

# 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>2</b>  |
| <b>3</b> | <b>zref setup</b>                | <b>2</b>  |
| <b>4</b> | <b>Plumbing</b>                  | <b>7</b>  |
| 4.1      | Messages . . . . .               | 7         |
| 4.2      | Reference format . . . . .       | 8         |
| 4.3      | Languages . . . . .              | 10        |
| 4.4      | Dictionaries . . . . .           | 10        |
| 4.5      | Options . . . . .                | 14        |
| <b>5</b> | <b>User interface</b>            | <b>25</b> |
| 5.1      | \zcsetup . . . . .               | 25        |
| 5.2      | \zcRefTypeSetup . . . . .        | 25        |
| 5.3      | \zcDeclareTranslations . . . . . | 26        |
| 5.4      | \zceref . . . . .                | 29        |
| 5.5      | \zcpageref . . . . .             | 30        |
| <b>6</b> | <b>Sorting</b>                   | <b>30</b> |
| <b>7</b> | <b>Typesetting</b>               | <b>38</b> |
| <b>8</b> | <b>Special handling</b>          | <b>59</b> |
| 8.1      | \appendix . . . . .              | 59        |
| 8.2      | \newtheorem . . . . .            | 59        |
| 8.3      | enumitem package . . . . .       | 59        |
| <b>9</b> | <b>Dictionaries</b>              | <b>59</b> |
| 9.1      | English . . . . .                | 59        |
| 9.2      | German . . . . .                 | 63        |
| 9.3      | French . . . . .                 | 66        |
| 9.4      | Portuguese . . . . .             | 69        |
| 9.5      | Spanish . . . . .                | 73        |

---

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

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

## 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-abspace` provides the `abspace` 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 ‘ltxcounts.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 `\cl@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

```
\__zrefclever_get_enclosing_counters: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 `\counter`.

```

    \_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 { c1@ #2 }
72     {
73       \tl_map_tokens:cn { c1@ #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 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 `\zcref`, but just as well in `\zcsetup` or as package options at load-time (see Section 4.5). “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.

|   |  |
|---|--|
| <code>\l__zrefclever_setup_type_tl</code><br><code>\l__zrefclever_dict_language_tl</code> | Store “current” type and language in different places for option and translation handling, notably in <code>\__zrefclever_provide_dictionary:n</code> , <code>\zcRefTypeSetup</code> , and |
|---|--|



\zcDeclareTranslations. But also for translations retrieval, in \\_\_zrefclever\_get\_type\_transl:nnnN and \\_\_zrefclever\_get\_default\_transl:nnN.

```
164 \tl_new:N \l__zrefclever_setup_type_tl
165 \tl_new:N \l__zrefclever_dict_language_tl
```

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

f\_options\_necessarily\_not\_type\_specific\_seq  
ever\_ref\_options\_possibly\_type\_specific\_seq  
r\_ref\_options\_necessarily\_type\_specific\_seq  
c\_zrefclever\_ref\_options\_type\_specific\_seq  
zrefclever\_ref\_options\_not\_type\_specific\_seq

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

```
166 \seq_const_from_clist:Nn
167   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
168   {
169     tpairsep ,
170     tlistsep ,
171     tlastsep ,
172     notesep ,
173   }
174 \seq_const_from_clist:Nn
175   \c__zrefclever_ref_options_possibly_type_specific_seq
176   {
177     namesep ,
178     pairsep ,
179     listsep ,
180     lastsep ,
181     rangesep ,
182     refpre ,
183     refpos ,
184     refpre-in ,
185     refpos-in ,
186   }
187 \seq_const_from_clist:Nn
188   \c__zrefclever_ref_options_necessarily_type_specific_seq
189   {
190     Name-sg ,
191     name-sg ,
192     Name-pl ,
193     name-pl ,
194     Name-sg-ab ,
195     name-sg-ab ,
196     Name-pl-ab ,
197     name-pl-ab ,
198   }
199 \seq_new:N \c__zrefclever_ref_options_type_specific_seq
200 \seq_gconcat:NNN \c__zrefclever_ref_options_type_specific_seq
201   \c__zrefclever_ref_options_possibly_type_specific_seq
202   \c__zrefclever_ref_options_necessarily_type_specific_seq
203 \seq_new:N \c__zrefclever_ref_options_not_type_specific_seq
204 \seq_gconcat:NNN \c__zrefclever_ref_options_not_type_specific_seq
205   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
206   \c__zrefclever_ref_options_possibly_type_specific_seq
```

(End definition for \c\_\_zrefclever\_ref\_options\_necessarily\_not\_type\_specific\_seq and others.)

### 4.3 Languages

```

207 \prop_new:N \g__zrefclever_language_aliases_prop
208
209 % {<base language>}
210 \NewDocumentCommand \zcDeclareLanguage { m }
211 {
212   \tl_if_empty:nF {#1}
213   {
214     \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#1}
215     {
216       \str_if_eq:eeTF {#1}
217       { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
218       { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
219       {
220         \msg_warning:nnxx { zref-clever } { alias-declared } {#1}
221         { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
222       }
223     }
224     { \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1} {#1} }
225   }
226 }
227 \@onlypreamble \zcDeclareLanguage
228
229 % {<alias>}{<base language>}
230 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
231 {
232   \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#2}
233   {
234     \exp_args:NNnx \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1}
235     { \prop_item:Nn \g__zrefclever_language_aliases_prop {#2} }
236   }
237   { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
238 }
239 \@onlypreamble \zcDeclareLanguageAlias

```

### 4.4 Dictionaries

```

240 \seq_new:N \g__zrefclever_loaded_dictionaries_seq
241 \bool_new:N \l__zrefclever_load_dict_verbose_bool
242
243 % {<language>}
244 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
245 {
246   \group_begin:
247   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
248   \l__zrefclever_dict_language_tl
249   {
250     \seq_if_in:NVF
251     \g__zrefclever_loaded_dictionaries_seq
252     \l__zrefclever_dict_language_tl
253     {
254       \tl_clear:N \l_tmpa_tl
255       \exp_args:Nx \file_get:nnNTF
256       { zref-clever- \l__zrefclever_dict_language_tl .dict }

```

```

257         { \ExplSyntaxOn }
258         \l_tmpa_tl
259         {
260             \prop_if_exist:cF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
261                 { \prop_new:c { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
262                     \tl_clear:N \l__zrefclever_setup_type_tl
263                     \exp_args:NnV
264                     \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
265                     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
266                     \l__zrefclever_dict_language_tl
267                     \msg_note:nnx { zref-clever } { dict-loaded }
268                     { \l__zrefclever_dict_language_tl }
269                 }
270             {
271                 \bool_if:NT \l__zrefclever_load_dict_verbose_bool
272                 {
273                     \msg_warning:nnx { zref-clever } { dict-not-available }
274                     { \l__zrefclever_dict_language_tl }
275                 }
276             }
277         }
278     }
279     {
280         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
281         { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
282     }
283     \group_end:
284 }
285 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }
286
287 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
288 {
289     \group_begin:
290     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
291     \__zrefclever_provide_dictionary:n {#1}
292     \group_end:
293 }
294 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }
295
296 % {<key>}{<translation>}
297 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
298 {
299     \exp_args:Nnx \prop_gput_if_new:cnn
300     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
301     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
302 }
303
304 % {<key>}{<translation>}
305 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
306 {
307     \prop_gput_if_new:cnn
308     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
309     { default- #1 } {#2}
310 }

```

```

311 \keys_define:nn { zref-clever / dictionary }
312 {
313   type .code:n =
314   {
315     \tl_if_empty:NTF {#1}
316     { \tl_clear:N \l__zrefclever_setup_type_tl }
317     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
318   } ,
319 }
320 \seq_map_inline:Nn
321 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
322 {
323   \keys_define:nn { zref-clever / dictionary }
324   {
325     #1 .value_required:n = true ,
326     #1 .code:n =
327     {
328       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
329       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
330       {
331         \msg_info:nnn { zref-clever }
332         { option-not-type-specific } {#1}
333       }
334     } ,
335   }
336 }
337 \seq_map_inline:Nn
338 \c__zrefclever_ref_options_possibly_type_specific_seq
339 {
340   \keys_define:nn { zref-clever / dictionary }
341   {
342     #1 .value_required:n = true ,
343     #1 .code:n =
344     {
345       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
346       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
347       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
348     } ,
349   }
350 }
351 \seq_map_inline:Nn
352 \c__zrefclever_ref_options_necessarily_type_specific_seq
353 {
354   \keys_define:nn { zref-clever / dictionary }
355   {
356     #1 .value_required:n = true ,
357     #1 .code:n =
358     {
359       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
360       {
361         \msg_info:nnn { zref-clever }
362         { option-only-type-specific } {#1}
363       }
364       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }

```

```

365     } ,
366   }
367 }
368 % {<language>}{<type>}{<key><tl var to set>
369 \prg_new_protected_conditional:Npnn \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
370 {
371   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
372   \l__zrefclever_dict_language_tl
373   {
374     \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
375     { type- #2 - #3 } #4
376     { \prg_return_true: }
377     { \prg_return_false: }
378   }
379   { \prg_return_false: }
380 }
381 \prg_generate_conditional_variant:Nnn \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }
382
383 % {<language>}{<key><tl var to set>
384 \prg_new_protected_conditional:Npnn \__zrefclever_get_default_transl:nnN #1#2#3 { F }
385 {
386   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
387   \l__zrefclever_dict_language_tl
388   {
389     \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
390     { default- #2 } #3
391     { \prg_return_true: }
392     { \prg_return_false: }
393   }
394   { \prg_return_false: }
395 }
396 \prg_generate_conditional_variant:Nnn \__zrefclever_get_default_transl:nnN { xnN } { F }
397
398 % {<key><tl var to set>
399 \prg_new_protected_conditional:Npnn \__zrefclever_get_fallback_transl:nN #1#2 { F }
400 {
401   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
402   { #1 } #2
403   { \prg_return_true: }
404   { \prg_return_false: }
405 }

```

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’.

```

406 \prop_new:N \g__zrefclever_fallback_dict_prop
407 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
408 {
409   namesep = {\nobreakspace} ,
410   pairsep = {,~} ,
411   listsep = {,~} ,
412   lastsep = {,~} ,
413   tpairsep = {,~} ,

```

```

414     tlistsep = {,~} ,
415     tlastsep = {,~} ,
416     notesep  = {~} ,
417     rangesep  = {\textendash} ,
418     refpre    = {} ,
419     refpos    = {} ,
420     refpre-in = {} ,
421     refpos-in = {} ,
422 }

```

## 4.5 Options

### Auxiliary

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

```

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

423 \cs_new_protected:Npn \_zrefclever_prop_put_non_empty:Nnn #1#2#3
424 {
425   \tl_if_empty:nTF {#3}
426     { \prop_remove:Nn #1 {#2} }
427     { \prop_put:Nnn #1 {#2} {#3} }
428 }

```

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

### 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`.

```

429 \prop_new:N \l_zrefclever_counter_type_prop
430 \keys_define:nn { zref-clever / label }
431 {
432   countertype .code:n =
433   {
434     \keyval_parse:nnn
435     {
436       \msg_warning:nnnn { zref-clever }
437       { key-requires-value } { countertype }
438     }
439     {
440       \_zrefclever_prop_put_non_empty:Nnn
441       \l_zrefclever_counter_type_prop
442     }
443     {#1}
444   } ,
445   countertype .value_required:n = true ,
446   countertype .initial:n =
447   {
448     subsection    = section ,
449     subsubsection = section ,

```

```

450     subparagraph = paragraph ,
451     enumi         = item ,
452     enumii        = item ,
453     enumiii       = item ,
454     enumiv        = item ,
455   } ,
456 }

```

#### 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.

```

457 \seq_new:N \l__zrefclever_counter_resetters_seq
458 \keys_define:nn { zref-clever / label }
459 {
460   counterresetters .code:n =
461   {
462     \clist_map_inline:nn {#1}
463     {
464       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
465       {
466         \seq_put_right:Nn
467         \l__zrefclever_counter_resetters_seq {##1}
468       }
469     }
470   } ,
471   counterresetters .initial:n =
472   {
473     part ,
474     chapter ,
475     section ,
476     subsection ,
477     subsubsection ,
478     paragraph ,
479     subparagraph ,
480   } ,
481   typesort .value_required:n = true ,
482 }

