the next label may be part of a range, we set ‘range_beg_label’
1855   % to ‘empty’ (we deal with it as the ‘first’, and must do it
1856   % there, to handle hyperlinking), but also step the range counters.
1857   \bool_if:NT \l__zrefclever_next_maybe_range_bool
1858   {
1859     \tl_clear:N \l__zrefclever_range_beg_label_tl
1860     \int_incr:N \l__zrefclever_range_count_int
1861     \bool_if:NT \l__zrefclever_next_is_same_bool
1862     { \int_incr:N \l__zrefclever_range_same_count_int }
1863   }
1864 }
1865 {
1866   % Current label is neither the first (nor the last) of its
1867   % type.
1868   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1869   {
1870     % Starting, or continuing a range.
1871     \int_compare:nNnTF
1872     { \l__zrefclever_range_count_int } = {0}
1873     {
1874       % There was no range going, we are starting one.
1875       \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1876       \int_incr:N \l__zrefclever_range_count_int
1877       \bool_if:NT \l__zrefclever_next_is_same_bool
1878       { \int_incr:N \l__zrefclever_range_same_count_int }
1879     }
1880     {
1881       % Second or more in the range, but not the last.
1882       \int_incr:N \l__zrefclever_range_count_int
1883       \bool_if:NT \l__zrefclever_next_is_same_bool

```



```

1884         { \int_incr:N \l__zrefclever_range_same_count_int }
1885     }
1886 }
1887 {
1888 % Next element is not in sequence, meaning: there was no range, or
1889 % we are closing one.
1890 \int_case:nnF { \l__zrefclever_range_count_int }
1891 {
1892     % There was no range going on.
1893     {0}
1894     {
1895         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1896         {
1897             \exp_not:V \l__zrefclever_listsep_tl
1898             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1899         }
1900     }
1901     % Last is second in the range: if 'range_same_count' is also
1902     % '1', it's a repetition (drop it), otherwise, it's a 'pair
1903     % within a list'', treat as list.
1904     {1}
1905     {
1906         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1907         {
1908             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1909             {
1910                 \exp_not:V \l__zrefclever_listsep_tl
1911                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1912             }
1913             \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1914             {
1915                 \exp_not:V \l__zrefclever_listsep_tl
1916                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1917             }
1918         }
1919     }
1920 }
1921 {
1922 % Last is third or more in the range: if 'range_count' and
1923 % 'range_same_count' are the same, its a repetition (drop it),
1924 % if they differ by '1', its a list, if they differ by more,
1925 % it is a real range.
1926 \int_case:nnF
1927 { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1928 {
1929     {0}
1930     {
1931         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1932         {
1933             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1934             {
1935                 \exp_not:V \l__zrefclever_listsep_tl
1936                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1937             }

```

```

1938         }
1939     }
1940     {1}
1941     {
1942         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1943         {
1944             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1945             {
1946                 \exp_not:V \l__zrefclever_listsep_tl
1947                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1948             }
1949             \exp_not:V \l__zrefclever_listsep_tl
1950             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1951         }
1952     }
1953 }
1954 {
1955     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1956     {
1957         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1958         {
1959             \exp_not:V \l__zrefclever_listsep_tl
1960             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1961         }
1962         \exp_not:V \l__zrefclever_rangeseq_tl
1963         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1964     }
1965 }
1966 }
1967 % Reset counters.
1968 \int_zero:N \l__zrefclever_range_count_int
1969 \int_zero:N \l__zrefclever_range_same_count_int
1970 }
1971 }
1972 % Step label counter for next iteration.
1973 \int_incr:N \l__zrefclever_label_count_int
1974 }

```

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

## Aux functions

`\__zrefclever_get_ref:n` Auxiliary function to `\__zrefclever_typeset_refs:.` Handles a complete “ref-block”, including “pre” and “pos” elements, and *hyperlinking*. It does not handle the reference type “name”, for that use `\__zrefclever_get_ref_first:.` It should get the reference with `\zref@extractdefault` as usual but, if the reference is not available, should put `\zref@default` on the stream protected, so that it can be accumulated in the queue. `\hyperlink` must also be protected from expansion for the same reason.

```

1975 \cs_new:Npn \__zrefclever_get_ref:n #1
1976 {
1977     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
1978     {
1979         \bool_if:nTF
1980         { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }

```

```

1981 {
1982   \exp_not:N \group_begin:
1983   \exp_not:V \l__zrefclever_reffont_out_tl
1984   \exp_not:V \l__zrefclever_refpre_out_tl
1985   \exp_not:N \group_begin:
1986   \exp_not:V \l__zrefclever_reffont_in_tl
1987   % It's two '@s', but escaped for DocStrip.
1988   \exp_not:N \hyper@@link
1989   {
1990     \zref@ifrefcontainsprop {#1} { urluse }
1991     { \zref@extractdefault {#1} { urluse } {} }
1992     { \zref@extractdefault {#1} { url } {} }
1993   }
1994   { \zref@extractdefault {#1} { anchor } {} }
1995   {
1996     \exp_not:V \l__zrefclever_refpre_in_tl
1997     \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
1998     \exp_not:V \l__zrefclever_refpos_in_tl
1999   }
2000   \exp_not:N \group_end:
2001   \exp_not:V \l__zrefclever_refpos_out_tl
2002   \exp_not:N \group_end:
2003 }
2004 {
2005   \exp_not:N \group_begin:
2006   \exp_not:V \l__zrefclever_reffont_out_tl
2007   \exp_not:V \l__zrefclever_refpre_out_tl
2008   \exp_not:N \group_begin:
2009   \exp_not:V \l__zrefclever_reffont_in_tl
2010   \exp_not:V \l__zrefclever_refpre_in_tl
2011   \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2012   \exp_not:V \l__zrefclever_refpos_in_tl
2013   \exp_not:N \group_end:
2014   \exp_not:V \l__zrefclever_refpos_out_tl
2015   \exp_not:N \group_end:
2016 }
2017 }
2018 { \exp_not:N \zref@default }
2019 }
2020 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

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

\\_\_zrefclever\_type\_name\_setup: Auxiliary function to \\_\_zrefclever\_typeset\_refs:. Sets the type name variable \l\_\_zrefclever\_type\_name\_tl. When it cannot be found, clears it.

```

2021 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2022 {
2023   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2024   { \tl_clear:N \l__zrefclever_type_name_tl }
2025   {
2026     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2027     { \tl_clear:N \l__zrefclever_type_name_tl }
2028     {

```

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

```

2029 \bool_lazy_or:nnTF
2030 { \l__zrefclever_capitalize_bool }
2031 {
2032   \l__zrefclever_capitalize_first_bool &&
2033   \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2034 }
2035 { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2036 { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2037 % If the queue is empty, we have a singular, otherwise, plural.
2038 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2039 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2040 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2041 \bool_lazy_and:nnTF
2042 { \l__zrefclever_abbrev_bool }
2043 {
2044   ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
2045   ! \l__zrefclever_noabbrev_first_bool
2046 }
2047 {
2048   \tl_set:NV \l__zrefclever_name_format_fallback_tl \l__zrefclever_name_format
2049   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2050 }
2051 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2052
2053 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2054 {
2055   \prop_get:cVNF
2056   { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2057     \l__zrefclever_name_format_tl
2058     \l__zrefclever_type_name_tl
2059     {
2060       \__zrefclever_get_type_transl:xxxNF
2061       { \l__zrefclever_ref_language_tl }
2062       { \l__zrefclever_type_first_label_type_tl }
2063       { \l__zrefclever_name_format_tl }
2064       \l__zrefclever_type_name_tl
2065       {
2066         \tl_clear:N \l__zrefclever_type_name_tl
2067         \msg_warning:nxx { zref-clever } { missing-name }
2068         { \l__zrefclever_type_first_label_type_tl }
2069       }
2070     }
2071   }
2072   {
2073     \prop_get:cVNF
2074     { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2075       \l__zrefclever_name_format_tl
2076       \l__zrefclever_type_name_tl
2077       {
2078         \prop_get:cVNF
2079         { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options
2080           \l__zrefclever_name_format_fallback_tl
2081           \l__zrefclever_type_name_tl
2082           {

```

```

2083 \_zrefclever_get_type_transl:xxxNF
2084 { \l__zrefclever_ref_language_tl }
2085 { \l__zrefclever_type_first_label_type_tl }
2086 { \l__zrefclever_name_format_tl }
2087 \l__zrefclever_type_name_tl
2088 {
2089   \_zrefclever_get_type_transl:xxxNF
2090   { \l__zrefclever_ref_language_tl }
2091   { \l__zrefclever_type_first_label_type_tl }
2092   { \l__zrefclever_name_format_fallback_tl }
2093   \l__zrefclever_type_name_tl
2094   {
2095     \tl_clear:N \l__zrefclever_type_name_tl
2096     \msg_warning:nxx { zref-clever } { missing-name }
2097     { \l__zrefclever_type_first_label_type_tl }
2098   }
2099 }
2100 }
2101 }
2102 }
2103 }
2104 }

```

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

```

2105 \bool_lazy_any:nTF
2106 {
2107   { ! \l__zrefclever_use_hyperref_bool }
2108   { \l__zrefclever_link_star_bool }
2109   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2110   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2111 }
2112 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2113 {
2114   \bool_lazy_any:nTF
2115   {
2116     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2117     {
2118       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2119       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2120     }
2121     {
2122       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2123       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2124       \l__zrefclever_typeset_last_bool &&
2125       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2126     }
2127   }
2128   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2129   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2130 }
2131 }

```

(End definition for \\_zrefclever\_type\_name\_setup:.)