```

#### 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`.

```

483 \prop_new:N \l__zrefclever_counter_resetby_prop
484 \keys_define:nn { zref-clever / label }
485 {
486   counterresetby .code:n =
487   {
488     \keyval_parse:nnn
489     {
490       \msg_warning:nnn { zref-clever }
491       { key-requires-value } { counterresetby }
492     }
493     {
494       \__zrefclever_prop_put_non_empty:Nnn
495       \l__zrefclever_counter_resetby_prop
496     }
497     {#1}
498   } ,
499   counterresetby .value_required:n = true ,
500   counterresetby .initial:n =
501   {

```

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.

```

502     enumii = enumi ,
503     enumiii = enumii ,
504     enumiv = enumiii ,
505   } ,
506 }

```

## 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.

```

507 \tl_new:N \l__zrefclever_ref_property_tl
508 \keys_define:nn { zref-clever / reference }
509 {
510   ref .choice: ,
511   ref / zc@thecnt .code:n =
512   { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
513   ref / page .code:n =
514   { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
515   ref / title .code:n =
516   {
517     \AddToHook { begindocument }

```



```

518     {
519       \@ifpackageloaded { zref-titleref }
520       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
521       {
522         \msg_warning:nn { zref-clever } { missing-zref-titleref }
523         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
524       }
525     }
526   } ,
527   ref .initial:n = zc@thecnt ,
528   ref .value_required:n = true ,
529   page .meta:n = { ref = page } ,
530   page .value_forbidden:n = true ,
531 }
532 \AddToHook { begindocument }
533 {
534   \@ifpackageloaded { zref-titleref }
535   {
536     \keys_define:nn { zref-clever / reference }
537     {
538       ref / title .code:n =
539       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
540     }
541   }
542   {
543     \keys_define:nn { zref-clever / reference }
544     {
545       ref / title .code:n =
546       {
547         \msg_warning:nn { zref-clever } { missing-zref-titleref }
548         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
549       }
550     }
551   }
552 }

```

## typeset option

```

553 \bool_new:N \l__zrefclever_typeset_ref_bool
554 \bool_new:N \l__zrefclever_typeset_name_bool
555 \keys_define:nn { zref-clever / reference }
556 {
557   typeset .choice: ,
558   typeset / both .code:n =
559   {
560     \bool_set_true:N \l__zrefclever_typeset_ref_bool
561     \bool_set_true:N \l__zrefclever_typeset_name_bool
562   } ,
563   typeset / ref .code:n =
564   {
565     \bool_set_true:N \l__zrefclever_typeset_ref_bool
566     \bool_set_false:N \l__zrefclever_typeset_name_bool
567   } ,
568   typeset / name .code:n =

```

```

569     {
570       \bool_set_false:N \l__zrefclever_typeset_ref_bool
571       \bool_set_true:N \l__zrefclever_typeset_name_bool
572     } ,
573     typeset .initial:n = both ,
574     typeset .value_required:n = true ,
575
576     noname .meta:n = { typeset = ref },
577     noname .value_forbidden:n = true ,
578   }

```

#### sort option

```

579 \bool_new:N \l__zrefclever_typeset_sort_bool
580 \keys_define:nn { zref-clever / reference }
581 {
582   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
583   sort .initial:n = true ,
584   sort .default:n = true ,
585   nosort .meta:n = { sort = false },
586   nosort .value_forbidden:n = true ,
587 }

```

#### 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.

```

588 \seq_new:N \l__zrefclever_typesort_seq
589 \keys_define:nn { zref-clever / reference }
590 {
591   typesort .code:n =
592   {
593     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
594     \seq_reverse:N \l__zrefclever_typesort_seq
595   } ,
596   typesort .initial:n =
597   { part , chapter , section , paragraph },
598   typesort .value_required:n = true ,
599   notypesort .code:n =
600   { \seq_clear:N \l__zrefclever_typesort_seq } ,
601   notypesort .value_forbidden:n = true ,
602 }

```

#### comp option

```

603 \bool_new:N \l__zrefclever_typeset_compress_bool
604 \keys_define:nn { zref-clever / reference }
605 {
606   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
607   comp .initial:n = true ,
608   comp .default:n = true ,
609   nocomp .meta:n = { comp = false },
610   nocomp .value_forbidden:n = true ,

```

```
611 }
```

#### range option

```
612 \bool_new:N \l__zrefclever_typeset_range_bool
613 \keys_define:nn { zref-clever / reference }
614 {
615     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
616     range .initial:n = false ,
617     range .default:n = true ,
618 }
```

#### hyperref option

```
619 \bool_new:N \l__zrefclever_use_hyperref_bool
620 \bool_new:N \l__zrefclever_warn_hyperref_bool
621 \keys_define:nn { zref-clever / reference }
622 {
623     hyperref .choice: ,
624     hyperref / auto .code:n =
625     {
626         \bool_set_true:N \l__zrefclever_use_hyperref_bool
627         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
628     } ,
629     hyperref / true .code:n =
630     {
631         \bool_set_true:N \l__zrefclever_use_hyperref_bool
632         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
633     } ,
634     hyperref / false .code:n =
635     {
636         \bool_set_false:N \l__zrefclever_use_hyperref_bool
637         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
638     } ,
639     hyperref .initial:n = auto ,
640     hyperref .default:n = auto
641 }
642 \AddToHook { begindocument }
643 {
644     \@ifpackageloaded { hyperref }
645     {
646         \bool_if:NT \l__zrefclever_use_hyperref_bool
647         { \RequirePackage { zref-hyperref } }
648     }
649     {
650         \bool_if:NT \l__zrefclever_warn_hyperref_bool
651         { \msg_warning:nn { zref-clever } { missing-hyperref } }
652         \bool_set_false:N \l__zrefclever_use_hyperref_bool
653     }
654     \keys_define:nn { zref-clever / reference }
655     {
656         hyperref .code:n =
657         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
658     }
659 }
```

### nameinlink option

```
660 \str_new:N \l__zrefclever_nameinlink_str
661 \keys_define:nn { zref-clever / reference }
662 {
663   nameinlink .choice: ,
664   nameinlink / true .code:n =
665     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
666   nameinlink / false .code:n =
667     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
668   nameinlink / single .code:n =
669     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
670   nameinlink / tsingle .code:n =
671     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
672   nameinlink .initial:n = tsingle ,
673   nameinlink .default:n = true ,
674 }
```

### cap and capfirst options

```
675 \bool_new:N \l__zrefclever_capitalize_bool
676 \bool_new:N \l__zrefclever_capitalize_first_bool
677 \keys_define:nn { zref-clever / reference }
678 {
679   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
680   cap .initial:n = false ,
681   cap .default:n = true ,
682   nocap .meta:n = { cap = false },
683   nocap .value_forbidden:n = true ,
684
685   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
686   capfirst .initial:n = false ,
687   capfirst .default:n = true ,
688
689   C .meta:n =
690     { capfirst = true , noabbrevfirst = true },
691   C .value_forbidden:n = true ,
692 }
```

### abbrev and noabbrevfirst options

```
693 \bool_new:N \l__zrefclever_abbrev_bool
694 \bool_new:N \l__zrefclever_noabbrev_first_bool
695 \keys_define:nn { zref-clever / reference }
696 {
697   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
698   abbrev .initial:n = false ,
699   abbrev .default:n = true ,
700   noabbrev .meta:n = { abbrev = false },
701   noabbrev .value_forbidden:n = true ,
702
703   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
704   noabbrevfirst .initial:n = false ,
705   noabbrevfirst .default:n = true ,
706 }
```

## 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`.

```
707 \tl_new:N \l__zrefclever_ref_language_tl
708 \tl_new:N \l__zrefclever_main_language_tl
709 \tl_new:N \l__zrefclever_current_language_tl
710 \AddToHook { begindocument }
711 {
712   \@ifpackageloaded { babel }
713   {
714     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
715     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
716   }
717   {
718     \@ifpackageloaded { polyglossia }
719     {
720       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
721       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
722     }
723     {
724       \tl_set:Nn \l__zrefclever_current_language_tl { english }
725       \tl_set:Nn \l__zrefclever_main_language_tl { english }
726     }
727   }
```

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.

```

728     \tl_set:Nn \l__zrefclever_ref_language_tl { \l__zrefclever_main_language_tl }
729   }
730   \keys_define:nn { zref-clever / reference }
731   {
732     lang .code:n =
733     {
734       \AddToHook { begindocument }
735       {
736         \str_case:nnF {#1}
737         {
738           { main }
739           {
740             \tl_set:Nn \l__zrefclever_ref_language_tl
741             { \l__zrefclever_main_language_tl }
742             \__zrefclever_provide_dictionary_verbosely:x
743             { \l__zrefclever_ref_language_tl }
744           }
745
746           { current }
747           {
748             \tl_set:Nn \l__zrefclever_ref_language_tl
749             { \l__zrefclever_current_language_tl }
750             \__zrefclever_provide_dictionary_verbosely:x
751             { \l__zrefclever_ref_language_tl }
752           }
753         }
754       }
755       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
756       \__zrefclever_provide_dictionary_verbosely:x
757       { \l__zrefclever_ref_language_tl }
758     }
759   } ,
760   lang .value_required:n = true ,
761 }
762
763 \AddToHook { begindocument / before }
764 {
765   \AddToHook { begindocument }
766   {

```

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.

```

767   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

768   \keys_define:nn { zref-clever / reference }
769   {
770     lang .code:n =
771     {
772       \str_case:nnF {#1}
773       {
774         { main }
775         {
776           \tl_set:Nn \l__zrefclever_ref_language_tl

```

```

777         { \l__zrefclever_main_language_tl }
778         \__zrefclever_provide_dictionary_verbose:x
779         { \l__zrefclever_ref_language_tl }
780     }
781
782     { current }
783     {
784         \tl_set:Nn \l__zrefclever_ref_language_tl
785         { \l__zrefclever_current_language_tl }
786         \__zrefclever_provide_dictionary_verbose:x
787         { \l__zrefclever_ref_language_tl }
788     }
789 }
790 {
791     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
792     \__zrefclever_provide_dictionary_verbose:x
793     { \l__zrefclever_ref_language_tl }
794 }
795 },
796 lang .value_required:n = true ,
797 }
798 }
799 }

```

#### font option

```

800 \tl_new:N \l__zrefclever_ref_typeset_font_tl
801 \keys_define:nn { zref-clever / reference }
802 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

#### note option

```

803 \tl_new:N \l__zrefclever_zcref_note_tl
804 \keys_define:nn { zref-clever / reference }
805 {
806     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
807     note .value_required:n = true ,
808 }

```

#### check option

Integration with zref-check.

```

809 \bool_new:N \l__zrefclever_zrefcheck_available_bool
810 \bool_new:N \l__zrefclever_zcref_with_check_bool
811 \keys_define:nn { zref-clever / reference }
812 {
813     check .code:n =
814     { \msg_warning:nn { zref-clever } { check-document-only } } ,
815 }
816 \AddToHook { begindocument }
817 {
818     \@ifpackageloaded { zref-check }
819     {
820         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
821         \keys_define:nn { zref-clever / reference }

```

```

822     {
823       check .code:n =
824       {
825         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
826         \keys_set:nn { zref-check / zcheck } {#1}
827       }
828     }
829   }
830   {
831     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
832     \keys_define:nn { zref-clever / reference }
833     {
834       check .code:n =
835       { \msg_warning:nn { zref-clever } { missing-zref-check } }
836     }
837   }
838 }

```

## 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 `\zsetup` 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 5.2.

```

839 \prop_new:N \l__zrefclever_ref_options_prop
840 \seq_map_inline:Nn
841   \c__zrefclever_ref_options_not_type_specific_seq
842   {
843     \keys_define:nn { zref-clever / reference }
844     {
845       #1 .default:V = \c_novalue_tl ,
846       #1 .code:n =
847       {
848         \tl_if_novalue:nTF {##1}
849         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
850         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
851       } ,
852     }
853   }

```

## Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: **label** and **reference**. Currently, the only use of this selection is the ability to exclude label related options from 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.

```

854 \keys_define:nn { }
855 {
856     zref-clever / zcsetup .inherit:n = zref-clever / label ,
857     zref-clever / zcsetup .inherit:n = zref-clever / reference ,
858 }

```

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

```

859 \ProcessKeysOptions { zref-clever / zcsetup }

```

## 5 User interface

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

860 \NewDocumentCommand \zcsetup { m }
861 { \keys_set:nn { zref-clever / zcsetup } {#1} }

```

(End definition for `\zcsetup`.)