\\_zrefclever\_get\_ref\_first: Auxiliary function to \\_zrefclever\_typeset\_refs:. Handles a complete “ref-block”,

including “pre” and “pos” elements, *hyperlinking*, and the reference type “name”. For use on the first reference of each type.

```

2132 \cs_new:Npn \__zrefclever_get_ref_first:
2133 {
2134   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2135   { \exp_not:N \zref@default }
2136   {
2137     \bool_if:NTF \l__zrefclever_name_in_link_bool
2138     {
2139       \zref@ifrefcontainsprop
2140       { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2141       {
2142         % It's two '@s', but escaped for DocStrip.
2143         \exp_not:N \hyper@@link
2144         {
2145           \zref@ifrefcontainsprop
2146           { \l__zrefclever_type_first_label_tl } { urluse }
2147           {
2148             \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2149             { urluse } {}
2150           }
2151           {
2152             \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2153             { url } {}
2154           }
2155         }
2156       }
2157       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2158       { anchor } {}
2159     }
2160     {
2161       \exp_not:N \group_begin:
2162       \exp_not:N \l__zrefclever_namefont_tl
2163       \exp_not:N \l__zrefclever_type_name_tl
2164       \exp_not:N \group_end:
2165       \exp_not:N \l__zrefclever_namesep_tl
2166       \exp_not:N \group_begin:
2167       \exp_not:N \l__zrefclever_reffont_out_tl
2168       \exp_not:N \l__zrefclever_refpre_out_tl
2169       \exp_not:N \group_begin:
2170       \exp_not:N \l__zrefclever_reffont_in_tl
2171       \exp_not:N \l__zrefclever_refpre_in_tl
2172       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2173       { \l__zrefclever_ref_property_tl } {}
2174       \exp_not:N \l__zrefclever_refpos_in_tl
2175       \exp_not:N \group_end:
2176       % hyperlink makes it's own group, we'd like to close the
2177       % 'refpre-out' group after 'refpos-out', but... we close
2178       % it here, and give the trailing 'refpos-out' its own
2179       % group. This will result that formatting given to
2180       % 'refpre-out' will not reach 'refpos-out', but I see no
2181       % alternative, and this has to be handled specially.
2182       \exp_not:N \group_end:
2183     }
  }

```

```

2184         \exp_not:N \group_begin:
2185         % Ditto: special treatment.
2186         \exp_not:V \l__zrefclever_reffont_out_tl
2187         \exp_not:V \l__zrefclever_refpos_out_tl
2188         \exp_not:N \group_end:
2189     }
2190     {
2191         \exp_not:N \group_begin:
2192         \exp_not:V \l__zrefclever_namefont_tl
2193         \exp_not:V \l__zrefclever_type_name_tl
2194         \exp_not:N \group_end:
2195         \exp_not:V \l__zrefclever_namesep_tl
2196         \exp_not:N \zref@default
2197     }
2198 }
2199 {
2200     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2201     {
2202         \exp_not:N \zref@default
2203         \exp_not:V \l__zrefclever_namesep_tl
2204     }
2205     {
2206         \exp_not:N \group_begin:
2207         \exp_not:V \l__zrefclever_namefont_tl
2208         \exp_not:V \l__zrefclever_type_name_tl
2209         \exp_not:N \group_end:
2210         \exp_not:V \l__zrefclever_namesep_tl
2211     }
2212 \zref@ifrefcontainsprop
2213 { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2214 {
2215     \bool_if:nTF
2216     {
2217         \l__zrefclever_use_hyperref_bool &&
2218         ! \l__zrefclever_link_star_bool
2219     }
2220     {
2221         \exp_not:N \group_begin:
2222         \exp_not:V \l__zrefclever_reffont_out_tl
2223         \exp_not:V \l__zrefclever_refpre_out_tl
2224         \exp_not:N \group_begin:
2225         \exp_not:V \l__zrefclever_reffont_in_tl
2226         % It's two '@s', but escaped for DocStrip.
2227         \exp_not:N \hyper@@link
2228         {
2229             \zref@ifrefcontainsprop
2230             { \l__zrefclever_type_first_label_tl } { urluse }
2231             {
2232                 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2233                 { urluse } {}
2234             }
2235             {
2236                 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2237                 { url } {}

```

```

2238         }
2239     }
2240     {
2241         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2242         { anchor } {}
2243     }
2244     {
2245         \exp_not:V \l__zrefclever_refpre_in_tl
2246         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2247         { \l__zrefclever_ref_property_tl } {}
2248         \exp_not:V \l__zrefclever_refpos_in_tl
2249     }
2250     \exp_not:N \group_end:
2251     \exp_not:V \l__zrefclever_refpos_out_tl
2252     \exp_not:N \group_end:
2253 }
2254 {
2255     \exp_not:N \group_begin:
2256     \exp_not:V \l__zrefclever_reffont_out_tl
2257     \exp_not:V \l__zrefclever_refpre_out_tl
2258     \exp_not:N \group_begin:
2259     \exp_not:V \l__zrefclever_reffont_in_tl
2260     \exp_not:V \l__zrefclever_refpre_in_tl
2261     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2262     { \l__zrefclever_ref_property_tl } {}
2263     \exp_not:V \l__zrefclever_refpos_in_tl
2264     \exp_not:N \group_end:
2265     \exp_not:V \l__zrefclever_refpos_out_tl
2266     \exp_not:N \group_end:
2267 }
2268 }
2269 { \exp_not:N \zref@default }
2270 }
2271 }
2272 }

```

(End definition for \\_zrefclever\_get\_ref\_first:.)

\\_zrefclever\_get\_option\_with\_transl:nN

```

2273 % \Arg{option} \Arg{var to store result}
2274 \cs_new_protected:Npn \_zrefclever_get_option_with_transl:nN #1#2
2275 {
2276     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2277     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2278     {
2279         % If not found, try the type specific options.
2280         \bool_lazy_all:nTF
2281         {
2282             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2283             {
2284                 \prop_if_exist_p:c
2285                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2286             }
2287         }

```



```

2288         \prop_if_in_p:cn
2289         { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2290     }
2291 }
2292 {
2293     \prop_get:cnN
2294     { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2295 }
2296 {
2297     % If not found, try the type specific translations.
2298     \__zrefclever_get_type_transl:xnNF
2299     { \l__zrefclever_ref_language_tl }
2300     { \l__zrefclever_label_type_a_tl }
2301     {#1} #2
2302     {
2303         % If not found, try default translations.
2304         \__zrefclever_get_default_transl:xnNF
2305         { \l__zrefclever_ref_language_tl }
2306         {#1} #2
2307         {
2308             % If not found, try fallback.
2309             \__zrefclever_get_fallback_transl:nNF {#1} #2
2310             { \tl_clear:N #2 }
2311         }
2312     }
2313 }
2314 }
2315 }

```

(End definition for \\_\_zrefclever\_get\_option\_with\_transl:nN.)

\\_\_zrefclever\_get\_option\_plain:nN

```

2316 \cs_new_protected:Npn \__zrefclever_get_option_plain:nN #1#2
2317 {
2318     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2319     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2320     {
2321         % If not found, try the type specific options.
2322         \bool_lazy_and:nnTF
2323         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2324         {
2325             \prop_if_exist_p:c
2326             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2327         }
2328         {
2329             \prop_get:cnNF
2330             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2331             { \tl_clear:N #2 }
2332         }
2333         { \tl_clear:N #2 }
2334     }
2335 }

```

(End definition for \\_\_zrefclever\_get\_option\_plain:nN.)

`\_zrefclever_labels_in_sequence:nn` Sets `\l__zrefclever_next_maybe_range_bool` to true if label ‘1’ comes in immediate sequence from label ‘2’. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__zrefclever_next_is_same_bool` if the labels are the “same”.

```

2336 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
2337 {
2338   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2339   {
2340     \exp_args:Nxx \tl_if_eq:nnT
2341     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2342     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2343     {
2344       \int_compare:nNnTF
2345       { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2346       =
2347       { \zref@extractdefault {#2} { zc@pgval } {-1} }
2348       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2349       {
2350         \int_compare:nNnT
2351         { \zref@extractdefault {#1} { zc@pgval } {-1} }
2352         =
2353         { \zref@extractdefault {#2} { zc@pgval } {-1} }
2354         {
2355           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2356           \bool_set_true:N \l__zrefclever_next_is_same_bool
2357         }
2358       }
2359     }
2360   }
2361   {
2362     \exp_args:Nxx \tl_if_eq:nnT
2363     { \zref@extractdefault {#1} { counter } { } }
2364     { \zref@extractdefault {#2} { counter } { } }
2365     {
2366       \exp_args:Nxx \tl_if_eq:nnT
2367       { \zref@extractdefault {#1} { zc@enclval } { } }
2368       { \zref@extractdefault {#2} { zc@enclval } { } }
2369       {
2370         \int_compare:nNnTF
2371         { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
2372         =
2373         { \zref@extractdefault {#2} { zc@cntval } {-1} }
2374         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2375         {
2376           \int_compare:nNnT
2377           { \zref@extractdefault {#1} { zc@cntval } {-1} }
2378           =
2379           { \zref@extractdefault {#2} { zc@cntval } {-1} }
2380           {
2381             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2382             \bool_set_true:N \l__zrefclever_next_is_same_bool
2383           }
2384         }
2385       }
2386     }

```

```

2387     }
2388 }

```

(End definition for `\_zrefclever_labels_in_sequence:nn`.)

## 9 Special handling

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. It is not meant to be a “kitchen sink of workarounds”. Rather, I intend to keep this as lean as possible, trying to add things selectively when they are safe and reasonable. And, hopefully, doing so by proper setting of `zref-clever`’s options, not by messing with other packages’ code. In particular, I do not mean to compensate for “lack of support for `zref`” by individual packages here, unless there is really no alternative.

### 9.1 `\appendix`

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

### 9.2 `\newtheorem`

### 9.3 `enumitem` package

TODO Option `counterresetby` should probably be extended for `enumitem`, conditioned on it being loaded.

## 10 Dictionaries

```

2389 </package>

```

### 10.1 English

```

<package> <package> <package> <package> <package> <package> <package>
<package>

```

All options retrieved with `\_zrefclever_get_option_with_transl:nN` must have their values set for ‘English’, since this is what will be retrieved if no language package is loaded.

```

2390 <*dict-english>

```

```

2391 namesep   = {\nobreakspace} ,
2392 pairsep    = {\~and\nobreakspace} ,
2393 listsep     = {\~,~} ,
2394 lastsep     = {\~and\nobreakspace} ,
2395 tpairsep    = {\~and\nobreakspace} ,
2396 tlistsep    = {\~,~} ,
2397 tlastsep    = {\~,~and\nobreakspace} ,
2398 notesep     = {\~} ,
2399 rangesep    = {\~to\nobreakspace} ,
2400 refpre      = ,
2401 refpos      = ,
2402 refpre-in   = ,
2403 refpos-in   = ,
2404
2405 type = part ,
2406   Name-sg = Part ,
2407   name-sg = part ,
2408   Name-pl = Parts ,
2409   name-pl = parts ,
2410
2411 type = chapter ,
2412   Name-sg = Chapter ,
2413   name-sg = chapter ,
2414   Name-pl = Chapters ,
2415   name-pl = chapters ,
2416
2417 type = section ,
2418   Name-sg = Section ,
2419   name-sg = section ,
2420   Name-pl = Sections ,
2421   name-pl = sections ,
2422
2423 type = paragraph ,
2424   Name-sg = Paragraph ,
2425   name-sg = paragraph ,
2426   Name-pl = Paragraphs ,
2427   name-pl = paragraphs ,
2428   Name-sg-ab = Par. ,
2429   name-sg-ab = par. ,
2430   Name-pl-ab = Par. ,
2431   name-pl-ab = par. ,
2432
2433 type = appendix ,
2434   Name-sg = Appendix ,
2435   name-sg = appendix ,
2436   Name-pl = Appendices ,
2437   name-pl = appendices ,
2438
2439 type = page ,
2440   Name-sg = Page ,
2441   name-sg = page ,
2442   Name-pl = Pages ,
2443   name-pl = pages ,
2444   name-sg-ab = p. ,

```

```

2445     name-pl-ab = pp. ,
2446
2447 type = line ,
2448     Name-sg = Line ,
2449     name-sg = line ,
2450     Name-pl = Lines ,
2451     name-pl = lines ,
2452
2453 type = figure ,
2454     Name-sg = Figure ,
2455     name-sg = figure ,
2456     Name-pl = Figures ,
2457     name-pl = figures ,
2458     Name-sg-ab = Fig. ,
2459     name-sg-ab = fig. ,
2460     Name-pl-ab = Figs. ,
2461     name-pl-ab = figs. ,
2462
2463 type = table ,
2464     Name-sg = Table ,
2465     name-sg = table ,
2466     Name-pl = Tables ,
2467     name-pl = tables ,
2468
2469 type = item ,
2470     Name-sg = Item ,
2471     name-sg = item ,
2472     Name-pl = Items ,
2473     name-pl = items ,
2474
2475 type = footnote ,
2476     Name-sg = Footnote ,
2477     name-sg = footnote ,
2478     Name-pl = Footnotes ,
2479     name-pl = footnotes ,
2480
2481 type = note ,
2482     Name-sg = Note ,
2483     name-sg = note ,
2484     Name-pl = Notes ,
2485     name-pl = notes ,
2486
2487 type = equation ,
2488     Name-sg = Equation ,
2489     name-sg = equation ,
2490     Name-pl = Equations ,
2491     name-pl = equations ,
2492     Name-sg-ab = Eq. ,
2493     name-sg-ab = eq. ,
2494     Name-pl-ab = Eqs. ,
2495     name-pl-ab = eqs. ,
2496     refpre-in = {(} ,
2497     refpos-in = {)} ,
2498

```

```

2499 type = theorem ,
2500     Name-sg = Theorem ,
2501     name-sg = theorem ,
2502     Name-pl = Theorems ,
2503     name-pl = theorems ,
2504
2505 type = lemma ,
2506     Name-sg = Lemma ,
2507     name-sg = lemma ,
2508     Name-pl = Lemmas ,
2509     name-pl = lemmas ,
2510
2511 type = corollary ,
2512     Name-sg = Corollary ,
2513     name-sg = corollary ,
2514     Name-pl = Corollaries ,
2515     name-pl = corollaries ,
2516
2517 type = proposition ,
2518     Name-sg = Proposition ,
2519     name-sg = proposition ,
2520     Name-pl = Propositions ,
2521     name-pl = propositions ,
2522
2523 type = definition ,
2524     Name-sg = Definition ,
2525     name-sg = definition ,
2526     Name-pl = Definitions ,
2527     name-pl = definitions ,
2528
2529 type = proof ,
2530     Name-sg = Proof ,
2531     name-sg = proof ,
2532     Name-pl = Proofs ,
2533     name-pl = proofs ,
2534
2535 type = result ,
2536     Name-sg = Result ,
2537     name-sg = result ,
2538     Name-pl = Results ,
2539     name-pl = results ,
2540
2541 type = example ,
2542     Name-sg = Example ,
2543     name-sg = example ,
2544     Name-pl = Examples ,
2545     name-pl = examples ,
2546
2547 type = remark ,
2548     Name-sg = Remark ,
2549     name-sg = remark ,
2550     Name-pl = Remarks ,
2551     name-pl = remarks ,
2552

```

```

2553 type = algorithm ,
2554   Name-sg = Algorithm ,
2555   name-sg = algorithm ,
2556   Name-pl = Algorithms ,
2557   name-pl = algorithms ,
2558
2559 type = listing ,
2560   Name-sg = Listing ,
2561   name-sg = listing ,
2562   Name-pl = Listings ,
2563   name-pl = listings ,
2564
2565 type = exercise ,
2566   Name-sg = Exercise ,
2567   name-sg = exercise ,
2568   Name-pl = Exercises ,
2569   name-pl = exercises ,
2570
2571 type = solution ,
2572   Name-sg = Solution ,
2573   name-sg = solution ,
2574   Name-pl = Solutions ,
2575   name-pl = solutions ,
2576 </dict-english>

```

## 10.2 German

<package> <package> <package> <package> <package> <package> <package>

```

2577 <*dict-german>
2578 namesep = {\nobreakspace} ,
2579 pairsep = {\simund\nobreakspace} ,
2580 listsep = {,~} ,
2581 lastsep = {\simund\nobreakspace} ,
2582 tpairsep = {\simund\nobreakspace} ,
2583 tlistsep = {,~} ,
2584 tlastsep = {\simund\nobreakspace} ,
2585 notesep = {\sim} ,
2586 rangesep = {\simbis\nobreakspace} ,
2587
2588 type = part ,
2589   Name-sg = Teil ,
2590   name-sg = Teil ,
2591   Name-pl = Teile ,
2592   name-pl = Teile ,
2593
2594 type = chapter ,
2595   Name-sg = Kapitel ,
2596   name-sg = Kapitel ,
2597   Name-pl = Kapitel ,
2598   name-pl = Kapitel ,
2599
2600 type = section ,
2601   Name-sg = Abschnitt ,

```

```

2602     name-sg = Abschnitt ,
2603     Name-pl = Abschnitte ,
2604     name-pl = Abschnitte ,
2605
2606 type = paragraph ,
2607     Name-sg = Absatz ,
2608     name-sg = Absatz ,
2609     Name-pl = Absätze ,
2610     name-pl = Absätze ,
2611
2612 type = appendix ,
2613     Name-sg = Anhang ,
2614     name-sg = Anhang ,
2615     Name-pl = Anhänge ,
2616     name-pl = Anhänge ,
2617
2618 type = page ,
2619     Name-sg = Seite ,
2620     name-sg = Seite ,
2621     Name-pl = Seiten ,
2622     name-pl = Seiten ,
2623
2624 type = line ,
2625     Name-sg = Zeile ,
2626     name-sg = Zeile ,
2627     Name-pl = Zeilen ,
2628     name-pl = Zeilen ,
2629
2630 type = figure ,
2631     Name-sg = Abbildung ,
2632     name-sg = Abbildung ,
2633     Name-pl = Abbildungen ,
2634     name-pl = Abbildungen ,
2635     Name-sg-ab = Abb. ,
2636     name-sg-ab = Abb. ,
2637     Name-pl-ab = Abb. ,
2638     name-pl-ab = Abb. ,
2639
2640 type = table ,
2641     Name-sg = Tabelle ,
2642     name-sg = Tabelle ,
2643     Name-pl = Tabellen ,
2644     name-pl = Tabellen ,
2645
2646 type = item ,
2647     Name-sg = Punkt ,
2648     name-sg = Punkt ,
2649     Name-pl = Punkte ,
2650     name-pl = Punkte ,
2651
2652 type = footnote ,
2653     Name-sg = Fußnote ,
2654     name-sg = Fußnote ,
2655     Name-pl = Fußnoten ,

```



```

2656     name-pl = Fußnoten ,
2657
2658 type = note ,
2659     Name-sg = Anmerkung ,
2660     name-sg = Anmerkung ,
2661     Name-pl = Anmerkungen ,
2662     name-pl = Anmerkungen ,
2663
2664 type = equation ,
2665     Name-sg = Gleichung ,
2666     name-sg = Gleichung ,
2667     Name-pl = Gleichungen ,
2668     name-pl = Gleichungen ,
2669     refpre-in = {} ,
2670     refpos-in = {} ,
2671
2672 type = theorem ,
2673     Name-sg = Theorem ,
2674     name-sg = Theorem ,
2675     Name-pl = Theoreme ,
2676     name-pl = Theoreme ,
2677
2678 type = lemma ,
2679     Name-sg = Lemma ,
2680     name-sg = Lemma ,
2681     Name-pl = Lemmata ,
2682     name-pl = Lemmata ,
2683
2684 type = corollary ,
2685     Name-sg = Korollar ,
2686     name-sg = Korollar ,
2687     Name-pl = Korollare ,
2688     name-pl = Korollare ,
2689
2690 type = proposition ,
2691     Name-sg = Satz ,
2692     name-sg = Satz ,
2693     Name-pl = Sätze ,
2694     name-pl = Sätze ,
2695
2696 type = definition ,
2697     Name-sg = Definition ,
2698     name-sg = Definition ,
2699     Name-pl = Definitionen ,
2700     name-pl = Definitionen ,
2701
2702 type = proof ,
2703     Name-sg = Beweis ,
2704     name-sg = Beweis ,
2705     Name-pl = Beweise ,
2706     name-pl = Beweise ,
2707
2708 type = result ,
2709     Name-sg = Ergebnis ,

```

```

2710 name-sg = Ergebnis ,
2711 Name-pl = Ergebnisse ,
2712 name-pl = Ergebnisse ,
2713
2714 type = example ,
2715 Name-sg = Beispiel ,
2716 name-sg = Beispiel ,
2717 Name-pl = Beispiele ,
2718 name-pl = Beispiele ,
2719
2720 type = remark ,
2721 Name-sg = Bemerkung ,
2722 name-sg = Bemerkung ,
2723 Name-pl = Bemerkungen ,
2724 name-pl = Bemerkungen ,
2725
2726 type = algorithm ,
2727 Name-sg = Algorithmus ,
2728 name-sg = Algorithmus ,
2729 Name-pl = Algorithmen ,
2730 name-pl = Algorithmen ,
2731
2732 type = listing ,
2733 Name-sg = Listing , % CHECK
2734 name-sg = Listing , % CHECK
2735 Name-pl = Listings , % CHECK
2736 name-pl = Listings , % CHECK
2737
2738 type = exercise ,
2739 Name-sg = Übungsaufgabe ,
2740 name-sg = Übungsaufgabe ,
2741 Name-pl = Übungsaufgaben ,
2742 name-pl = Übungsaufgaben ,
2743
2744 type = solution ,
2745 Name-sg = Lösung ,
2746 name-sg = Lösung ,
2747 Name-pl = Lösungen ,
2748 name-pl = Lösungen ,
2749 </dict-german>

```

### 10.3 French

```

<package> <package> <package> <package> <package>
2750 <*dict-french>
2751 namesep = {\nobreakspace} ,
2752 pairsep = {\~et\nobreakspace} ,
2753 listsep = {\,~} ,
2754 lastsep = {\~et\nobreakspace} ,
2755 tpairsep = {\~et\nobreakspace} ,
2756 tlistsep = {\,~} ,
2757 tlastsep = {\~et\nobreakspace} ,
2758 notesep = {\~} ,

```

```

2759 rangesep = {\~\nobreakspace} ,
2760
2761 type = part ,
2762   Name-sg = Partie ,
2763   name-sg = partie ,
2764   Name-pl = Parties ,
2765   name-pl = parties ,
2766
2767 type = chapter ,
2768   Name-sg = Chapitre ,
2769   name-sg = chapitre ,
2770   Name-pl = Chapitres ,
2771   name-pl = chapitres ,
2772
2773 type = section ,
2774   Name-sg = Section ,
2775   name-sg = section ,
2776   Name-pl = Sections ,
2777   name-pl = sections ,
2778
2779 type = paragraph ,
2780   Name-sg = Paragraphe ,
2781   name-sg = paragraphe ,
2782   Name-pl = Paragraphes ,
2783   name-pl = paragraphes ,
2784
2785 type = appendix ,
2786   Name-sg = Annexe ,
2787   name-sg = annexe ,
2788   Name-pl = Annexes ,
2789   name-pl = annexes ,
2790
2791 type = page ,
2792   Name-sg = Page ,
2793   name-sg = page ,
2794   Name-pl = Pages ,
2795   name-pl = pages ,
2796
2797 type = line ,
2798   Name-sg = Ligne ,
2799   name-sg = ligne ,
2800   Name-pl = Lignes ,
2801   name-pl = lignes ,
2802
2803 type = figure ,
2804   Name-sg = Figure ,
2805   name-sg = figure ,
2806   Name-pl = Figures ,
2807   name-pl = figures ,
2808
2809 type = table ,
2810   Name-sg = Table ,
2811   name-sg = table ,
2812   Name-pl = Tables ,