### 5.2 `\zcRefTypeSetup`

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

```

\zcRefTypeSetup      \zcRefTypeSetup {<type>} {<options>}
862 \NewDocumentCommand \zcRefTypeSetup { m m }
863 {
864     \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
865     { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
866     \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
867     \keys_set:nn { zref-clever / typesetup } {#2}
868 }

```

(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 4.5), 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>.

```

869 \seq_map_inline:Nn
870   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
871   {
872     \keys_define:nn { zref-clever / typesetup }
873     {
874       #1 .code:n =
875       {
876         \msg_warning:nnn { zref-clever }
877           { option-not-type-specific } {#1}
878       } ,
879     }
880   }

881 \seq_map_inline:Nn
882   \c__zrefclever_ref_options_type_specific_seq
883   {
884     \keys_define:nn { zref-clever / typesetup }
885     {
886       #1 .default:V = \c_novalue_tl ,
887       #1 .code:n =
888       {
889         \tl_if_novalue:nTF {##1}
890         {
891           \prop_remove:cn
892             {
893               l__zrefclever_type_
894               \l__zrefclever_setup_type_tl _options_prop
895             }
896             {#1}
897         }
898         {
899           \prop_put:cnn
900             {
901               l__zrefclever_type_
902               \l__zrefclever_setup_type_tl _options_prop
903             }
904             {#1} {##1}
905         }
906       } ,
907     }
908   }

```

### 5.3 \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  $\langle options \rangle$  argument of `\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 is preamble only.

```

\zcDeclareTranslations      \zcDeclareTranslations {<language>} {<options>}
909 \NewDocumentCommand \zcDeclareTranslations { m m }
910 {
911   \group_begin:
912   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
913   \l__zrefclever_dict_language_tl
914   {
915     \tl_clear:N \l__zrefclever_setup_type_tl
916     \keys_set:nn { zref-clever / translations } {#2}
917   }
918   { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
919   \group_end:
920 }
921 \@onlypreamble \zcDeclareTranslations

(End definition for \zcDeclareTranslations.)

922 \keys_define:nn { zref-clever / translations }
923 {
924   type .code:n =
925   {
926     \tl_if_empty:NTF {#1}
927     { \tl_clear:N \l__zrefclever_setup_type_tl }
928     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
929   } ,
930 }

931 % {<language>}{<type>}{<key>}{<translation>}
932 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
933 {
934   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
935   { type- #2 - #3 } {#4}
936 }
937 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
938
939 % {<language>}{<key>}{<translation>}
940 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
941 {
942   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
943   { default- #2 } {#3}
944 }
945 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }
946 \seq_map_inline:Nn
947 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
948 {
949   \keys_define:nn { zref-clever / translations }
950   {
951     #1 .value_required:n = true ,
952     #1 .code:n =
953     {
954       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
955       {

```

```

956         \__zrefclever_declare_default_transl:Vnn
957         \l__zrefclever_dict_language_tl
958         {#1} {##1}
959     }
960     {
961         \msg_warning:nnn { zref-clever }
962         { option-not-type-specific } {#1}
963     }
964 } ,
965 }
966 }
967 \seq_map_inline:Nn
968 \c__zrefclever_ref_options_possibly_type_specific_seq
969 {
970     \keys_define:nn { zref-clever / translations }
971     {
972         #1 .value_required:n = true ,
973         #1 .code:n =
974         {
975             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
976             {
977                 \__zrefclever_declare_default_transl:Vnn
978                 \l__zrefclever_dict_language_tl
979                 {#1} {##1}
980             }
981             {
982                 \__zrefclever_declare_type_transl:VVnn
983                 \l__zrefclever_dict_language_tl
984                 \l__zrefclever_setup_type_tl
985                 {#1} {##1}
986             }
987         } ,
988     }
989 }
990 \seq_map_inline:Nn
991 \c__zrefclever_ref_options_necessarily_type_specific_seq
992 {
993     \keys_define:nn { zref-clever / translations }
994     {
995         #1 .value_required:n = true ,
996         #1 .code:n =
997         {
998             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
999             {
1000                 \msg_warning:nnn { zref-clever }
1001                 { option-only-type-specific } {#1}
1002             }
1003             {
1004                 \__zrefclever_declare_type_transl:VVnn
1005                 \l__zrefclever_dict_language_tl
1006                 \l__zrefclever_setup_type_tl
1007                 {#1} {##1}
1008             }

```

```

1009         } ,
1010     }
1011 }

```

## 5.4 \zcref

```

\zcref          \zcref<*>[<options>]{<labels>}
1012 \NewDocumentCommand \zcref { s O { } m }
1013 { \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>}
1014 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1015 {
1016     \group_begin:

```

Set options.

```

1017     \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1018     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1019     \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.

```

1020     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with `zref-check`.

```

1021     \bool_lazy_and:nnT
1022     { \l__zrefclever_zrefcheck_available_bool }
1023     { \l__zrefclever_zcref_with_check_bool }
1024     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1025     \bool_lazy_or:nnT
1026     { \l__zrefclever_typeset_sort_bool }
1027     { \l__zrefclever_typeset_range_bool }
1028     { \__zrefclever_sort_labels: }

```

Typeset the references.

```

1029     \__zrefclever_typeset_refs:

```

Typeset note.

```

1030     \l__zrefclever_notesep_tl
1031     \l__zrefclever_zcref_note_tl

```

Integration with zref-check.

```

1032     \bool_lazy_and:nnT
1033     { \l__zrefclever_zrefcheck_available_bool }
1034     { \l__zrefclever_zcref_with_check_bool }
1035     {
1036       \zrefcheck_zcref_end_label_maybe:
1037       \zrefcheck_zcref_run_checks_on_labels:n
1038       { \l__zrefclever_zcref_labels_seq }
1039     }
1040   \group_end:
1041 }

```

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

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1042 \seq_new:N \l__zrefclever_zcref_labels_seq
1043 \bool_new:N \l__zrefclever_link_star_bool

```

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

## 5.5 \zcpageref

```

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

1044 \NewDocumentCommand \zcpageref { s O { } m }
1045 {
1046   \IfBooleanTF {#1}
1047   { \zcref*[#2, ref = page] {#3} }
1048   { \zcref [ #2, ref = page] {#3} }
1049 }

```

(End definition for \zcpageref.)

## 6 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 1050 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl 1051 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_label_type_a_tl 1052 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl 1053 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclcnt_a_tl 1054 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclcnt_b_tl 1055 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
\l__zrefclever_label_enclval_a_tl 1056 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl 1057 \tl_new:N \l__zrefclever_label_enclval_b_tl

```

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

```

1058 \int_new:N \l__zrefclever_sort_prior_a_int
1059 \int_new:N \l__zrefclever_sort_prior_b_int

```

Auxiliary variable for \l\_\_zrefclever\_sort\_default:nn, signals if the sorting between two labels has been decided or not.

```

1060 \bool_new:N \l__zrefclever_sort_decided_bool

```

(End definition for \l\_\_zrefclever\_sort\_decided\_bool.)

Variant not provided by the kernel.

```
1061 \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}}
```

```
1062 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1063 {
1064   \tl_set:Nx \l__zrefclever_label_type_a_tl
1065   { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1066   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1067   {
1068     \seq_if_in:NVF
1069     \l__zrefclever_label_types_seq
1070     \l__zrefclever_label_type_a_tl
1071     {
1072       \seq_put_right:NV \l__zrefclever_label_types_seq
1073       \l__zrefclever_label_type_a_tl
1074     }
1075   }
1076 }
```

(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 \zceref. This order is required as a “last resort” sort criterion between the reference types, for use in \\_zrefclever\_sort\_default:nn.

```
1077 \seq_new:N \l__zrefclever_label_types_seq
```

(End definition for \l\_\_zrefclever\_label\_types\_seq.)

\\_zrefclever\_sort\_labels: The main sorting function. It does not receive arguments, but it is expected to be run inside \\_zrefclever\_zceref:nnnn where a number of environment variables are to be set appropriately. In particular, \l\_\_zrefclever\_zceref\_labels\_seq should contain the labels received as argument to \zceref, and the function performs its task by sorting this variable.

```
1078 \cs_new_protected:Npn \_zrefclever_sort_labels:
1079 {
```

Store label types sequence.

```
1080   \seq_clear:N \l__zrefclever_label_types_seq
1081   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1082   {
1083     \seq_map_function:NN \l__zrefclever_zceref_labels_seq
1084     \_zrefclever_label_type_put_new_right:n
1085   }
```

Sort.

```

1086 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1087 {
1088   \zref@ifrefundefined {##1}
1089   {
1090     \zref@ifrefundefined {##2}
1091     {
1092       % Neither label is defined.
1093       \sort_return_same:
1094     }
1095     {
1096       % The second label is defined, but the first isn't, leave the
1097       % undefined first (to be more visible).
1098       \sort_return_same:
1099     }
1100   }
1101   {
1102     \zref@ifrefundefined {##2}
1103     {
1104       % The first label is defined, but the second isn't, bring the
1105       % second forward.
1106       \sort_return_swapped:
1107     }
1108     {
1109       % The interesting case: both labels are defined. The
1110       % reference to the "default" property/counter or to the page
1111       % are quite different from our perspective, they rely on
1112       % different fields and even use different information for
1113       % sorting, so we branch them here to specialized functions.
1114       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1115       { \__zrefclever_sort_page:nn {##1} {##2} }
1116       { \__zrefclever_sort_default:nn {##1} {##2} }
1117     }
1118   }
1119 }
1120 }

```

(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)}

1121 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1122 {
1123   \tl_set:Nx \l__zrefclever_label_type_a_tl
1124   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1125   \tl_set:Nx \l__zrefclever_label_type_b_tl
1126   { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1127

```



```

1128 \bool_if:nTF
1129 {
1130   % The second label has a type, but the first doesn't, leave the
1131   % undefined first (to be more visible).
1132   \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1133   ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1134 }
1135 { \sort_return_same: }
1136 {
1137   \bool_if:nTF
1138   {
1139     % The first label has a type, but the second doesn't, bring the
1140     % second forward.
1141     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1142     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1143   }
1144   { \sort_return_swapped: }
1145   {
1146     \bool_if:nTF
1147     {
1148       % The interesting case: both labels have a type...
1149       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1150       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1151     }
1152     {
1153       % Here we send this to a couple of auxiliary functions for no
1154       % other reason than to keep this long function a little less
1155       % unreadable.
1156       \tl_if_eq:NNTF
1157       \l__zrefclever_label_type_a_tl
1158       \l__zrefclever_label_type_b_tl
1159       {
1160         % ...and it's the same type.
1161         \__zrefclever_sort_default_same_type:nn {#1} {#2}
1162       }
1163       {
1164         % ...and they are different types.
1165         \__zrefclever_sort_default_different_types:nn {#1} {#2}
1166       }
1167     }
1168   }
1169   {
1170     % Neither of the labels has a type. We can't do much of
1171     % meaningful here, but if it's the same counter, compare it.
1172     \exp_args:Nxx \tl_if_eq:nnTF
1173     { \zref@extractdefault {#1} { counter } { } }
1174     { \zref@extractdefault {#2} { counter } { } }
1175     {
1176       \int_compare:nNnTF
1177       { \zref@extractdefault {#1} { zc@cntval } {-1} }
1178       >
1179       { \zref@extractdefault {#2} { zc@cntval } {-1} }
1180       { \sort_return_swapped: }
1181       { \sort_return_same: }
1182     }
1183   }

```

```

1182         { \sort_return_same: }
1183     }
1184 }
1185 }
1186 }

```

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

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

```

1187 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1188 {
1189   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1190     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1191   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1192     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1193   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1194     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1195   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1196     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1197   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1198     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1199   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1200     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1201   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1202     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1203   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1204     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1205
1206   \bool_set_false:N \l__zrefclever_sort_decided_bool
1207   % CHECK should I replace the tmp variables here?
1208   \tl_clear:N \l_tmpa_tl
1209   \tl_clear:N \l_tmpb_tl
1210   \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1211   {
1212     \tl_set:Nx \l_tmpa_tl
1213       { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1214     \tl_set:Nx \l_tmpb_tl
1215       { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1216
1217     \bool_if:nTF
1218     {
1219       % Both are empty, meaning: neither labels have any (further)
1220       % ‘‘enclosing counters’’ (left).
1221       \tl_if_empty_p:V \l_tmpa_tl &&
1222       \tl_if_empty_p:V \l_tmpb_tl
1223     }
1224     {
1225       \exp_args:Nxx \tl_if_eq:nnTF
1226       { \zref@extractdefault {#1} { counter } { } }
1227       { \zref@extractdefault {#2} { counter } { } }
1228       {
1229         \bool_set_true:N \l__zrefclever_sort_decided_bool
1230         \int_compare:nNnTF
1231         { \zref@extractdefault {#1} { zc@cntval } {-1} }

```