```

```

2813     name-pl = tables ,
2814
2815 type = item ,
2816     Name-sg = Point ,
2817     name-sg = point ,
2818     Name-pl = Points ,
2819     name-pl = points ,
2820
2821 type = footnote ,
2822     Name-sg = Note ,
2823     name-sg = note ,
2824     Name-pl = Notes ,
2825     name-pl = notes ,
2826
2827 type = note ,
2828     Name-sg = Note ,
2829     name-sg = note ,
2830     Name-pl = Notes ,
2831     name-pl = notes ,
2832
2833 type = equation ,
2834     Name-sg = Équation ,
2835     name-sg = équation ,
2836     Name-pl = Équations ,
2837     name-pl = équations ,
2838     refpre-in = {() ,
2839     refpos-in = {} } ,
2840
2841 type = theorem ,
2842     Name-sg = Théorème ,
2843     name-sg = théorème ,
2844     Name-pl = Théorèmes ,
2845     name-pl = théorèmes ,
2846
2847 type = lemma ,
2848     Name-sg = Lemme ,
2849     name-sg = lemme ,
2850     Name-pl = Lemmes ,
2851     name-pl = lemmes ,
2852
2853 type = corollary ,
2854     Name-sg = Corollaire ,
2855     name-sg = corollaire ,
2856     Name-pl = Corollaires ,
2857     name-pl = corollaires ,
2858
2859 type = proposition ,
2860     Name-sg = Proposition ,
2861     name-sg = proposition ,
2862     Name-pl = Propositions ,
2863     name-pl = propositions ,
2864
2865 type = definition ,
2866     Name-sg = Définition ,

```

```

2867     name-sg = définition ,
2868     Name-pl = Définitions ,
2869     name-pl = définitions ,
2870
2871 type = proof ,
2872     Name-sg = Démonstration ,
2873     name-sg = démonstration ,
2874     Name-pl = Démonstrations ,
2875     name-pl = démonstrations ,
2876
2877 type = result ,
2878     Name-sg = Résultat ,
2879     name-sg = résultat ,
2880     Name-pl = Résultats ,
2881     name-pl = résultats ,
2882
2883 type = example ,
2884     Name-sg = Exemple ,
2885     name-sg = exemple ,
2886     Name-pl = Exemples ,
2887     name-pl = exemples ,
2888
2889 type = remark ,
2890     Name-sg = Remarque ,
2891     name-sg = remarque ,
2892     Name-pl = Remarques ,
2893     name-pl = remarques ,
2894
2895 type = algorithm ,
2896     Name-sg = Algorithme ,
2897     name-sg = algorithme ,
2898     Name-pl = Algorithmes ,
2899     name-pl = algorithmes ,
2900
2901 type = listing ,
2902     Name-sg = Liste ,
2903     name-sg = liste ,
2904     Name-pl = Listes ,
2905     name-pl = listes ,
2906
2907 type = exercise ,
2908     Name-sg = Exercice ,
2909     name-sg = exercice ,
2910     Name-pl = Exercices ,
2911     name-pl = exercices ,
2912
2913 type = solution ,
2914     Name-sg = Solution ,
2915     name-sg = solution ,
2916     Name-pl = Solutions ,
2917     name-pl = solutions ,
2918 </dict-french>

```

## 10.4 Portuguese

<package> <package> <package> <package>

```
2919 \(*dict-portuguese)
2920 namesep = {\nobreakspace} ,
2921 pairsep = {\~e\nobreakspace} ,
2922 listsep = {,~} ,
2923 lastsep = {\~e\nobreakspace} ,
2924 tpairsep = {\~e\nobreakspace} ,
2925 tlistsep = {,~} ,
2926 tlastsep = {\~e\nobreakspace} ,
2927 notesep = {\~} ,
2928 rangesep = {\~a\nobreakspace} ,
2929
2930 type = part ,
2931   Name-sg = Parte ,
2932   name-sg = parte ,
2933   Name-pl = Partes ,
2934   name-pl = partes ,
2935
2936 type = chapter ,
2937   Name-sg = Capítulo ,
2938   name-sg = capítulo ,
2939   Name-pl = Capítulos ,
2940   name-pl = capítulos ,
2941
2942 type = section ,
2943   Name-sg = Seção ,
2944   name-sg = seção ,
2945   Name-pl = Seções ,
2946   name-pl = seções ,
2947
2948 type = paragraph ,
2949   Name-sg = Parágrafo ,
2950   name-sg = parágrafo ,
2951   Name-pl = Parágrafos ,
2952   name-pl = parágrafos ,
2953   Name-sg-ab = Par. ,
2954   name-sg-ab = par. ,
2955   Name-pl-ab = Par. ,
2956   name-pl-ab = par. ,
2957
2958 type = appendix ,
2959   Name-sg = Apêndice ,
2960   name-sg = apêndice ,
2961   Name-pl = Apêndices ,
2962   name-pl = apêndices ,
2963
2964 type = page ,
2965   Name-sg = Página ,
2966   name-sg = página ,
2967   Name-pl = Páginas ,
2968   name-pl = páginas ,
2969   name-sg-ab = p. ,
```

```

2970     name-pl-ab = pp. ,
2971
2972 type = line ,
2973     Name-sg = Linha ,
2974     name-sg = linha ,
2975     Name-pl = Linhas ,
2976     name-pl = linhas ,
2977
2978 type = figure ,
2979     Name-sg = Figura ,
2980     name-sg = figura ,
2981     Name-pl = Figuras ,
2982     name-pl = figuras ,
2983     Name-sg-ab = Fig. ,
2984     name-sg-ab = fig. ,
2985     Name-pl-ab = Figs. ,
2986     name-pl-ab = figs. ,
2987
2988 type = table ,
2989     Name-sg = Tabela ,
2990     name-sg = tabela ,
2991     Name-pl = Tabelas ,
2992     name-pl = tabelas ,
2993
2994 type = item ,
2995     Name-sg = Item ,
2996     name-sg = item ,
2997     Name-pl = Itens ,
2998     name-pl = itens ,
2999
3000 type = footnote ,
3001     Name-sg = Nota ,
3002     name-sg = nota ,
3003     Name-pl = Notas ,
3004     name-pl = notas ,
3005
3006 type = note ,
3007     Name-sg = Nota ,
3008     name-sg = nota ,
3009     Name-pl = Notas ,
3010     name-pl = notas ,
3011
3012 type = equation ,
3013     Name-sg = Equação ,
3014     name-sg = equação ,
3015     Name-pl = Equações ,
3016     name-pl = equações ,
3017     Name-sg-ab = Eq. ,
3018     name-sg-ab = eq. ,
3019     Name-pl-ab = Eqs. ,
3020     name-pl-ab = eqs. ,
3021     refpre-in = ( ,
3022     refpos-in = ) ,
3023

```

```

3024 type = theorem ,
3025     Name-sg = Teorema ,
3026     name-sg = teorema ,
3027     Name-pl = Teoremas ,
3028     name-pl = teoremas ,
3029
3030 type = lemma ,
3031     Name-sg = Lema ,
3032     name-sg = lema ,
3033     Name-pl = Lemas ,
3034     name-pl = lemas ,
3035
3036 type = corollary ,
3037     Name-sg = Corolário ,
3038     name-sg = corolário ,
3039     Name-pl = Corolários ,
3040     name-pl = corolários ,
3041
3042 type = proposition ,
3043     Name-sg = Proposição ,
3044     name-sg = proposição ,
3045     Name-pl = Proposições ,
3046     name-pl = proposições ,
3047
3048 type = definition ,
3049     Name-sg = Definição ,
3050     name-sg = definição ,
3051     Name-pl = Definições ,
3052     name-pl = definições ,
3053
3054 type = proof ,
3055     Name-sg = Demonstração ,
3056     name-sg = demonstração ,
3057     Name-pl = Demonstrações ,
3058     name-pl = demonstrações ,
3059
3060 type = result ,
3061     Name-sg = Resultado ,
3062     name-sg = resultado ,
3063     Name-pl = Resultados ,
3064     name-pl = resultados ,
3065
3066 type = example ,
3067     Name-sg = Exemplo ,
3068     name-sg = exemplo ,
3069     Name-pl = Exemplos ,
3070     name-pl = exemplos ,
3071
3072 type = remark ,
3073     Name-sg = Observação ,
3074     name-sg = observação ,
3075     Name-pl = Observações ,
3076     name-pl = observações ,
3077

```



```

3078 type = algorithm ,
3079   Name-sg = Algoritmo ,
3080   name-sg = algoritmo ,
3081   Name-pl = Algoritmos ,
3082   name-pl = algoritmos ,
3083
3084 type = listing ,
3085   Name-sg = Listagem ,
3086   name-sg = listagem ,
3087   Name-pl = Listagens ,
3088   name-pl = listagens ,
3089
3090 type = exercise ,
3091   Name-sg = Exercício ,
3092   name-sg = exercício ,
3093   Name-pl = Exercícios ,
3094   name-pl = exercícios ,
3095
3096 type = solution ,
3097   Name-sg = Solução ,
3098   name-sg = solução ,
3099   Name-pl = Soluções ,
3100   name-pl = soluções ,
3101 </dict-portuguese>

```

## 10.5 Spanish

<package>

```

3102 <*dict-spanish>

3103 namesep = {\nobreakspace} ,
3104 pairsep = {\~y\nobreakspace} ,
3105 listsep = {,~} ,
3106 lastsep = {\~y\nobreakspace} ,
3107 tpairsep = {\~y\nobreakspace} ,
3108 tlistsep = {,~} ,
3109 tlastsep = {\~y\nobreakspace} ,
3110 notesep = {\~} ,
3111 rangesep = {\~a\nobreakspace} ,
3112
3113 type = part ,
3114   Name-sg = Parte ,
3115   name-sg = parte ,
3116   Name-pl = Partes ,
3117   name-pl = partes ,
3118
3119 type = chapter ,
3120   Name-sg = Capítulo ,
3121   name-sg = capítulo ,
3122   Name-pl = Capítulos ,
3123   name-pl = capítulos ,
3124
3125 type = section ,
3126   Name-sg = Sección ,