```

1232         >
1233         { \zref@extractdefault {#2} { zc@cntval } {-1} }
1234         { \sort_return_swapped: }
1235         { \sort_return_same: }
1236     }
1237     {
1238         \msg_warning:nnnn { zref-clever }
1239         { counters-not-nested } {#1} {#2}
1240         \bool_set_true:N \l__zrefclever_sort_decided_bool
1241         \sort_return_same:
1242     }
1243 }
1244 {
1245     \bool_if:nTF
1246     {
1247         % 'a' is empty (and 'b' is not), meaning: 'b' is (possibly)
1248         % nested in 'a'.
1249         \tl_if_empty_p:V \l_tmpa_tl
1250     }
1251     {
1252         \tl_set:Nx \l_tmpa_tl
1253         { {\zref@extractdefault {#1} { counter } { }} }
1254         \exp_args:NNx \tl_if_in:NnTF
1255         \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1256         {
1257             \bool_set_true:N \l__zrefclever_sort_decided_bool
1258             \sort_return_same:
1259         }
1260         {
1261             \msg_warning:nnnn { zref-clever }
1262             { counters-not-nested } {#1} {#2}
1263             \bool_set_true:N \l__zrefclever_sort_decided_bool
1264             \sort_return_same:
1265         }
1266     }
1267 }
1268 \bool_if:nTF
1269 {
1270     % 'b' is empty (and 'a' is not), meaning: 'a' is
1271     % (possibly) nested in 'b'.
1272     \tl_if_empty_p:V \l_tmpb_tl
1273 }
1274 {
1275     \tl_set:Nx \l_tmpb_tl
1276     { {\zref@extractdefault {#2} { counter } { }} }
1277     \exp_args:NNx \tl_if_in:NnTF
1278     \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1279     {
1280         \bool_set_true:N \l__zrefclever_sort_decided_bool
1281         \sort_return_swapped:
1282     }
1283     {
1284         \msg_warning:nnnn { zref-clever }
1285         { counters-not-nested } {#1} {#2}

```

```

1286         \bool_set_true:N \l__zrefclever_sort_decided_bool
1287         \sort_return_same:
1288     }
1289 }
1290 {
1291     % Neither is empty, meaning: we can (possibly) compare the
1292     % values of the current enclosing counter in the loop, if
1293     % they are equal, we are still in the loop, if they are
1294     % not, a sorting decision can be made directly.
1295     \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
1296     {
1297         \int_compare:nNnTF
1298         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1299         =
1300         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1301         {
1302             \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1303             { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1304             \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1305             { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1306             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1307             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1308             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1309             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1310         }
1311         {
1312             \bool_set_true:N \l__zrefclever_sort_decided_bool
1313             \int_compare:nNnTF
1314             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1315             >
1316             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1317             { \sort_return_swapped: }
1318             { \sort_return_same: }
1319         }
1320     }
1321     {
1322         \msg_warning:nnnn { zref-clever }
1323         { counters-not-nested } {#1} {#2}
1324         \bool_set_true:N \l__zrefclever_sort_decided_bool
1325         \sort_return_same:
1326     }
1327 }
1328 }
1329 }
1330 }
1331 }

```

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

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

```

1332 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1333 {
1334     \int_zero:N \l__zrefclever_sort_prior_a_int
1335     \int_zero:N \l__zrefclever_sort_prior_b_int

```

```

1336 % \cs{l__zrefclever_typesort_seq} was stored in reverse sequence, and
1337 % we compute the sort priorities in the negative range, so that we can
1338 % implicitly rely on '0' being the "last value".
1339 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1340 {
1341   \tl_if_eq:nnTF {##2} {{othertypes}}
1342   {
1343     \int_compare:nNt { \l__zrefclever_sort_prior_a_int } = { 0 }
1344     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1345     \int_compare:nNt { \l__zrefclever_sort_prior_b_int } = { 0 }
1346     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1347   }
1348   {
1349     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1350     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1351     {
1352       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1353       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1354     }
1355   }
1356 }
1357 \bool_if:nTF
1358 {
1359   \int_compare_p:nNn
1360   { \l__zrefclever_sort_prior_a_int } <
1361   { \l__zrefclever_sort_prior_b_int }
1362 }
1363 { \sort_return_same: }
1364 {
1365   \bool_if:nTF
1366   {
1367     \int_compare_p:nNn
1368     { \l__zrefclever_sort_prior_a_int } >
1369     { \l__zrefclever_sort_prior_b_int }
1370   }
1371   { \sort_return_swapped: }
1372   {
1373     % Sort priorities are equal for different types: the type that
1374     % occurs first in 'labels', as given by the user, is kept (or
1375     % brought) forward.
1376     \seq_map_inline:Nn \l__zrefclever_label_types_seq
1377     {
1378       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1379       { \seq_map_break:n { \sort_return_same: } }
1380       {
1381         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1382         { \seq_map_break:n { \sort_return_swapped: } }
1383       }
1384     }
1385   }
1386 }
1387 }

```

(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}

1388 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1389 {
1390   \int_compare:nNnTF
1391     { \zref@extractdefault {#1} { abspage } {-1} }
1392     >
1393     { \zref@extractdefault {#2} { abspage } {-1} }
1394     { \sort_return_swapped: }
1395     { \sort_return_same:   }
1396 }

```

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

## 7 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.

```

1397 \bool_new:N \l__zrefclever_typeset_last_bool
1398 \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` 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.

```

1399 \seq_new:N \l__zrefclever_typeset_labels_seq
1400 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1401 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1402 \tl_new:N \l__zrefclever_type_first_label_tl
1403 \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.

```

1404 \int_new:N \l__zrefclever_label_count_int
1405 \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.

```

1406 \int_new:N \l__zrefclever_range_count_int
1407 \int_new:N \l__zrefclever_range_same_count_int
1408 \tl_new:N \l__zrefclever_range_beg_label_tl
1409 \bool_new:N \l__zrefclever_next_maybe_range_bool
1410 \bool_new:N \l__zrefclever_next_is_same_bool
1411 \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.

```

1412 \tl_new:N \l__zrefclever_namefont_tl
1413 \tl_new:N \l__zrefclever_reffont_out_tl
1414 \tl_new:N \l__zrefclever_reffont_in_tl
1415
1416 \tl_new:N \l__zrefclever_namesep_tl
1417 \tl_new:N \l__zrefclever_rangesep_tl
1418 \tl_new:N \l__zrefclever_pairsep_tl
1419 \tl_new:N \l__zrefclever_listsep_tl
1420 \tl_new:N \l__zrefclever_lastsep_tl
1421 \tl_new:N \l__zrefclever_tpairsep_tl
1422 \tl_new:N \l__zrefclever_tlistsep_tl
1423 \tl_new:N \l__zrefclever_tlastsep_tl
1424 \tl_new:N \l__zrefclever_notesep_tl
1425 \tl_new:N \l__zrefclever_refpre_out_tl
1426 \tl_new:N \l__zrefclever_refpos_out_tl
1427 \tl_new:N \l__zrefclever_refpre_in_tl
1428 \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:`.

```

1429 \tl_new:N \l__zrefclever_type_name_tl
1430 \bool_new:N \l__zrefclever_name_in_link_bool
1431 \tl_new:N \l__zrefclever_name_format_tl
1432 \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 `\zcref`.

```

1433 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1434 {
1435   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zcref_labels_seq
1436   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1437   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1438   \tl_clear:N \l__zrefclever_type_first_label_tl
1439   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1440   \tl_clear:N \l__zrefclever_range_beg_label_tl
1441   \int_zero:N \l__zrefclever_label_count_int
1442   \int_zero:N \l__zrefclever_type_count_int
1443   \int_zero:N \l__zrefclever_range_count_int
1444   \int_zero:N \l__zrefclever_range_same_count_int
1445
1446   % Get not-type-specific separators and refpre/pos options.
1447   \__zrefclever_get_option_with_transl:nN {tpairsep} \l__zrefclever_tpairsep_tl
1448   \__zrefclever_get_option_with_transl:nN {tlistsep} \l__zrefclever_tlistsep_tl
1449   \__zrefclever_get_option_with_transl:nN {tlastsep} \l__zrefclever_tlastsep_tl
1450   \__zrefclever_get_option_with_transl:nN {notesep} \l__zrefclever_notesep_tl
1451
1452   % Set the font option for this zcref call.
1453   \l__zrefclever_ref_typeset_font_tl
1454
1455   % Loop over the label list in sequence.
1456   \bool_set_false:N \l__zrefclever_typeset_last_bool
1457   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1458   {
1459     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl
1460     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1461     {
1462       \tl_clear:N \l__zrefclever_label_b_tl
1463       \bool_set_true:N \l__zrefclever_typeset_last_bool
1464     }
1465     { \seq_get_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_b_tl }
1466
1467     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1468     {
1469       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1470       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1471     }
1472     {

```



```

1473 \tl_set:Nx \l__zrefclever_label_type_a_tl
1474 {
1475   \zref@extractdefault
1476   { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1477 }
1478 \tl_set:Nx \l__zrefclever_label_type_b_tl
1479 {
1480   \zref@extractdefault
1481   { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1482 }
1483 }
1484
1485 % First, we establish whether the ‘current label’ (i.e. ‘a’) is the
1486 % last one of its type. This can happen because the ‘next label’
1487 % (i.e. ‘b’) is of a different type (or different definition status),
1488 % or because we are at the end of the list.
1489 \bool_if:NTF \l__zrefclever_typeset_last_bool
1490 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1491 {
1492   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1493   {
1494     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1495     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1496     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1497   }
1498   {
1499     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1500     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1501     {
1502       % Neither is undefined, we must check the types.
1503       \bool_if:nTF
1504       % Both empty: same ‘type’.
1505       {
1506         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1507         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1508       }
1509       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1510       {
1511         \bool_if:nTF
1512         % Neither empty: compare types.
1513         {
1514           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1515           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1516         }
1517         {
1518           \tl_if_eq:NNTF
1519           \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1520           { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1521           { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1522         }
1523         % One empty, the other not: different ‘types’.
1524         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1525       }
1526     }
1527   }

```

```

1527     }
1528   }
1529
1530   % Handle warnings in case of reference or type undefined.
1531   \zref@refused { \l__zrefclever_label_a_tl }
1532   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1533   {}
1534   {
1535     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1536     {
1537       \msg_warning:nmx { zref-clever } { missing-type }
1538       { \l__zrefclever_label_a_tl }
1539     }
1540   }
1541
1542   % Get type-specific separators, refpre/pos and font options, once per
1543   % type.
1544   \int_compare:nNt { \l__zrefclever_label_count_int } = { 0 }
1545   {
1546     \__zrefclever_get_option_plain:nN {namefont}      \l__zrefclever_namefont_tl
1547     \__zrefclever_get_option_plain:nN {reffont}      \l__zrefclever_reffont_out_tl
1548     \__zrefclever_get_option_plain:nN {reffont-in}   \l__zrefclever_reffont_in_tl
1549     \__zrefclever_get_option_with_transl:nN {namesep} \l__zrefclever_namesep_tl
1550     \__zrefclever_get_option_with_transl:nN {rangesep} \l__zrefclever_rangesep_tl
1551     \__zrefclever_get_option_with_transl:nN {pairsep} \l__zrefclever_pairsep_tl
1552     \__zrefclever_get_option_with_transl:nN {listsep} \l__zrefclever_listsep_tl
1553     \__zrefclever_get_option_with_transl:nN {lastsep} \l__zrefclever_lastsep_tl
1554     \__zrefclever_get_option_with_transl:nN {refpre}  \l__zrefclever_refpre_out_tl
1555     \__zrefclever_get_option_with_transl:nN {refpos}  \l__zrefclever_refpos_out_tl
1556     \__zrefclever_get_option_with_transl:nN {refpre-in} \l__zrefclever_refpre_in_tl
1557     \__zrefclever_get_option_with_transl:nN {refpos-in} \l__zrefclever_refpos_in_tl
1558   }
1559
1560   % Here we send this to a couple of auxiliary functions for no other
1561   % reason than to keep this long function a little less unreadable.
1562   \bool_if:NTF \l__zrefclever_last_of_type_bool
1563   {
1564     % There exists no next label of the same type as the current.
1565     \__zrefclever_typeset_refs_aux_last_of_type:
1566   }
1567   {
1568     % There exists a next label of the same type as the current.
1569     \__zrefclever_typeset_refs_aux_not_last_of_type:
1570   }
1571 }
1572 }