```

```

3127     name-sg = sección ,
3128     Name-pl = Secciones ,
3129     name-pl = secciones ,
3130
3131     type = paragraph ,
3132     Name-sg = Párrafo ,
3133     name-sg = párrafo ,
3134     Name-pl = Párrafos ,
3135     name-pl = párrafos ,
3136
3137     type = appendix ,
3138     Name-sg = Apéndice ,
3139     name-sg = apéndice ,
3140     Name-pl = Apéndices ,
3141     name-pl = apéndices ,
3142
3143     type = page ,
3144     Name-sg = Página ,
3145     name-sg = página ,
3146     Name-pl = Páginas ,
3147     name-pl = páginas ,
3148
3149     type = line ,
3150     Name-sg = Línea ,
3151     name-sg = línea ,
3152     Name-pl = Líneas ,
3153     name-pl = líneas ,
3154
3155     type = figure ,
3156     Name-sg = Figura ,
3157     name-sg = figura ,
3158     Name-pl = Figuras ,
3159     name-pl = figuras ,
3160
3161     type = table ,
3162     Name-sg = Cuadro ,
3163     name-sg = cuadro ,
3164     Name-pl = Cuadros ,
3165     name-pl = cuadros ,
3166
3167     type = item ,
3168     Name-sg = Punto ,
3169     name-sg = punto ,
3170     Name-pl = Puntos ,
3171     name-pl = puntos ,
3172
3173     type = footnote ,
3174     Name-sg = Nota ,
3175     name-sg = nota ,
3176     Name-pl = Notas ,
3177     name-pl = notas ,
3178
3179     type = note ,
3180     Name-sg = Nota ,

```

```

3181     name-sg = nota ,
3182     Name-pl = Notas ,
3183     name-pl = notas ,
3184
3185 type = equation ,
3186     Name-sg = Ecuación ,
3187     name-sg = ecuación ,
3188     Name-pl = Ecuaciones ,
3189     name-pl = ecuaciones ,
3190     refpre-in = ( ,
3191     refpos-in = ) ,
3192
3193 type = theorem ,
3194     Name-sg = Teorema ,
3195     name-sg = teorema ,
3196     Name-pl = Teoremas ,
3197     name-pl = teoremas ,
3198
3199 type = lemma ,
3200     Name-sg = Lema ,
3201     name-sg = lema ,
3202     Name-pl = Lemas ,
3203     name-pl = lemas ,
3204
3205 type = corollary ,
3206     Name-sg = Corolario ,
3207     name-sg = corolario ,
3208     Name-pl = Corolarios ,
3209     name-pl = corolarios ,
3210
3211 type = proposition ,
3212     Name-sg = Proposición ,
3213     name-sg = proposición ,
3214     Name-pl = Proposiciones ,
3215     name-pl = proposiciones ,
3216
3217 type = definition ,
3218     Name-sg = Definición ,
3219     name-sg = definición ,
3220     Name-pl = Definiciones ,
3221     name-pl = definiciones ,
3222
3223 type = proof ,
3224     Name-sg = Demostración ,
3225     name-sg = demostración ,
3226     Name-pl = Demostraciones ,
3227     name-pl = demostraciones ,
3228
3229 type = result ,
3230     Name-sg = Resultado ,
3231     name-sg = resultado ,
3232     Name-pl = Resultados ,
3233     name-pl = resultados ,
3234