```

(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.

```

1573 \cs_new_protected:Npn \__zrefclever_typeset_refs_aux_last_of_type:
1574 {
1575   % Process the current label to the current queue.
1576   \int_case:nnF { \l__zrefclever_label_count_int }

```

```

1577 {
1578   % It is the last label of its type, but also the first one, and that's
1579   % what matters here: just store it.
1580   { 0 }
1581   {
1582     \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1583     \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1584   }
1585
1586   % The last is the second: we have a pair (if not repeated).
1587   { 1 }
1588   {
1589     \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1590     {
1591       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1592       {
1593         \exp_not:V \l__zrefclever_pairsep_tl
1594         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1595       }
1596     }
1597   }
1598 }
1599 % If neither the first, nor the second: we have the last label
1600 % on the current type list (if not repeated).
1601 {
1602   \int_case:nnF { \l__zrefclever_range_count_int }
1603   {
1604     % There was no range going on.
1605     {0}
1606     {
1607       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1608       {
1609         \exp_not:V \l__zrefclever_lastsep_tl
1610         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1611       }
1612     }
1613     % Last in the range is also the second in it.
1614     {1}
1615     {
1616       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1617       {
1618         % We know 'range_beg_label' is not empty, since this is the
1619         % second element in the range, but the third or more in the
1620         % type list.
1621         \exp_not:V \l__zrefclever_listsep_tl
1622         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1623         \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1624         {
1625           \exp_not:V \l__zrefclever_lastsep_tl
1626           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1627         }
1628       }
1629     }
1630   }

```

```

1631 % Last in the range is third or more in it.
1632 {
1633   \int_case:nnF
1634   { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1635   {
1636     % Repetition, not a range.
1637     {0}
1638     {
1639       % If 'range_beg_label' is empty, it means it was also the
1640       % first of the type, and hence was already handled.
1641       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1642       {
1643         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1644         {
1645           \exp_not:V \l__zrefclever_lastsep_tl
1646           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1647         }
1648       }
1649     }
1650     % A 'range', but with no skipped value, treat as list.
1651     {1}
1652     {
1653       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1654       {
1655         % Ditto.
1656         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1657         {
1658           \exp_not:V \l__zrefclever_listsep_tl
1659           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1660         }
1661         \exp_not:V \l__zrefclever_lastsep_tl
1662         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1663       }
1664     }
1665   }
1666   {
1667     % An actual range.
1668     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1669     {
1670       % Ditto.
1671       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1672       {
1673         \exp_not:V \l__zrefclever_lastsep_tl
1674         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1675       }
1676       \exp_not:V \l__zrefclever_rangesep_tl
1677       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1678     }
1679   }
1680 }
1681 }
1682
1683 % Handle 'range' option. The idea is simple: if the queue is not empty,
1684 % we replace it with the end of the range (or pair). We can still

```

```

1685 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1686 % be processing the last label of its type at this point.
1687 \bool_if:NT \l__zrefclever_typeset_range_bool
1688 {
1689   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1690   {
1691     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1692     { }
1693     {
1694       \msg_warning:nxx { zref-clever } { single-element-range }
1695       { \l__zrefclever_type_first_label_type_tl }
1696     }
1697   }
1698   {
1699     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1700     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1701     { }
1702     {
1703       \__zrefclever_labels_in_sequence:nn
1704       { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
1705     }
1706     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1707     {
1708       \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1709       { \exp_not:V \l__zrefclever_pairsep_tl }
1710       { \exp_not:V \l__zrefclever_rangeseq_tl }
1711       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1712     }
1713   }
1714 }
1715
1716 % Now that the type is finished, we can add the name and the first ref to
1717 % the queue. Or, if ‘‘typset’’ option is not ‘‘both’’, handle it here
1718 % too.
1719 \__zrefclever_type_name_setup:
1720 \bool_if:nTF
1721 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1722 {
1723   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1724   { \__zrefclever_get_ref_first: }
1725 }
1726 {
1727   \bool_if:nTF
1728   { \l__zrefclever_typeset_ref_bool }
1729   {
1730     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1731     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1732   }
1733   {
1734     \bool_if:nTF
1735     { \l__zrefclever_typeset_name_bool }
1736     {
1737       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1738       {

```

```

1739 \bool_if:NTF \l__zrefclever_name_in_link_bool
1740 {
1741   \exp_not:N \group_begin:
1742   \exp_not:V \l__zrefclever_namefont_tl
1743   % It's two '@s', but escaped for DocStrip.
1744   \exp_not:N \hyper@@link
1745   {
1746     \zref@ifrefcontainsprop
1747     { \l__zrefclever_type_first_label_tl } { urluse }
1748     {
1749       \zref@extractdefault
1750       { \l__zrefclever_type_first_label_tl }
1751       { urluse } {}
1752     }
1753     {
1754       \zref@extractdefault
1755       { \l__zrefclever_type_first_label_tl }
1756       { url } {}
1757     }
1758   }
1759   {
1760     \zref@extractdefault
1761     { \l__zrefclever_type_first_label_tl } { anchor } {}
1762   }
1763   { \exp_not:V \l__zrefclever_type_name_tl }
1764   \exp_not:N \group_end:
1765 }
1766 {
1767   \exp_not:N \group_begin:
1768   \exp_not:V \l__zrefclever_namefont_tl
1769   \exp_not:V \l__zrefclever_type_name_tl
1770   \exp_not:N \group_end:
1771 }
1772 }
1773 }
1774 {
1775   % This case would correspond to "typeset=none" but should not
1776   % happen, given the options are set up to typeset at least one
1777   % of "ref" or "name", but a sensible fallback, equal to the
1778   % behavior of 'both'.
1779   \tl_put_left:Nx
1780     \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1781 }
1782 }
1783 }
1784
1785 % Typeset the previous type, if there is one.
1786 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1787 {
1788   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1789   { \l__zrefclever_tlistsep_tl }
1790   \l__zrefclever_typeset_queue_prev_tl
1791 }
1792

```

```

1793 % Wrap up loop, or prepare for next iteration.
1794 \bool_if:NTF \l__zrefclever_typeset_last_bool
1795 {
1796   % We are finishing, typeset the current queue.
1797   \int_case:nnF { \l__zrefclever_type_count_int }
1798   {
1799     % Single type.
1800     { 0 }
1801     { \l__zrefclever_typeset_queue_curr_tl }
1802     % Pair of types.
1803     { 1 }
1804     {
1805       \l__zrefclever_tpairsep_tl
1806       \l__zrefclever_typeset_queue_curr_tl
1807     }
1808   }
1809   {
1810     % Last in list of types.
1811     \l__zrefclever_tlastsep_tl
1812     \l__zrefclever_typeset_queue_curr_tl
1813   }
1814 }
1815 {
1816   % There are further labels, set variables for next iteration.
1817   \tl_set_eq:NN
1818     \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1819   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1820   \tl_clear:N \l__zrefclever_type_first_label_tl
1821   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1822   \tl_clear:N \l__zrefclever_range_beg_label_tl
1823   \int_zero:N \l__zrefclever_label_count_int
1824   \int_incr:N \l__zrefclever_type_count_int
1825   \int_zero:N \l__zrefclever_range_count_int
1826   \int_zero:N \l__zrefclever_range_same_count_int
1827 }
1828 }

```

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

efclever\_typeset\_refs\_aux\_not\_last\_of\_type: Handles typesetting of when the current label is not the last of its type.

```

1829 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_not_last_of_type:
1830 {
1831   % Signal if next label may form a range with the current one (of
1832   % course, only considered if compression is enabled in the first
1833   % place).
1834   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1835   \bool_set_false:N \l__zrefclever_next_is_same_bool
1836   \bool_lazy_and:nnT
1837     { \l__zrefclever_typeset_compress_bool }
1838     % Currently no-op, but kept as ‘handle’ to inhibit compression of
1839     % individual labels.
1840     { ! \l__zrefclever_range_inhibit_next_bool }
1841   {
1842     \zref@ifrefundefined { \l__zrefclever_label_a_tl }

```

```

1843     { }
1844     {
1845         \_zrefclever_labels_in_sequence:nn
1846         { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1847     }
1848 }
1849
1850 % Process the current label to the current queue.
1851 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1852 {
1853     % Current label is the first of its type (also not the last, but it
1854     % doesn't matter here): just store the label.
1855     \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1856     \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1857
1858     % If the next label may be part of a range, we set 'range_beg_label'
1859     % to 'empty' (we deal with it as the 'first', and must do it
1860     % there, to handle hyperlinking), but also step the range counters.
1861     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1862     {
1863         \tl_clear:N \l__zrefclever_range_beg_label_tl
1864         \int_incr:N \l__zrefclever_range_count_int
1865         \bool_if:NT \l__zrefclever_next_is_same_bool
1866         { \int_incr:N \l__zrefclever_range_same_count_int }
1867     }
1868 }
1869 {
1870     % Current label is neither the first (nor the last) of its
1871     % type.
1872     \bool_if:NnTF \l__zrefclever_next_maybe_range_bool
1873     {
1874         % Starting, or continuing a range.
1875         \int_compare:nNnTF
1876         { \l__zrefclever_range_count_int } = {0}
1877         {
1878             % There was no range going, we are starting one.
1879             \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1880             \int_incr:N \l__zrefclever_range_count_int
1881             \bool_if:NT \l__zrefclever_next_is_same_bool
1882             { \int_incr:N \l__zrefclever_range_same_count_int }
1883         }
1884         {
1885             % Second or more in the range, but not the last.
1886             \int_incr:N \l__zrefclever_range_count_int
1887             \bool_if:NT \l__zrefclever_next_is_same_bool
1888             { \int_incr:N \l__zrefclever_range_same_count_int }
1889         }
1890     }
1891 }
1892 {
1893     % Next element is not in sequence, meaning: there was no range, or
1894     % we are closing one.
1895     \int_case:nnF { \l__zrefclever_range_count_int }
1896     {
1897         % There was no range going on.

```



```

1897 {0}
1898 {
1899   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1900   {
1901     \exp_not:V \l__zrefclever_listsep_tl
1902     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1903   }
1904 }
1905 % Last is second in the range: if 'range_same_count' is also
1906 % '1', it's a repetition (drop it), otherwise, it's a 'pair
1907 % within a list'', treat as list.
1908 {1}
1909 {
1910   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1911   {
1912     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1913     {
1914       \exp_not:V \l__zrefclever_listsep_tl
1915       \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1916     }
1917     \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1918     {
1919       \exp_not:V \l__zrefclever_listsep_tl
1920       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1921     }
1922   }
1923 }
1924 }
1925 {
1926 % Last is third or more in the range: if 'range_count' and
1927 % 'range_same_count' are the same, its a repetition (drop it),
1928 % if they differ by '1', its a list, if they differ by more,
1929 % it is a real range.
1930 \int_case:nnF
1931 { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1932 {
1933   {0}
1934   {
1935     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1936     {
1937       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1938       {
1939         \exp_not:V \l__zrefclever_listsep_tl
1940         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1941       }
1942     }
1943   }
1944   {1}
1945   {
1946     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1947     {
1948       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1949       {
1950         \exp_not:V \l__zrefclever_listsep_tl

```

```

1951         \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1952     }
1953     \exp_not:V \l__zrefclever_listsep_tl
1954     \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1955 }
1956 }
1957 }
1958 {
1959     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1960     {
1961         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1962         {
1963             \exp_not:V \l__zrefclever_listsep_tl
1964             \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1965         }
1966         \exp_not:V \l__zrefclever_rangesep_tl
1967         \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1968     }
1969 }
1970 }
1971 % Reset counters.
1972 \int_zero:N \l__zrefclever_range_count_int
1973 \int_zero:N \l__zrefclever_range_same_count_int
1974 }
1975 }
1976 % Step label counter for next iteration.
1977 \int_incr:N \l__zrefclever_label_count_int
1978 }

```

(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.

```

1979 \cs_new:Npn \_zrefclever_get_ref:n #1
1980 {
1981     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
1982     {
1983         \bool_if:nTF
1984         { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
1985         {
1986             \exp_not:N \group_begin:
1987             \exp_not:V \l__zrefclever_reffont_out_tl
1988             \exp_not:V \l__zrefclever_refpre_out_tl
1989             \exp_not:N \group_begin:
1990             \exp_not:V \l__zrefclever_reffont_in_tl
1991             % It's two '@s', but escaped for DocStrip.
1992             \exp_not:N \hyper@@link
1993             {

```

```

1994         \zref@ifrefcontainsprop {#1} { urluse }
1995         { \zref@extractdefault {#1} { urluse } {} }
1996         { \zref@extractdefault {#1} { url } {} }
1997     }
1998     { \zref@extractdefault {#1} { anchor } {} }
1999     {
2000         \exp_not:V \l__zrefclever_refpre_in_tl
2001         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2002         \exp_not:V \l__zrefclever_refpos_in_tl
2003     }
2004     \exp_not:N \group_end:
2005     \exp_not:V \l__zrefclever_refpos_out_tl
2006     \exp_not:N \group_end:
2007 }
2008 {
2009     \exp_not:N \group_begin:
2010     \exp_not:V \l__zrefclever_reffont_out_tl
2011     \exp_not:V \l__zrefclever_refpre_out_tl
2012     \exp_not:N \group_begin:
2013     \exp_not:V \l__zrefclever_reffont_in_tl
2014     \exp_not:V \l__zrefclever_refpre_in_tl
2015     \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2016     \exp_not:V \l__zrefclever_refpos_in_tl
2017     \exp_not:N \group_end:
2018     \exp_not:V \l__zrefclever_refpos_out_tl
2019     \exp_not:N \group_end:
2020 }
2021 }
2022 { \exp_not:N \zref@default }
2023 }
2024 \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.