```

```

3235 type = example ,
3236   Name-sg = Ejemplo ,
3237   name-sg = ejemplo ,
3238   Name-pl = Ejemplos ,
3239   name-pl = ejemplos ,
3240
3241 type = remark ,
3242   Name-sg = Observación ,
3243   name-sg = observación ,
3244   Name-pl = Observaciones ,
3245   name-pl = observaciones ,
3246
3247 type = algorithm ,
3248   Name-sg = Algoritmo ,
3249   name-sg = algoritmo ,
3250   Name-pl = Algoritmos ,
3251   name-pl = algoritmos ,
3252
3253 type = listing ,
3254   Name-sg = Listado ,
3255   name-sg = listado ,
3256   Name-pl = Listados ,
3257   name-pl = listados ,
3258
3259 type = exercise ,
3260   Name-sg = Ejercicio ,
3261   name-sg = ejercicio ,
3262   Name-pl = Ejercicios ,
3263   name-pl = ejercicios ,
3264
3265 type = solution ,
3266   Name-sg = Solución ,
3267   name-sg = solución ,
3268   Name-pl = Soluciones ,
3269   name-pl = soluciones ,
3270 </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	
\\	103, 109, 120, 125, 126, 135, 145	\babelname	732
		\babelprovide	21
		bool commands:	
		\bool_case_true:	2
		\bool_if:NTF	268, 277, 658, 662, 1485, 1558, 1683, 1704, 1735, 1790, 1857, 1861, 1868, 1877, 1883, 2137
		\bool_if:nTF	59, 1124, 1133, 1142, 1213, 1241, 1264, 1353, 1361, 1499,
A			
\AddToHook	91, 529, 544, 654, 722, 746, 775, 777, 828		
\appendix	59		
\appendixname	59		
\Arg	2273		

1507, 1716, 1723, 1730, 1979, 2215	2008, 2013, 2015, 2018, 2135, 2143,
\bool_lazy_all:nTF . . . . . 2280	2161, 2164, 2166, 2169, 2175, 2182,
\bool_lazy_and:nnTF . . . . .	2184, 2188, 2191, 2194, 2196, 2202,
1017, 1028, 1832, 2041, 2322	2206, 2209, 2221, 2224, 2227, 2250,
\bool_lazy_any:nTF . . . . . 2105, 2114	2252, 2255, 2258, 2264, 2266, 2269
\bool_lazy_or:nnTF . . . . . 1021, 2029	\exp_not:n . 1589, 1605, 1617, 1621,
\bool_new:N . . . . . 240, 565,	1641, 1654, 1657, 1669, 1672, 1705,
566, 591, 615, 624, 631, 632, 687,	1706, 1738, 1759, 1764, 1765, 1897,
688, 705, 706, 821, 822, 1039, 1056,	1910, 1915, 1935, 1946, 1949, 1959,
1393, 1394, 1405, 1406, 1407, 1426	1962, 1983, 1984, 1986, 1996, 1998,
\bool_set:Nn . . . . . 1015	2001, 2006, 2007, 2009, 2010, 2012,
\bool_set_false:N . . . . .	2014, 2162, 2163, 2165, 2167, 2168,
578, 582, 639, 648, 649,	2170, 2171, 2174, 2186, 2187, 2192,
664, 843, 1202, 1452, 1491, 1505,	2193, 2195, 2203, 2207, 2208, 2210,
1516, 1695, 1830, 1831, 2112, 2129	2222, 2223, 2225, 2245, 2248, 2251,
\bool_set_true:N . . . . .	2256, 2257, 2259, 2260, 2263, 2265
286, 572, 573, 577, 583, 638, 643,	\ExplSyntaxOn . . . . . 254
644, 832, 837, 1225, 1236, 1253,	
1259, 1276, 1282, 1308, 1320, 1459,	<b>F</b>
1486, 1492, 1496, 1517, 1520, 2128,	file commands:
2348, 2355, 2356, 2374, 2381, 2382	\file_get:nnNTF . . . . . 252
\bool_until_do:Nn . . . . . 1206, 1453	\fmtversion . . . . . 3
<b>C</b>	<b>G</b>
clist commands:	group commands:
\clist_map_inline:nn . . . . . 474	\group_begin: . . . . . 93,
\counterwithin . . . . . 4	285, 1012, 1737, 1763, 1982, 1985,
\cs . . . . . 1332, 1681, 2276, 2318	2005, 2008, 2161, 2166, 2169, 2184,
cs commands:	2191, 2206, 2221, 2224, 2255, 2258
\cs_generate_variant:Nn . . . . .	\group_end: . . . . . 96,
55, 56, 281, 290, 314, 322, 1057, 2020	288, 1036, 1760, 1766, 2000, 2002,
\cs_if_exist:NTF . . . . . 39, 48, 69	2013, 2015, 2164, 2175, 2182, 2188,
\cs_new:Npn 37, 46, 57, 67, 78, 1975, 2132	2194, 2209, 2250, 2252, 2264, 2266
\cs_new_protected:Npn . . 243, 283,	
293, 301, 309, 317, 435, 1010, 1058,	<b>H</b>
1074, 1117, 1183, 1328, 1384, 1429,	\hyperlink . . . . . 50
1569, 1825, 2021, 2274, 2316, 2336	
\cs_new_protected:Npx . . . . . 90	<b>I</b>
\cs_set_eq:NN . . . . . 94	\IfBooleanTF . . . . . 1042
	\IfFormatAtLeastTF . . . . . 3, 4
<b>E</b>	int commands:
\endinput . . . . . 12	\int_case:nnTF . . . . .
exp commands:	1572, 1598, 1629, 1793, 1890, 1926
\exp_args:NNe . . . . . 27	\int_compare:nNnTF . . . . . 1171,
\exp_args:NNnx . . . . . 230	1226, 1293, 1309, 1339, 1341, 1386,
\exp_args:NnV . . . . . 260	1540, 1585, 1619, 1782, 1784, 1847,
\exp_args:NNx . . . . . 95, 1250, 1273	1871, 1913, 2344, 2350, 2370, 2376
\exp_args:Nnx . . . . . 295	\int_compare_p:nNn . . . . .
\exp_args:Nx . . . . . 252	1355, 1363, 2033, 2044, 2125
\exp_args:Nxx . . . . .	\int_eval:n . . . . . 90
1167, 1221, 2340, 2362, 2366	\int_incr:N . . . . . 1820, 1860,
\exp_not:N . . . . .	1862, 1876, 1878, 1882, 1884, 1973
1737, 1740, 1760, 1763, 1766,	\int_new:N . . . . .
1982, 1985, 1988, 2000, 2002, 2005,	1054, 1055, 1400, 1401, 1402, 1403
	\int_set:Nn . . . 1340, 1342, 1346, 1349

\int_use:N	33, 35, 50	2920, 2921, 2923, 2924, 2926, 2928,
\int_zero:N	1330, 1331, 1437, 1438, 1439,	3103, 3104, 3106, 3107, 3109, 3111
1440, 1819, 1821, 1822, 1968, 1969		
iow commands:		<b>P</b>
\iow_char:N	103, 109, 120, 125, 126, 135, 145	\PackageError ..... 7
\iow_newline:	144, 148	\pagenumbering ..... 6
		prg commands:
<b>K</b>		\prg_generate_conditional_-
keys commands:		variant:Nnn ..... 393, 408
\keys_define:nn	26, 323, 335, 352, 366, 442, 470,	\prg_new_protected_conditional:Npnn
496, 520, 548, 555, 567, 592, 601,		..... 381, 396, 411
616, 625, 633, 666, 673, 689, 707,		\prg_return_false: .....
742, 780, 813, 816, 823, 833, 844,		..... 389, 391, 404, 406, 416
855, 866, 886, 898, 933, 945, 966, 989		\prg_return_true: ..... 388, 403, 415
\keys_set:nn	26, 29, 260, 838, 872, 881, 929, 1013	\ProcessKeysOptions ..... 873
keyval commands:		prop commands:
\keyval_parse:nnn	446, 500	\prop_get:NnN ..... 2293
		\prop_get:NnNTF .....
<b>L</b>		..... 245, 383, 386, 398, 401, 413,
\labelformat	3	925, 2055, 2073, 2078, 2277, 2319, 2329
\language name	20, 726	\prop_gput:Nnn .... 220, 230, 311, 319
		\prop_gput_if_new:Nnn ..... 295, 303
<b>M</b>		\prop_gset_from_keyval:Nn ..... 419
\mainbabelname	20, 733	\prop_if_exist:NTF ..... 257, 878
\MessageBreak	10	\prop_if_exist_p:N ..... 2284, 2325
msg commands:		\prop_if_in:NnTF ..... 25, 210, 228
\msg_info:nnn	343, 373	\prop_if_in_p:Nn ..... 60, 2288
\msg_line_context:	102, 108, 139, 153, 157, 159, 161, 163	\prop_item:Nn ... 27, 61, 213, 217, 231
\msg_new:nnn	100, 106, 111, 113,	\prop_new:N .....
115, 117, 122, 128, 130, 132, 137,		..... 205, 258, 418, 441, 495, 851, 879
142, 147, 149, 151, 156, 158, 160, 162		\prop_put:Nnn ..... 439, 862, 913
\msg_note:nnn	264	\prop_remove:Nn ..... 438, 861, 905
\msg_warning:nn	534, 559, 663, 669, 826, 847	\providecommand ..... 3
\msg_warning:nnn	214, 233, 270, 278, 502, 890,	\ProvidesExplPackage ..... 14
931, 957, 996, 1533, 1690, 2067, 2096		
\msg_warning:nnnn	216, 448, 1234, 1257, 1280, 1318	<b>R</b>
		\refstepcounter ..... 3
<b>N</b>		\RequirePackage ... 16, 17, 18, 19, 20, 659
\newcounter	4	
\NewDocumentCommand	208, 224, 871, 876, 923, 1008, 1040	<b>S</b>
\newtheorem	59	seq commands:
\nobreakspace	421,	\seq_clear:N ..... 612, 1076
2391, 2392, 2394, 2395, 2397, 2399,		\seq_const_from_clist:Nn 164, 172, 185
2578, 2579, 2581, 2582, 2584, 2586,		\seq_gconcat:NNN ..... 198, 202
2751, 2752, 2754, 2755, 2757, 2759,		\seq_get_left:NN ..... 1461
		\seq_gput_right:Nn ..... 262
		\seq_if_empty:NTF ..... 1456
		\seq_if_in:NnTF ..... 248, 476, 1064
		\seq_map_break:n ..... 81, 1375, 1378
		\seq_map_function:NN ..... 1079
		\seq_map_indexed_inline:Nn . 18, 1335
		\seq_map_inline:Nn ..... 332, 349,
		363, 852, 883, 895, 942, 963, 986, 1372
		\seq_map_tokens:Nn ..... 63

<code>\seq_new:N</code> . . . . .	197,	<code>\zref@ifpropundefined</code> . . . . .	16
	201, 238, 469, 600, 1038, 1073, 1395	<code>\zref@ifrefcontainsprop</code> .	16, 1742,
<code>\seq_pop_left:NN</code> . . . . .	1455		1977, 1990, 2139, 2145, 2212, 2229
<code>\seq_put_right:Nn</code> . . . . .	478, 1068	<code>\zref@ifrefundefined</code> . . . . .	
<code>\seq_reverse:N</code> . . . . .	606		1084, 1086, 1098, 1488, 1490, 1495,
<code>\seq_set_eq:NN</code> . . . . .	1431		1528, 1687, 1696, 1838, 2023, 2134
<code>\seq_set_from_clist:Nn</code> . . . .	605, 1014	<code>\ZREF@mainlist</code> 22, 32, 34, 36, 87, 88, 99	
<code>\seq_sort:Nn</code> . . . . .	1082	<code>\zref@newprop</code> 5, 21, 23, 33, 35, 83, 85, 98	
sort commands:		<code>\zref@refused</code> . . . . .	1527
<code>\sort_return_same:</code> . . . . .		<code>\zref@wrapper@babel</code> . . . . .	29, 1009
	. . . . . 32, 38, 1089, 1094, 1131,	<code>\textendash</code> . . . . .	429
	1176, 1178, 1231, 1237, 1254, 1260,	<code>\the</code> . . . . .	3
	1283, 1314, 1321, 1359, 1375, 1391	<code>\thechapter</code> . . . . .	59
<code>\sort_return_swapped:</code> . . . . .		<code>\thepage</code> . . . . .	6, 95
	. . . . . 32, 38, 1102, 1140, 1175,	<code>\thesection</code> . . . . .	59
	1230, 1277, 1313, 1367, 1378, 1390	tl commands:	
str commands:		<code>\c_empty_tl</code> . . . . .	1061, 1120, 1122,
<code>\str_case:nnTF</code> . . . . .	748, 784		1186, 1190, 1194, 1198, 1472, 1477
<code>\str_if_eq:nnTF</code> . . . . .	80, 212	<code>\c_novalue_tl</code> . . . . .	857, 900
<code>\str_if_eq_p:nn</code> . . . . .	2110, 2116, 2118, 2122	<code>\tl_clear:N</code> . . . . .	259,
<code>\str_new:N</code> . . . . .	672		328, 928, 938, 1204, 1205, 1432,
<code>\str_set:Nn</code> . . . . .	677, 679, 681, 683		1433, 1434, 1435, 1436, 1458, 1815,
			1816, 1817, 1818, 1859, 2024, 2027,
			2051, 2066, 2095, 2310, 2331, 2333
		<code>\tl_gset:Nn</code> . . . . .	95
		<code>\tl_head:N</code> . . . . .	
			. . . . . 1209, 1211, 1294, 1296, 1310, 1312
		<code>\tl_if_empty:NnTF</code> . . . . .	
			. . . . . 71, 340, 357, 371, 950, 971,
			994, 1062, 1531, 1685, 2038, 2053, 2200
		<code>\tl_if_empty:nTF</code> . . . . .	
			. . . . . 226, 327, 437, 937, 1637, 1652,
			1667, 1908, 1933, 1944, 1957, 2026
		<code>\tl_if_empty_p:N</code> . . . . .	1128, 1129, 1137,
			1138, 1145, 1146, 1502, 1503, 1510,
			1511, 2109, 2119, 2123, 2282, 2323
		<code>\tl_if_empty_p:n</code> . . . . .	1217, 1218, 1245, 1268
		<code>\tl_if_eq:NnTF</code> . . . . .	1152, 1291, 1514
		<code>\tl_if_eq:NnTF</code> . . . . .	1077, 1110,
			1345, 1348, 1374, 1377, 1463, 2338
		<code>\tl_if_eq:nnTF</code> . . . . .	
			. . . . . 1167, 1221, 1337, 2340, 2362, 2366
		<code>\tl_if_in:NnTF</code> . . . . .	1250, 1273
		<code>\tl_if_novalue:nTF</code> . . . . .	860, 903
		<code>\tl_map_break:n</code> . . . . .	81
		<code>\tl_map_tokens:Nn</code> . . . . .	73
		<code>\tl_new:N</code> . . . . .	
			. . . . . 89, 236, 237, 239, 519, 719, 720,
			721, 812, 815, 874, 875, 1046, 1047,
			1048, 1049, 1050, 1051, 1052, 1053,
			1396, 1397, 1398, 1399, 1404, 1408,
			1409, 1410, 1412, 1413, 1414, 1415,
			1416, 1417, 1418, 1419, 1420, 1421,
			1422, 1423, 1424, 1425, 1427, 1428

## T

TeX and L<sup>A</sup>T<sub>ε</sub>X commands:

<code>\@Alph</code> . . . . .	59
<code>\@addtoreset</code> . . . . .	4
<code>\@chapapp</code> . . . . .	59
<code>\@currentcounter</code> . . . . .	
	. . . . . 4, 21, 25, 28, 30, 33, 84, 86
<code>\@currentlabel</code> . . . . .	3
<code>\@ifl@t@r</code> . . . . .	3
<code>\@ifpackageloaded</code> . . . . .	
	. . . . . 531, 546, 656, 724, 730, 830
<code>\bbl@loaded</code> . . . . .	21
<code>\bbl@main@language</code> . . . . .	20, 727
<code>\c@</code> . . . . .	4
<code>\c@page</code> . . . . .	6, 94
<code>\cl@</code> . . . . .	4, 5
<code>\hyper@link</code> . . . . .	1740, 1988, 2143, 2227
<code>\p@...</code> . . . . .	3
<code>\zref@addprop</code> . . . . .	22, 32, 34, 36, 87, 88, 99
<code>\zref@default</code> . . . . .	
	. . . . . 50, 2018, 2135, 2196, 2202, 2269
<code>\zref@extractdefault</code> . . . . .	
	. . . . . 50, 1061, 1120, 1122, 1168,
	1169, 1172, 1174, 1186, 1190, 1194,
	1198, 1222, 1223, 1227, 1229, 1249,
	1272, 1387, 1389, 1471, 1476, 1745,
	1750, 1756, 1991, 1992, 1994, 1997,
	2011, 2148, 2152, 2157, 2172, 2232,
	2236, 2241, 2246, 2261, 2341, 2342,
	2345, 2347, 2351, 2353, 2363, 2364,
	2367, 2368, 2371, 2373, 2377, 2379

<code>\tl_put_left:Nn</code> . . . .	1719, 1726, 1775	<code>\l__zrefclever_counter_resetby_</code> prop . . . . .	5, 15, 60, 61, 495, 507
<code>\tl_put_right:Nn</code> . . . .	1587, 1603, 1612, 1639, 1649, 1664, 1895, 1906, 1931, 1942, 1955, 2039, 2040, 2049	<code>\l__zrefclever_counter_resettters_</code> seq . . . . .	4, 5, 15, 63, 469, 476, 479
<code>\tl_reverse_items:n</code> . . . . .	1057, 1188, 1192, 1196, 1200	<code>\l__zrefclever_counter_type_prop</code> . . . . .	3, 14, 25, 28, 441, 453
<code>\tl_set:Nn</code> . . . .	329, 524, 526, 532, 535, 551, 560, 726, 727, 732, 733, 736, 737, 740, 752, 760, 767, 788, 796, 803, 880, 939, 1060, 1119, 1121, 1185, 1187, 1189, 1191, 1193, 1195, 1197, 1199, 1208, 1210, 1248, 1271, 1298, 1300, 1302, 1304, 1465, 1466, 1469, 1474, 1578, 1579, 1702, 1733, 1851, 1852, 1875, 2035, 2036, 2048	<code>\l__zrefclever_current_language_</code> tl . . . .	20, 21, 721, 726, 732, 736, 761, 797
<code>\tl_set_eq:NN</code> . . . . .	1813	<code>\__zrefclever_declare_default_</code> transl:nnn . . . .	317, 322, 952, 973
<code>\tl_tail:N</code> . . . .	1299, 1301, 1303, 1305	<code>\__zrefclever_declare_type_</code> transl:nnnn . . . .	309, 314, 978, 1000
<code>\l_tmpa_tl</code> . . . . .	1204, 1208, 1217, 1245, 1248, 1251, 1291	<code>\l__zrefclever_dict_file_tl</code> . . . .	239, 255, 261
<code>\l_tmpb_tl</code> . . . . .	1205, 1210, 1218, 1268, 1271, 1274, 1291	<code>\l__zrefclever_dict_language_tl</code> . . . .	236, 246, 250, 253, 257, 258, 263, 265, 271, 296, 304, 384, 386, 399, 401
U		<code>\l__zrefclever_dict_type_tl</code> . . . .	236, 259, 297, 328, 329, 340, 357, 371
use commands:		<code>\g__zrefclever_fallback_dict_</code> prop . . . . .	413, 418, 419
<code>\use:N</code> . . . . .	21	<code>\__zrefclever_get_default_</code> transl:nnN . . . . .	396, 408
Z		<code>\__zrefclever_get_default_</code> transl:nnNTF . . . . .	2304
<code>\zcDeclareLanguage</code> . . . . .	208	<code>\__zrefclever_get_enclosing_</code> counters:n . . . . .	5, 37, 42, 84
<code>\zcDeclareLanguageAlias</code> . . . . .	224	<code>\__zrefclever_get_enclosing_</code> counters_value:n . . . .	5, 37, 51, 86
<code>\zcDeclareTranslations</code> . . . .	25–27, 923	<code>\__zrefclever_get_fallback_</code> transl:nN . . . . .	411
<code>\zcpageref</code> . . . . .	30, 1040	<code>\__zrefclever_get_fallback_</code> transl:nNTF . . . . .	2309
<code>\zcRef</code> 24, 25, 29, 31, 38, 40, 1008, 1043, 1044		<code>\__zrefclever_get_option_</code> plain:nN . . . . .	24, 25, 1542, 1543, 1544, 2316
<code>\zcRefTypeSetup</code> . . . . .	25, 26, 876	<code>\__zrefclever_get_option_with_</code> transl:nN . . . . .	13, 24, 25, 59, 1443, 1444, 1445, 1446, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 2273
<code>\zcsetup</code> . . . . .	21, 24, 25, 871	<code>\__zrefclever_get_ref:n</code> 1590, 1606, 1618, 1622, 1642, 1655, 1658, 1670, 1673, 1707, 1727, 1898, 1911, 1916, 1936, 1947, 1950, 1960, 1963, 1975	
zrefcheck commands:		<code>\__zrefclever_get_ref_first:</code> . . . .	40, 50, 1720, 1776, 2132
<code>\zrefcheck_zcRef_beg_label:</code> . .	1020	<code>\__zrefclever_get_type_transl:nnnN</code> . . . . .	381, 393
<code>\zrefcheck_zcRef_end_label_</code> maybe: . . . . .	1032	<code>\__zrefclever_get_type_transl:nnnNTF</code> . . . . .	2060, 2083, 2089, 2298
<code>\zrefcheck_zcRef_run_checks_on_</code> labels:n . . . . .	1033	<code>\l__zrefclever_label_a_tl</code> . . . . .	1046, 1455, 1472, 1488, 1527, 1528, 1534, 1578, 1590, 1606, 1622,
zrefclever internal commands:			
<code>\l__zrefclever_abbrev_bool</code> . . . .	705, 709, 2042		
<code>\l__zrefclever_capitalize_bool</code> . .	687, 691, 2030		
<code>\l__zrefclever_capitalize_first_</code> bool . . . . .	688, 697, 2032		
<code>\__zrefclever_counter_reset_by:n</code> . . . . .	5, 6, 15, 39, 41, 43, 48, 50, 52, 57		
<code>\__zrefclever_counter_reset_by_</code> aux:nn . . . . .	64, 67		
<code>\__zrefclever_counter_reset_by_</code> auxi:nnn . . . . .	74, 78		



1658, 1673, 1700, 1707, 1838, 1842,  
 1851, 1875, 1898, 1916, 1950, 1963  
 \l\_zrefclever\_label\_b\_tl . 1046,  
 1458, 1461, 1477, 1490, 1495, 1842  
 \l\_zrefclever\_label\_count\_int ..  
 ..... 39, 1400,  
 1437, 1540, 1572, 1819, 1847, 1973  
 \l\_zrefclever\_label\_enclcnt\_a\_-  
 tl ..... 1046, 1185,  
 1187, 1188, 1209, 1274, 1298, 1299  
 \l\_zrefclever\_label\_enclcnt\_b\_-  
 tl ..... 1046, 1189,  
 1191, 1192, 1211, 1251, 1300, 1301  
 \l\_zrefclever\_label\_enclval\_a\_-  
 tl ..... 1046, 1193,  
 1195, 1196, 1294, 1302, 1303, 1310  
 \l\_zrefclever\_label\_enclval\_b\_-  
 tl ..... 1046, 1197,  
 1199, 1200, 1296, 1304, 1305, 1312  
 \l\_zrefclever\_label\_type\_a\_tl ..  
 ..... 1046, 1060, 1062, 1066,  
 1069, 1119, 1128, 1137, 1145, 1153,  
 1345, 1374, 1465, 1469, 1502, 1510,  
 1515, 1531, 1579, 1852, 2282, 2285,  
 2289, 2294, 2300, 2323, 2326, 2330  
 \l\_zrefclever\_label\_type\_b\_tl ..  
 ..... 1046,  
 1121, 1129, 1138, 1146, 1154, 1348,  
 1377, 1466, 1474, 1503, 1511, 1515  
 \\_zrefclever\_label\_type\_put\_-  
 new\_right:n ..... 31, 1058, 1080  
 \l\_zrefclever\_label\_types\_seq ..  
 .... 31, 1065, 1068, 1073, 1076, 1372  
 \\_zrefclever\_labels\_in\_sequence:nn  
 ..... 1699, 1841, 2336  
 \g\_zrefclever\_language\_aliases\_-  
 prop ..... 205, 210, 213, 217,  
 220, 228, 230, 231, 245, 383, 398, 925  
 \l\_zrefclever\_last\_of\_type\_bool  
 ..... 38, 1393, 1486, 1491, 1492,  
 1496, 1505, 1516, 1517, 1520, 1558  
 \l\_zrefclever\_lastsep\_tl . 1416,  
 1549, 1605, 1621, 1641, 1657, 1669  
 \l\_zrefclever\_link\_star\_bool ...  
 ..... 1015, 1038, 1980, 2108, 2218  
 \l\_zrefclever\_listsep\_tl .....  
 ... 1415, 1548, 1617, 1654, 1897,  
 1910, 1915, 1935, 1946, 1949, 1959  
 \l\_zrefclever\_load\_dict\_-  
 verbose\_bool ... 240, 268, 277, 286  
 \g\_zrefclever\_loaded\_dictionaries\_-  
 seq ..... 238, 249, 262  
 \l\_zrefclever\_main\_language\_tl .  
 ..... 20,  
 21, 720, 727, 733, 737, 740, 753, 789  
 \l\_zrefclever\_name\_format\_-  
 fallback\_tl .....  
 .. 1425, 2048, 2051, 2053, 2080, 2092  
 \l\_zrefclever\_name\_format\_tl ...  
 ... 1425, 2035, 2036, 2039, 2040,  
 2048, 2049, 2057, 2063, 2075, 2086  
 \l\_zrefclever\_name\_in\_link\_bool  
 .. 1425, 1735, 2112, 2128, 2129, 2137  
 \l\_zrefclever\_namefont\_tl 1408,  
 1542, 1738, 1764, 2162, 2192, 2207  
 \l\_zrefclever\_nameinlink\_str ...  
 ..... 672, 677,  
 679, 681, 683, 2110, 2116, 2118, 2122  
 \l\_zrefclever\_namesep\_tl .....  
 .. 1412, 1545, 2165, 2195, 2203, 2210  
 \l\_zrefclever\_next\_is\_same\_bool  
 ..... 39, 58, 1402,  
 1831, 1861, 1877, 1883, 2356, 2382  
 \l\_zrefclever\_next\_maybe\_range\_-  
 bool .....  
 .. 39, 58, 1402, 1695, 1704, 1830,  
 1857, 1868, 2348, 2355, 2374, 2381  
 \l\_zrefclever\_noabbrev\_first\_-  
 bool ..... 706, 715, 2045  
 \l\_zrefclever\_noteseptl .....  
 ..... 1026, 1420, 1446  
 \\_zrefclever\_page\_format\_aux: ..  
 ..... 90, 94  
 \g\_zrefclever\_page\_format\_tl ...  
 ..... 7, 89, 95, 98  
 \l\_zrefclever\_pairsep\_tl .....  
 ..... 1414, 1547, 1589, 1705  
 \\_zrefclever\_prop\_put\_non\_-  
 empty:Nnn ..... 14, 435, 452, 506  
 \\_zrefclever\_provide\_dict\_-  
 default\_transl:nn .. 301, 341, 358  
 \\_zrefclever\_provide\_dict\_type\_-  
 transl:nn ..... 293, 359, 376  
 \\_zrefclever\_provide\_dictionary:n  
 ..... 29, 243, 281, 287, 779, 1016  
 \\_zrefclever\_provide\_dictionary\_-  
 verbose:n ..... 283,  
 290, 754, 762, 768, 790, 798, 804  
 \l\_zrefclever\_range\_beg\_label\_-  
 tl ..... 39, 1402, 1436,  
 1618, 1637, 1642, 1652, 1655, 1667,  
 1670, 1818, 1859, 1875, 1908, 1911,  
 1933, 1936, 1944, 1947, 1957, 1960  
 \l\_zrefclever\_range\_count\_int ..  
 ..... 39,  
 1402, 1439, 1598, 1630, 1821, 1860,  
 1872, 1876, 1882, 1890, 1927, 1968