```

2025 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2026 {
2027     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2028     { \tl_clear:N \l__zrefclever_type_name_tl }
2029     {
2030         \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl
2031         { \tl_clear:N \l__zrefclever_type_name_tl }
2032         {

```

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

```

2033         \bool_lazy_or:nnTF
2034         { \l__zrefclever_capitalize_bool }
2035         {
2036             \l__zrefclever_capitalize_first_bool &&
2037             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2038         }
2039         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2040         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2041         % If the queue is empty, we have a singular, otherwise, plural.

```

```

2042 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2043 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2044 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2045 \bool_lazy_and:nnTF
2046 { \l__zrefclever_abbrev_bool }
2047 {
2048   ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
2049   ! \l__zrefclever_noabbrev_first_bool
2050 }
2051 {
2052   \tl_set:NV \l__zrefclever_name_format_fallback_tl \l__zrefclever_name_format
2053   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2054 }
2055 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2056
2057 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2058 {
2059   \prop_get:cVNF
2060   { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2061     \l__zrefclever_name_format_tl
2062     \l__zrefclever_type_name_tl
2063     {
2064       \__zrefclever_get_type_transl:xxxNF
2065       { \l__zrefclever_ref_language_tl }
2066       { \l__zrefclever_type_first_label_type_tl }
2067       { \l__zrefclever_name_format_tl }
2068       \l__zrefclever_type_name_tl
2069       {
2070         \tl_clear:N \l__zrefclever_type_name_tl
2071         \msg_warning:nxx { zref-clever } { missing-name }
2072         { \l__zrefclever_type_first_label_type_tl }
2073       }
2074     }
2075   }
2076   {
2077     \prop_get:cVNF
2078     { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2079       \l__zrefclever_name_format_tl
2080       \l__zrefclever_type_name_tl
2081       {
2082         \prop_get:cVNF
2083         { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options
2084           \l__zrefclever_name_format_fallback_tl
2085           \l__zrefclever_type_name_tl
2086           {
2087             \__zrefclever_get_type_transl:xxxNF
2088             { \l__zrefclever_ref_language_tl }
2089             { \l__zrefclever_type_first_label_type_tl }
2090             { \l__zrefclever_name_format_tl }
2091             \l__zrefclever_type_name_tl
2092             {
2093               \__zrefclever_get_type_transl:xxxNF
2094               { \l__zrefclever_ref_language_tl }
2095               { \l__zrefclever_type_first_label_type_tl }

```

```

2096         { \l__zrefclever_name_format_fallback_tl }
2097         \l__zrefclever_type_name_tl
2098         {
2099             \tl_clear:N \l__zrefclever_type_name_tl
2100             \msg_warning:nnx { zref-clever } { missing-name }
2101             { \l__zrefclever_type_first_label_type_tl }
2102         }
2103     }
2104 }
2105 }
2106 }
2107 }
2108 }

```

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

```

2109     \bool_lazy_any:nTF
2110     {
2111         { ! \l__zrefclever_use_hyperref_bool }
2112         { \l__zrefclever_link_star_bool }
2113         { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2114         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2115     }
2116     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2117     {
2118         \bool_lazy_any:nTF
2119         {
2120             { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2121             {
2122                 \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2123                 \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2124             }
2125             {
2126                 \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2127                 \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2128                 \l__zrefclever_typeset_last_bool &&
2129                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2130             }
2131         }
2132         { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2133         { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2134     }
2135 }

```

(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.

```

2136 \cs_new:Npn \__zrefclever_get_ref_first:
2137 {
2138     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2139     { \exp_not:N \zref@default }
2140     {
2141         \bool_if:NTF \l__zrefclever_name_in_link_bool
2142         {

```

```

2143 \zref@ifrefcontainsprop
2144 { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2145 {
2146   % It's two '@s', but escaped for DocStrip.
2147   \exp_not:N \hyper@@link
2148   {
2149     \zref@ifrefcontainsprop
2150     { \l__zrefclever_type_first_label_tl } { urluse }
2151     {
2152       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2153       { urluse } {}
2154     }
2155     {
2156       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2157       { url } {}
2158     }
2159   }
2160   {
2161     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2162     { anchor } {}
2163   }
2164   {
2165     \exp_not:N \group_begin:
2166     \exp_not:V \l__zrefclever_namefont_tl
2167     \exp_not:V \l__zrefclever_type_name_tl
2168     \exp_not:N \group_end:
2169     \exp_not:V \l__zrefclever_namesep_tl
2170     \exp_not:N \group_begin:
2171     \exp_not:V \l__zrefclever_reffont_out_tl
2172     \exp_not:V \l__zrefclever_refpre_out_tl
2173     \exp_not:N \group_begin:
2174     \exp_not:V \l__zrefclever_reffont_in_tl
2175     \exp_not:V \l__zrefclever_refpre_in_tl
2176     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2177     { \l__zrefclever_ref_property_tl } {}
2178     \exp_not:V \l__zrefclever_refpos_in_tl
2179     \exp_not:N \group_end:
2180     % hyperlink makes it's own group, we'd like to close the
2181     % 'refpre-out' group after 'refpos-out', but... we close
2182     % it here, and give the trailing 'refpos-out' its own
2183     % group. This will result that formatting given to
2184     % 'refpre-out' will not reach 'refpos-out', but I see no
2185     % alternative, and this has to be handled specially.
2186     \exp_not:N \group_end:
2187   }
2188   \exp_not:N \group_begin:
2189   % Ditto: special treatment.
2190   \exp_not:V \l__zrefclever_reffont_out_tl
2191   \exp_not:V \l__zrefclever_refpos_out_tl
2192   \exp_not:N \group_end:
2193 }
2194 {
2195   \exp_not:N \group_begin:
2196   \exp_not:V \l__zrefclever_namefont_tl

```

```

2197         \exp_not:V \l__zrefclever_type_name_tl
2198         \exp_not:N \group_end:
2199         \exp_not:V \l__zrefclever_namesep_tl
2200         \exp_not:N \zref@default
2201     }
2202 }
2203 {
2204     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2205     {
2206         \exp_not:N \zref@default
2207         \exp_not:V \l__zrefclever_namesep_tl
2208     }
2209     {
2210         \exp_not:N \group_begin:
2211         \exp_not:V \l__zrefclever_namefont_tl
2212         \exp_not:V \l__zrefclever_type_name_tl
2213         \exp_not:N \group_end:
2214         \exp_not:V \l__zrefclever_namesep_tl
2215     }
2216     \zref@ifrefcontainsprop
2217     { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2218     {
2219         \bool_if:nTF
2220         {
2221             \l__zrefclever_use_hyperref_bool &&
2222             ! \l__zrefclever_link_star_bool
2223         }
2224         {
2225             \exp_not:N \group_begin:
2226             \exp_not:V \l__zrefclever_reffont_out_tl
2227             \exp_not:V \l__zrefclever_refpre_out_tl
2228             \exp_not:N \group_begin:
2229             \exp_not:V \l__zrefclever_reffont_in_tl
2230             % It's two '@s', but escaped for DocStrip.
2231             \exp_not:N \hyper@@link
2232             {
2233                 \zref@ifrefcontainsprop
2234                 { \l__zrefclever_type_first_label_tl } { urluse }
2235                 {
2236                     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2237                     { urluse } {}
2238                 }
2239                 {
2240                     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2241                     { url } {}
2242                 }
2243             }
2244             {
2245                 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2246                 { anchor } {}
2247             }
2248             {
2249                 \exp_not:V \l__zrefclever_refpre_in_tl
2250                 \zref@extractdefault { \l__zrefclever_type_first_label_tl }

```

```

2251         { \l__zrefclever_ref_property_tl } {}
2252         \exp_not:V \l__zrefclever_refpos_in_tl
2253     }
2254     \exp_not:N \group_end:
2255     \exp_not:V \l__zrefclever_refpos_out_tl
2256     \exp_not:N \group_end:
2257 }
2258 {
2259     \exp_not:N \group_begin:
2260     \exp_not:V \l__zrefclever_reffont_out_tl
2261     \exp_not:V \l__zrefclever_refpre_out_tl
2262     \exp_not:N \group_begin:
2263     \exp_not:V \l__zrefclever_reffont_in_tl
2264     \exp_not:V \l__zrefclever_refpre_in_tl
2265     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2266         { \l__zrefclever_ref_property_tl } {}
2267     \exp_not:V \l__zrefclever_refpos_in_tl
2268     \exp_not:N \group_end:
2269     \exp_not:V \l__zrefclever_refpos_out_tl
2270     \exp_not:N \group_end:
2271 }
2272 }
2273 { \exp_not:N \zref@default }
2274 }
2275 }
2276 }

```

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

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

```

2277 % \Arg{option} \Arg{var to store result}
2278 \cs_new_protected:Npn \__zrefclever_get_option_with_transl:nN #1#2
2279 {
2280     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2281     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2282     {
2283         % If not found, try the type specific options.
2284         \bool_lazy_all:nTF
2285         {
2286             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2287             {
2288                 \prop_if_exist_p:c
2289                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2290             }
2291             {
2292                 \prop_if_in_p:cn
2293                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2294             }
2295         }
2296         {
2297             \prop_get:cnN
2298             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2299         }
2300     }

```



```

2301         % If not found, try the type specific translations.
2302         \__zrefclever_get_type_transl:xxnNF
2303         { \l__zrefclever_ref_language_tl }
2304         { \l__zrefclever_label_type_a_tl }
2305         {#1} #2
2306         {
2307             % If not found, try default translations.
2308             \__zrefclever_get_default_transl:xxnNF
2309             { \l__zrefclever_ref_language_tl }
2310             {#1} #2
2311             {
2312                 % If not found, try fallback.
2313                 \__zrefclever_get_fallback_transl:nNF {#1} #2
2314                 { \tl_clear:N #2 }
2315             }
2316         }
2317     }
2318 }
2319 }

```

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

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

```

2320 \cs_new_protected:Npn \__zrefclever_get_option_plain:nN #1#2
2321 {
2322     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2323     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2324     {
2325         % If not found, try the type specific options.
2326         \bool_lazy_and:nnTF
2327         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2328         {
2329             \prop_if_exist_p:c
2330             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2331         }
2332         {
2333             \prop_get:cnNF
2334             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2335             { \tl_clear:N #2 }
2336         }
2337         { \tl_clear:N #2 }
2338     }
2339 }

```

(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”.