\l__zrefclever_range_inhibit_- next_bool . . . . .	38, 39, <a href="#">1402</a> , <a href="#">1836</a>	\l__zrefclever_sort_decided_bool . . . . .	<a href="#">1056</a> , <a href="#">1202</a> , <a href="#">1206</a> , <a href="#">1225</a> , <a href="#">1236</a> , <a href="#">1253</a> , <a href="#">1259</a> , <a href="#">1276</a> , <a href="#">1282</a> , <a href="#">1308</a> , <a href="#">1320</a>
\l__zrefclever_range_same_count_- int . . . . .	39, <a href="#">1402</a> , <a href="#">1440</a> , <a href="#">1585</a> , <a href="#">1619</a> , <a href="#">1630</a> , <a href="#">1822</a> , <a href="#">1862</a> , <a href="#">1878</a> , <a href="#">1884</a> , <a href="#">1913</a> , <a href="#">1927</a> , <a href="#">1969</a>	\__zrefclever_sort_default:nn . . . . .	<a href="#">31</a> , <a href="#">32</a> , <a href="#">1112</a> , <a href="#">1117</a>
\l__zrefclever_rangesep_tl . . . . .	<a href="#">1413</a> , <a href="#">1546</a> , <a href="#">1672</a> , <a href="#">1706</a> , <a href="#">1962</a>	\__zrefclever_sort_default_- different_types:nn . . . . .	<a href="#">18</a> , <a href="#">1161</a> , <a href="#">1328</a>
\l__zrefclever_ref_language_tl . . . . .	<a href="#">20</a> , <a href="#">21</a> , <a href="#">719</a> , <a href="#">740</a> , <a href="#">752</a> , <a href="#">755</a> , <a href="#">760</a> , <a href="#">763</a> , <a href="#">767</a> , <a href="#">769</a> , <a href="#">779</a> , <a href="#">788</a> , <a href="#">791</a> , <a href="#">796</a> , <a href="#">799</a> , <a href="#">803</a> , <a href="#">805</a> , <a href="#">1016</a> , <a href="#">2061</a> , <a href="#">2084</a> , <a href="#">2090</a> , <a href="#">2299</a> , <a href="#">2305</a>	\__zrefclever_sort_default_same_- type:nn . . . . .	<a href="#">1157</a> , <a href="#">1183</a>
\c__zrefclever_ref_options_- necessarily_not_type_specific_- seq . . . . .	<a href="#">165</a> , <a href="#">203</a> , <a href="#">333</a> , <a href="#">884</a> , <a href="#">943</a>	\__zrefclever_sort_labels: . . . . .	<a href="#">31</a> , <a href="#">32</a> , <a href="#">38</a> , <a href="#">1024</a> , <a href="#">1074</a>
\c__zrefclever_ref_options_- necessarily_type_specific_seq . . . . .	<a href="#">186</a> , <a href="#">200</a> , <a href="#">364</a> , <a href="#">987</a>	\__zrefclever_sort_page:nn . . . . .	<a href="#">38</a> , <a href="#">1111</a> , <a href="#">1384</a>
\c__zrefclever_ref_options_not_- type_specific_seq . . . . .	<a href="#">201</a> , <a href="#">202</a> , <a href="#">853</a>	\l__zrefclever_sort_prior_a_int . . . . .	<a href="#">1054</a> , <a href="#">1330</a> , <a href="#">1339</a> , <a href="#">1340</a> , <a href="#">1346</a> , <a href="#">1356</a> , <a href="#">1364</a>
\c__zrefclever_ref_options_- possibly_type_specific_seq . . . . .	<a href="#">173</a> , <a href="#">199</a> , <a href="#">204</a> , <a href="#">350</a> , <a href="#">964</a>	\l__zrefclever_sort_prior_b_int . . . . .	<a href="#">1055</a> , <a href="#">1331</a> , <a href="#">1341</a> , <a href="#">1342</a> , <a href="#">1349</a> , <a href="#">1357</a> , <a href="#">1365</a>
\l__zrefclever_ref_options_prop . . . . .	<a href="#">24</a> , <a href="#">26</a> , <a href="#">851</a> , <a href="#">861</a> , <a href="#">862</a> , <a href="#">2277</a> , <a href="#">2319</a>	\l__zrefclever_tlastsep_tl . . . . .	<a href="#">1419</a> , <a href="#">1445</a> , <a href="#">1807</a>
\c__zrefclever_ref_options_type_- specific_seq . . . . .	<a href="#">197</a> , <a href="#">198</a> , <a href="#">896</a>	\l__zrefclever_tlistsep_tl . . . . .	<a href="#">1418</a> , <a href="#">1444</a> , <a href="#">1785</a>
\l__zrefclever_ref_property_tl . . . . .	<a href="#">16</a> , <a href="#">519</a> , <a href="#">524</a> , <a href="#">526</a> , <a href="#">532</a> , <a href="#">535</a> , <a href="#">551</a> , <a href="#">560</a> , <a href="#">1077</a> , <a href="#">1110</a> , <a href="#">1463</a> , <a href="#">1977</a> , <a href="#">1997</a> , <a href="#">2011</a> , <a href="#">2140</a> , <a href="#">2173</a> , <a href="#">2213</a> , <a href="#">2247</a> , <a href="#">2262</a> , <a href="#">2338</a>	\l__zrefclever_tpairsep_tl . . . . .	<a href="#">1417</a> , <a href="#">1443</a> , <a href="#">1801</a>
\l__zrefclever_ref_typeset_font_- tl . . . . .	<a href="#">812</a> , <a href="#">814</a> , <a href="#">1449</a>	\l__zrefclever_type_<type>- options_prop . . . . .	<a href="#">26</a>
\l__zrefclever_reffont_in_tl <a href="#">1410</a> ,	<a href="#">1544</a> , <a href="#">1986</a> , <a href="#">2009</a> , <a href="#">2170</a> , <a href="#">2225</a> , <a href="#">2259</a>	\l__zrefclever_type_count_int . . . . .	<a href="#">39</a> , <a href="#">1400</a> , <a href="#">1438</a> , <a href="#">1782</a> , <a href="#">1784</a> , <a href="#">1793</a> , <a href="#">1820</a> , <a href="#">2033</a> , <a href="#">2044</a> , <a href="#">2125</a>
\l__zrefclever_reffont_out_tl . . . . .	<a href="#">1409</a> , <a href="#">1543</a> , <a href="#">1983</a> , <a href="#">2006</a> , <a href="#">2167</a> , <a href="#">2186</a> , <a href="#">2222</a> , <a href="#">2256</a>	\l__zrefclever_type_first_label_- tl . . . . .	<a href="#">1395</a> , <a href="#">1434</a> , <a href="#">1578</a> , <a href="#">1687</a> , <a href="#">1696</a> , <a href="#">1700</a> , <a href="#">1727</a> , <a href="#">1743</a> , <a href="#">1746</a> , <a href="#">1751</a> , <a href="#">1757</a> , <a href="#">1816</a> , <a href="#">1851</a> , <a href="#">2023</a> , <a href="#">2134</a> , <a href="#">2140</a> , <a href="#">2146</a> , <a href="#">2148</a> , <a href="#">2152</a> , <a href="#">2157</a> , <a href="#">2172</a> , <a href="#">2213</a> , <a href="#">2230</a> , <a href="#">2232</a> , <a href="#">2236</a> , <a href="#">2241</a> , <a href="#">2246</a> , <a href="#">2261</a>
\l__zrefclever_refpos_in_tl <a href="#">1424</a> ,	<a href="#">1553</a> , <a href="#">1998</a> , <a href="#">2012</a> , <a href="#">2174</a> , <a href="#">2248</a> , <a href="#">2263</a>	\l__zrefclever_type_first_label_- type_tl . . . . .	<a href="#">1395</a> , <a href="#">1435</a> , <a href="#">1579</a> , <a href="#">1691</a> , <a href="#">1817</a> , <a href="#">1852</a> , <a href="#">2026</a> , <a href="#">2056</a> , <a href="#">2062</a> , <a href="#">2068</a> , <a href="#">2074</a> , <a href="#">2079</a> , <a href="#">2085</a> , <a href="#">2091</a> , <a href="#">2097</a>
\l__zrefclever_refpos_out_tl <a href="#">1422</a> ,	<a href="#">1551</a> , <a href="#">2001</a> , <a href="#">2014</a> , <a href="#">2187</a> , <a href="#">2251</a> , <a href="#">2265</a>	\__zrefclever_type_name_setup: . . . . .	<a href="#">40</a> , <a href="#">1715</a> , <a href="#">2021</a>
\l__zrefclever_refpre_in_tl <a href="#">1423</a> ,	<a href="#">1552</a> , <a href="#">1996</a> , <a href="#">2010</a> , <a href="#">2171</a> , <a href="#">2245</a> , <a href="#">2260</a>	\l__zrefclever_type_name_tl . . . . .	<a href="#">51</a> , <a href="#">1425</a> , <a href="#">1759</a> , <a href="#">1765</a> , <a href="#">2024</a> , <a href="#">2027</a> , <a href="#">2058</a> , <a href="#">2064</a> , <a href="#">2066</a> , <a href="#">2076</a> , <a href="#">2081</a> , <a href="#">2087</a> , <a href="#">2093</a> , <a href="#">2095</a> , <a href="#">2109</a> , <a href="#">2163</a> , <a href="#">2193</a> , <a href="#">2200</a> , <a href="#">2208</a>
\l__zrefclever_refpre_out_tl <a href="#">1421</a> ,	<a href="#">1550</a> , <a href="#">1984</a> , <a href="#">2007</a> , <a href="#">2168</a> , <a href="#">2223</a> , <a href="#">2257</a>	\l__zrefclever_typeset_compress_- bool . . . . .	<a href="#">615</a> , <a href="#">618</a> , <a href="#">1833</a>
\l__zrefclever_setup_language_tl . . . . .	<a href="#">874</a> , <a href="#">926</a> , <a href="#">953</a> , <a href="#">974</a> , <a href="#">979</a> , <a href="#">1001</a>	\l__zrefclever_typeset_labels_- seq . . . . .	<a href="#">1395</a> , <a href="#">1431</a> , <a href="#">1455</a> , <a href="#">1456</a> , <a href="#">1461</a>
\l__zrefclever_setup_type_tl . . . . .	<a href="#">874</a> , <a href="#">880</a> , <a href="#">908</a> , <a href="#">916</a> , <a href="#">928</a> , <a href="#">938</a> , <a href="#">939</a> , <a href="#">950</a> , <a href="#">971</a> , <a href="#">980</a> , <a href="#">994</a> , <a href="#">1002</a>	\l__zrefclever_typeset_last_bool . . . . .	<a href="#">38</a> , <a href="#">1393</a> , <a href="#">1452</a> , <a href="#">1453</a> , <a href="#">1459</a> , <a href="#">1485</a> , <a href="#">1790</a> , <a href="#">2124</a>

\l__zrefclever_typeset_name_bool	\l__zrefclever_typeset_sort_bool
..... 566, 573, 578, 583, 1717, 1731	..... 591, 594, 1022
\l__zrefclever_typeset_queue_-	\l__zrefclever_typesort_seq ....
curr_tl ..... 1395, 1433, 1587,	..... 18, 600, 605, 606, 612, 1335
1603, 1612, 1639, 1649, 1664, 1685,	\l__zrefclever_use_hyperref_bool
1702, 1719, 1726, 1733, 1776, 1797,	..... 631, 638,
1802, 1808, 1814, 1815, 1895, 1906,	643, 648, 658, 664, 1980, 2107, 2217
1931, 1942, 1955, 2038, 2119, 2123	\l__zrefclever_warn_hyperref_-
\l__zrefclever_typeset_queue_-	bool ..... 632, 639, 644, 649, 662
prev_tl .... 1395, 1432, 1786, 1814	\__zrefclever_zcref:nnn .. 1009, 1010
\l__zrefclever_typeset_range_-	\__zrefclever_zcref:nnnn 29, 31, 1010
bool ..... 624, 627, 1023, 1683	\l__zrefclever_zcref_labels_seq .
\l__zrefclever_typeset_ref_bool .	31, 1014, 1034, 1038, 1079, 1082, 1431
..... 565, 572, 577, 582, 1717, 1724	\l__zrefclever_zcref_note_tl ...
\__zrefclever_typeset_refs: ....	..... 815, 818, 1027
..... 38, 39, 50, 51, 53, 1025, 1429	\l__zrefclever_zcref_with_check_-
\__zrefclever_typeset_refs_aux_-	bool ..... 822, 837, 1019, 1030
last_of_type: ..... 1561, 1569	\l__zrefclever_zrefcheck_-
\__zrefclever_typeset_refs_aux_-	available_bool .....
not_last_of_type: .... 1565, 1825	..... 821, 832, 843, 1018, 1029