```

2340 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2341 {
2342     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2343     {
2344         \exp_args:Nxx \tl_if_eq:nnT

```

```

2345 { \zref@extractdefault {#1} { zc@pgfmt } { } }
2346 { \zref@extractdefault {#2} { zc@pgfmt } { } }
2347 {
2348   \int_compare:nNnTF
2349     { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2350     =
2351     { \zref@extractdefault {#2} { zc@pgval } {-1} }
2352     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2353     {
2354       \int_compare:nNnT
2355         { \zref@extractdefault {#1} { zc@pgval } {-1} }
2356         =
2357         { \zref@extractdefault {#2} { zc@pgval } {-1} }
2358         {
2359           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2360           \bool_set_true:N \l__zrefclever_next_is_same_bool
2361         }
2362       }
2363     }
2364   }
2365 {
2366   \exp_args:Nxx \tl_if_eq:nnT
2367   { \zref@extractdefault {#1} { counter } { } }
2368   { \zref@extractdefault {#2} { counter } { } }
2369   {
2370     \exp_args:Nxx \tl_if_eq:nnT
2371     { \zref@extractdefault {#1} { zc@enclval } { } }
2372     { \zref@extractdefault {#2} { zc@enclval } { } }
2373     {
2374       \int_compare:nNnTF
2375         { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
2376         =
2377         { \zref@extractdefault {#2} { zc@cntval } {-1} }
2378         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2379         {
2380           \int_compare:nNnT
2381             { \zref@extractdefault {#1} { zc@cntval } {-1} }
2382             =
2383             { \zref@extractdefault {#2} { zc@cntval } {-1} }
2384             {
2385               \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2386               \bool_set_true:N \l__zrefclever_next_is_same_bool
2387             }
2388           }
2389         }
2390       }
2391     }
2392   }

```

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

## 8 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.

### 8.1 `\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).

### 8.2 `\newtheorem`

### 8.3 `enumitem` package

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

```
2393 </package>
```

## 9 Dictionaries

### 9.1 English

```
2394 <package>\zcDeclareLanguage { english }
2395 <package>\zcDeclareLanguageAlias { american } { english }
2396 <package>\zcDeclareLanguageAlias { australian } { english }
2397 <package>\zcDeclareLanguageAlias { british } { english }
2398 <package>\zcDeclareLanguageAlias { canadian } { english }
2399 <package>\zcDeclareLanguageAlias { newzealand } { english }
2400 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2401 <package>\zcDeclareLanguageAlias { USenglish } { english }

2402 <*dict-english>

2403 namesep = {\nobreakspace} ,
2404 pairsep  = {\~and\nobreakspace} ,
2405 listsep  = {\,~} ,
2406 lastsep  = {\~and\nobreakspace} ,
2407 tpairsep = {\~and\nobreakspace} ,
2408 tlistsep = {\,~} ,
2409 tlastsep = {\,~and\nobreakspace} ,
```

```

2410 notesep    = {-} ,
2411 rangesep    = {-to\nobreakspace} ,
2412
2413 type = part ,
2414     Name-sg = Part ,
2415     name-sg = part ,
2416     Name-pl = Parts ,
2417     name-pl = parts ,
2418
2419 type = chapter ,
2420     Name-sg = Chapter ,
2421     name-sg = chapter ,
2422     Name-pl = Chapters ,
2423     name-pl = chapters ,
2424
2425 type = section ,
2426     Name-sg = Section ,
2427     name-sg = section ,
2428     Name-pl = Sections ,
2429     name-pl = sections ,
2430
2431 type = paragraph ,
2432     Name-sg = Paragraph ,
2433     name-sg = paragraph ,
2434     Name-pl = Paragraphs ,
2435     name-pl = paragraphs ,
2436     Name-sg-ab = Par. ,
2437     name-sg-ab = par. ,
2438     Name-pl-ab = Par. ,
2439     name-pl-ab = par. ,
2440
2441 type = appendix ,
2442     Name-sg = Appendix ,
2443     name-sg = appendix ,
2444     Name-pl = Appendices ,
2445     name-pl = appendices ,
2446
2447 type = page ,
2448     Name-sg = Page ,
2449     name-sg = page ,
2450     Name-pl = Pages ,
2451     name-pl = pages ,
2452     name-sg-ab = p. ,
2453     name-pl-ab = pp. ,
2454
2455 type = line ,
2456     Name-sg = Line ,
2457     name-sg = line ,
2458     Name-pl = Lines ,
2459     name-pl = lines ,
2460
2461 type = figure ,
2462     Name-sg = Figure ,
2463     name-sg = figure ,

```

```

2464 Name-pl = Figures ,
2465 name-pl = figures ,
2466 Name-sg-ab = Fig. ,
2467 name-sg-ab = fig. ,
2468 Name-pl-ab = Figs. ,
2469 name-pl-ab = figs. ,
2470
2471 type = table ,
2472 Name-sg = Table ,
2473 name-sg = table ,
2474 Name-pl = Tables ,
2475 name-pl = tables ,
2476
2477 type = item ,
2478 Name-sg = Item ,
2479 name-sg = item ,
2480 Name-pl = Items ,
2481 name-pl = items ,
2482
2483 type = footnote ,
2484 Name-sg = Footnote ,
2485 name-sg = footnote ,
2486 Name-pl = Footnotes ,
2487 name-pl = footnotes ,
2488
2489 type = note ,
2490 Name-sg = Note ,
2491 name-sg = note ,
2492 Name-pl = Notes ,
2493 name-pl = notes ,
2494
2495 type = equation ,
2496 Name-sg = Equation ,
2497 name-sg = equation ,
2498 Name-pl = Equations ,
2499 name-pl = equations ,
2500 Name-sg-ab = Eq. ,
2501 name-sg-ab = eq. ,
2502 Name-pl-ab = Eqs. ,
2503 name-pl-ab = eqs. ,
2504 refpre-in = { ( } ,
2505 refpos-in = { ) } ,
2506
2507 type = theorem ,
2508 Name-sg = Theorem ,
2509 name-sg = theorem ,
2510 Name-pl = Theorems ,
2511 name-pl = theorems ,
2512
2513 type = lemma ,
2514 Name-sg = Lemma ,
2515 name-sg = lemma ,
2516 Name-pl = Lemmas ,
2517 name-pl = lemmas ,

```

```

2518
2519 type = corollary ,
2520     Name-sg = Corollary ,
2521     name-sg = corollary ,
2522     Name-pl = Corollaries ,
2523     name-pl = corollaries ,
2524
2525 type = proposition ,
2526     Name-sg = Proposition ,
2527     name-sg = proposition ,
2528     Name-pl = Propositions ,
2529     name-pl = propositions ,
2530
2531 type = definition ,
2532     Name-sg = Definition ,
2533     name-sg = definition ,
2534     Name-pl = Definitions ,
2535     name-pl = definitions ,
2536
2537 type = proof ,
2538     Name-sg = Proof ,
2539     name-sg = proof ,
2540     Name-pl = Proofs ,
2541     name-pl = proofs ,
2542
2543 type = result ,
2544     Name-sg = Result ,
2545     name-sg = result ,
2546     Name-pl = Results ,
2547     name-pl = results ,
2548
2549 type = example ,
2550     Name-sg = Example ,
2551     name-sg = example ,
2552     Name-pl = Examples ,
2553     name-pl = examples ,
2554
2555 type = remark ,
2556     Name-sg = Remark ,
2557     name-sg = remark ,
2558     Name-pl = Remarks ,
2559     name-pl = remarks ,
2560
2561 type = algorithm ,
2562     Name-sg = Algorithm ,
2563     name-sg = algorithm ,
2564     Name-pl = Algorithms ,
2565     name-pl = algorithms ,
2566
2567 type = listing ,
2568     Name-sg = Listing ,
2569     name-sg = listing ,
2570     Name-pl = Listings ,
2571     name-pl = listings ,

```

```

2572
2573 type = exercise ,
2574     Name-sg = Exercise ,
2575     name-sg = exercise ,
2576     Name-pl = Exercises ,
2577     name-pl = exercises ,
2578
2579 type = solution ,
2580     Name-sg = Solution ,
2581     name-sg = solution ,
2582     Name-pl = Solutions ,
2583     name-pl = solutions ,
2584 </dict-english>

```

## 9.2 German

```

2585 <package>\zcDeclareLanguage { german }
2586 <package>\zcDeclareLanguageAlias { austrian      } { german }
2587 <package>\zcDeclareLanguageAlias { germanb       } { german }
2588 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2589 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2590 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2591 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2592 <*dict-german>
2593
2593 namesep = {\nobreakspace} ,
2594 pairsep  = {\und\nobreakspace} ,
2595 listsep  = {,~} ,
2596 lastsep  = {\und\nobreakspace} ,
2597 tpairsep = {\und\nobreakspace} ,
2598 tlistsep = {,~} ,
2599 tlastsep = {\und\nobreakspace} ,
2600 notesep  = {~} ,
2601 rangesep = {\bis\nobreakspace} ,
2602
2603 type = part ,
2604     Name-sg = Teil ,
2605     name-sg = Teil ,
2606     Name-pl = Teile ,
2607     name-pl = Teile ,
2608
2609 type = chapter ,
2610     Name-sg = Kapitel ,
2611     name-sg = Kapitel ,
2612     Name-pl = Kapitel ,
2613     name-pl = Kapitel ,
2614
2615 type = section ,
2616     Name-sg = Abschnitt ,
2617     name-sg = Abschnitt ,
2618     Name-pl = Abschnitte ,
2619     name-pl = Abschnitte ,
2620
2621 type = paragraph ,
2622     Name-sg = Absatz ,

```

```

2623     name-sg = Absatz ,
2624     Name-pl = Absätze ,
2625     name-pl = Absätze ,
2626
2627 type = appendix ,
2628     Name-sg = Anhang ,
2629     name-sg = Anhang ,
2630     Name-pl = Anhänge ,
2631     name-pl = Anhänge ,
2632
2633 type = page ,
2634     Name-sg = Seite ,
2635     name-sg = Seite ,
2636     Name-pl = Seiten ,
2637     name-pl = Seiten ,
2638
2639 type = line ,
2640     Name-sg = Zeile ,
2641     name-sg = Zeile ,
2642     Name-pl = Zeilen ,
2643     name-pl = Zeilen ,
2644
2645 type = figure ,
2646     Name-sg = Abbildung ,
2647     name-sg = Abbildung ,
2648     Name-pl = Abbildungen ,
2649     name-pl = Abbildungen ,
2650     Name-sg-ab = Abb. ,
2651     name-sg-ab = Abb. ,
2652     Name-pl-ab = Abb. ,
2653     name-pl-ab = Abb. ,
2654
2655 type = table ,
2656     Name-sg = Tabelle ,
2657     name-sg = Tabelle ,
2658     Name-pl = Tabellen ,
2659     name-pl = Tabellen ,
2660
2661 type = item ,
2662     Name-sg = Punkt ,
2663     name-sg = Punkt ,
2664     Name-pl = Punkte ,
2665     name-pl = Punkte ,
2666
2667 type = footnote ,
2668     Name-sg = Fußnote ,
2669     name-sg = Fußnote ,
2670     Name-pl = Fußnoten ,
2671     name-pl = Fußnoten ,
2672
2673 type = note ,
2674     Name-sg = Anmerkung ,
2675     name-sg = Anmerkung ,
2676     Name-pl = Anmerkungen ,

```



```

2677     name-pl = Anmerkungen ,
2678
2679 type = equation ,
2680     Name-sg = Gleichung ,
2681     name-sg = Gleichung ,
2682     Name-pl = Gleichungen ,
2683     name-pl = Gleichungen ,
2684     refpre-in = {()} ,
2685     refpos-in = {} ,
2686
2687 type = theorem ,
2688     Name-sg = Theorem ,
2689     name-sg = Theorem ,
2690     Name-pl = Theoreme ,
2691     name-pl = Theoreme ,
2692
2693 type = lemma ,
2694     Name-sg = Lemma ,
2695     name-sg = Lemma ,
2696     Name-pl = Lemmata ,
2697     name-pl = Lemmata ,
2698
2699 type = corollary ,
2700     Name-sg = Korollar ,
2701     name-sg = Korollar ,
2702     Name-pl = Korollare ,
2703     name-pl = Korollare ,
2704
2705 type = proposition ,
2706     Name-sg = Satz ,
2707     name-sg = Satz ,
2708     Name-pl = Sätze ,
2709     name-pl = Sätze ,
2710
2711 type = definition ,
2712     Name-sg = Definition ,
2713     name-sg = Definition ,
2714     Name-pl = Definitionen ,
2715     name-pl = Definitionen ,
2716
2717 type = proof ,
2718     Name-sg = Beweis ,
2719     name-sg = Beweis ,
2720     Name-pl = Beweise ,
2721     name-pl = Beweise ,
2722
2723 type = result ,
2724     Name-sg = Ergebnis ,
2725     name-sg = Ergebnis ,
2726     Name-pl = Ergebnisse ,
2727     name-pl = Ergebnisse ,
2728
2729 type = example ,
2730     Name-sg = Beispiel ,

```

```

2731 name-sg = Beispiel ,
2732 Name-pl = Beispiele ,
2733 name-pl = Beispiele ,
2734
2735 type = remark ,
2736 Name-sg = Bemerkung ,
2737 name-sg = Bemerkung ,
2738 Name-pl = Bemerkungen ,
2739 name-pl = Bemerkungen ,
2740
2741 type = algorithm ,
2742 Name-sg = Algorithmus ,
2743 name-sg = Algorithmus ,
2744 Name-pl = Algorithmen ,
2745 name-pl = Algorithmen ,
2746
2747 type = listing ,
2748 Name-sg = Listing , % CHECK
2749 name-sg = Listing , % CHECK
2750 Name-pl = Listings , % CHECK
2751 name-pl = Listings , % CHECK
2752
2753 type = exercise ,
2754 Name-sg = Übungsaufgabe ,
2755 name-sg = Übungsaufgabe ,
2756 Name-pl = Übungsaufgaben ,
2757 name-pl = Übungsaufgaben ,
2758
2759 type = solution ,
2760 Name-sg = Lösung ,
2761 name-sg = Lösung ,
2762 Name-pl = Lösungen ,
2763 name-pl = Lösungen ,
2764 </dict-german>

```

### 9.3 French

```

2765 <package>\zcDeclareLanguage { french }
2766 <package>\zcDeclareLanguageAlias { acadian } { french }
2767 <package>\zcDeclareLanguageAlias { canadien } { french }
2768 <package>\zcDeclareLanguageAlias { francais } { french }
2769 <package>\zcDeclareLanguageAlias { frenchb } { french }
2770 <*dict-french>
2771 namesep = {\nobreakspace} ,
2772 pairsep = {\~et\nobreakspace} ,
2773 listsep = {,~} ,
2774 lastsep = {\~et\nobreakspace} ,
2775 tpairsep = {\~et\nobreakspace} ,
2776 tlistsep = {,~} ,
2777 tlastsep = {\~et\nobreakspace} ,
2778 notesep = {\~} ,
2779 rangesep = {\~à\nobreakspace} ,
2780
2781 type = part ,

```

```

2782     Name-sg = Partie ,
2783     name-sg = partie ,
2784     Name-pl = Parties ,
2785     name-pl = parties ,
2786
2787 type = chapter ,
2788     Name-sg = Chapitre ,
2789     name-sg = chapitre ,
2790     Name-pl = Chapitres ,
2791     name-pl = chapitres ,
2792
2793 type = section ,
2794     Name-sg = Section ,
2795     name-sg = section ,
2796     Name-pl = Sections ,
2797     name-pl = sections ,
2798
2799 type = paragraph ,
2800     Name-sg = Paragraphe ,
2801     name-sg = paragraphe ,
2802     Name-pl = Paragraphes ,
2803     name-pl = paragraphes ,
2804
2805 type = appendix ,
2806     Name-sg = Annexe ,
2807     name-sg = annexe ,
2808     Name-pl = Annexes ,
2809     name-pl = annexes ,
2810
2811 type = page ,
2812     Name-sg = Page ,
2813     name-sg = page ,
2814     Name-pl = Pages ,
2815     name-pl = pages ,
2816
2817 type = line ,
2818     Name-sg = Ligne ,
2819     name-sg = ligne ,
2820     Name-pl = Lignes ,
2821     name-pl = lignes ,
2822
2823 type = figure ,
2824     Name-sg = Figure ,
2825     name-sg = figure ,
2826     Name-pl = Figures ,
2827     name-pl = figures ,
2828
2829 type = table ,
2830     Name-sg = Table ,
2831     name-sg = table ,
2832     Name-pl = Tables ,
2833     name-pl = tables ,
2834
2835 type = item ,

```

```

2836     Name-sg = Point ,
2837     name-sg = point ,
2838     Name-pl = Points ,
2839     name-pl = points ,
2840
2841 type = footnote ,
2842     Name-sg = Note ,
2843     name-sg = note ,
2844     Name-pl = Notes ,
2845     name-pl = notes ,
2846
2847 type = note ,
2848     Name-sg = Note ,
2849     name-sg = note ,
2850     Name-pl = Notes ,
2851     name-pl = notes ,
2852
2853 type = equation ,
2854     Name-sg = Équation ,
2855     name-sg = équation ,
2856     Name-pl = Équations ,
2857     name-pl = équations ,
2858     refpre-in = {()} ,
2859     refpos-in = {} } ,
2860
2861 type = theorem ,
2862     Name-sg = Théorème ,
2863     name-sg = théorème ,
2864     Name-pl = Théorèmes ,
2865     name-pl = théorèmes ,
2866
2867 type = lemma ,
2868     Name-sg = Lemme ,
2869     name-sg = lemme ,
2870     Name-pl = Lemmes ,
2871     name-pl = lemmes ,
2872
2873 type = corollary ,
2874     Name-sg = Corollaire ,
2875     name-sg = corollaire ,
2876     Name-pl = Corollaires ,
2877     name-pl = corollaires ,
2878
2879 type = proposition ,
2880     Name-sg = Proposition ,
2881     name-sg = proposition ,
2882     Name-pl = Propositions ,
2883     name-pl = propositions ,
2884
2885 type = definition ,
2886     Name-sg = Définition ,
2887     name-sg = définition ,
2888     Name-pl = Définitions ,
2889     name-pl = définitions ,

```

```

2890
2891 type = proof ,
2892   Name-sg = Démonstration ,
2893   name-sg = démonstration ,
2894   Name-pl = Démonstrations ,
2895   name-pl = démonstrations ,
2896
2897 type = result ,
2898   Name-sg = Résultat ,
2899   name-sg = résultat ,
2900   Name-pl = Résultats ,
2901   name-pl = résultats ,
2902
2903 type = example ,
2904   Name-sg = Exemple ,
2905   name-sg = exemple ,
2906   Name-pl = Exemples ,
2907   name-pl = exemples ,
2908
2909 type = remark ,
2910   Name-sg = Remarque ,
2911   name-sg = remarque ,
2912   Name-pl = Remarques ,
2913   name-pl = remarques ,
2914
2915 type = algorithm ,
2916   Name-sg = Algorithme ,
2917   name-sg = algorithme ,
2918   Name-pl = Algorithmes ,
2919   name-pl = algorithmes ,
2920
2921 type = listing ,
2922   Name-sg = Liste ,
2923   name-sg = liste ,
2924   Name-pl = Listes ,
2925   name-pl = listes ,
2926
2927 type = exercise ,
2928   Name-sg = Exercice ,
2929   name-sg = exercice ,
2930   Name-pl = Exercices ,
2931   name-pl = exercices ,
2932
2933 type = solution ,
2934   Name-sg = Solution ,
2935   name-sg = solution ,
2936   Name-pl = Solutions ,
2937   name-pl = solutions ,
2938 </dict-french>

```

## 9.4 Portuguese

```

2939 <package>\zcDeclareLanguage { portuguese }
2940 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
2941 <package>\zcDeclareLanguageAlias { brazil    } { portuguese }

```

```

2942 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
2943 <*dict-portuguese>
2944 namesep = {\nobreakspace} ,
2945 pairsep = {\~e\nobreakspace} ,
2946 listsep = {,~} ,
2947 lastsep = {\~e\nobreakspace} ,
2948 tpairsep = {\~e\nobreakspace} ,
2949 tlistsep = {,~} ,
2950 tlastsep = {\~e\nobreakspace} ,
2951 notesep = {\~} ,
2952 rangesep = {\~a\nobreakspace} ,
2953
2954 type = part ,
2955   Name-sg = Parte ,
2956   name-sg = parte ,
2957   Name-pl = Partes ,
2958   name-pl = partes ,
2959
2960 type = chapter ,
2961   Name-sg = Capítulo ,
2962   name-sg = capítulo ,
2963   Name-pl = Capítulos ,
2964   name-pl = capítulos ,
2965
2966 type = section ,
2967   Name-sg = Seção ,
2968   name-sg = seção ,
2969   Name-pl = Seções ,
2970   name-pl = seções ,
2971
2972 type = paragraph ,
2973   Name-sg = Parágrafo ,
2974   name-sg = parágrafo ,
2975   Name-pl = Parágrafos ,
2976   name-pl = parágrafos ,
2977   Name-sg-ab = Par. ,
2978   name-sg-ab = par. ,
2979   Name-pl-ab = Par. ,
2980   name-pl-ab = par. ,
2981
2982 type = appendix ,
2983   Name-sg = Apêndice ,
2984   name-sg = apêndice ,
2985   Name-pl = Apêndices ,
2986   name-pl = apêndices ,
2987
2988 type = page ,
2989   Name-sg = Página ,
2990   name-sg = página ,
2991   Name-pl = Páginas ,
2992   name-pl = páginas ,
2993   name-sg-ab = p. ,
2994   name-pl-ab = pp. ,

```

```

2995
2996 type = line ,
2997     Name-sg = Linha ,
2998     name-sg = linha ,
2999     Name-pl = Linhas ,
3000     name-pl = linhas ,
3001
3002 type = figure ,
3003     Name-sg = Figura ,
3004     name-sg = figura ,
3005     Name-pl = Figuras ,
3006     name-pl = figuras ,
3007     Name-sg-ab = Fig. ,
3008     name-sg-ab = fig. ,
3009     Name-pl-ab = Figs. ,
3010     name-pl-ab = figs. ,
3011
3012 type = table ,
3013     Name-sg = Tabela ,
3014     name-sg = tabela ,
3015     Name-pl = Tabelas ,
3016     name-pl = tabelas ,
3017
3018 type = item ,
3019     Name-sg = Item ,
3020     name-sg = item ,
3021     Name-pl = Itens ,
3022     name-pl = itens ,
3023
3024 type = footnote ,
3025     Name-sg = Nota ,
3026     name-sg = nota ,
3027     Name-pl = Notas ,
3028     name-pl = notas ,
3029
3030 type = note ,
3031     Name-sg = Nota ,
3032     name-sg = nota ,
3033     Name-pl = Notas ,
3034     name-pl = notas ,
3035
3036 type = equation ,
3037     Name-sg = Equação ,
3038     name-sg = equação ,
3039     Name-pl = Equações ,
3040     name-pl = equações ,
3041     Name-sg-ab = Eq. ,
3042     name-sg-ab = eq. ,
3043     Name-pl-ab = Eqs. ,
3044     name-pl-ab = eqs. ,
3045     refpre-in = {} ,
3046     refpos-in = {} ,
3047
3048 type = theorem ,

```

```

3049     Name-sg = Teorema ,
3050     name-sg = teorema ,
3051     Name-pl = Teoremas ,
3052     name-pl = teoremas ,
3053
3054     type = lemma ,
3055     Name-sg = Lema ,
3056     name-sg = lema ,
3057     Name-pl = Lemas ,
3058     name-pl = lemas ,
3059
3060     type = corollary ,
3061     Name-sg = Corolário ,
3062     name-sg = corolário ,
3063     Name-pl = Corolários ,
3064     name-pl = corolários ,
3065
3066     type = proposition ,
3067     Name-sg = Proposição ,
3068     name-sg = proposição ,
3069     Name-pl = Proposições ,
3070     name-pl = proposições ,
3071
3072     type = definition ,
3073     Name-sg = Definição ,
3074     name-sg = definição ,
3075     Name-pl = Definições ,
3076     name-pl = definições ,
3077
3078     type = proof ,
3079     Name-sg = Demonstração ,
3080     name-sg = demonstração ,
3081     Name-pl = Demonstrações ,
3082     name-pl = demonstrações ,
3083
3084     type = result ,
3085     Name-sg = Resultado ,
3086     name-sg = resultado ,
3087     Name-pl = Resultados ,
3088     name-pl = resultados ,
3089
3090     type = example ,
3091     Name-sg = Exemplo ,
3092     name-sg = exemplo ,
3093     Name-pl = Exemplos ,
3094     name-pl = exemplos ,
3095
3096     type = remark ,
3097     Name-sg = Observação ,
3098     name-sg = observação ,
3099     Name-pl = Observações ,
3100     name-pl = observações ,
3101
3102     type = algorithm ,

```



```

3103   Name-sg = Algoritmo ,
3104   name-sg = algoritmo ,
3105   Name-pl = Algoritmos ,
3106   name-pl = algoritmos ,
3107
3108   type = listing ,
3109   Name-sg = Listagem ,
3110   name-sg = listagem ,
3111   Name-pl = Listagens ,
3112   name-pl = listagens ,
3113
3114   type = exercise ,
3115   Name-sg = Exercício ,
3116   name-sg = exercício ,
3117   Name-pl = Exercícios ,
3118   name-pl = exercícios ,
3119
3120   type = solution ,
3121   Name-sg = Solução ,
3122   name-sg = solução ,
3123   Name-pl = Soluções ,
3124   name-pl = soluções ,
3125 </dict-portuguese>

```

## 9.5 Spanish

```

3126 <package>\zcDeclareLanguage { spanish }
3127 <*dict-spanish>
3128 namesep = {\nobreakspace} ,
3129 pairsep = {\~y\nobreakspace} ,
3130 listsep = {,~} ,
3131 lastsep = {\~y\nobreakspace} ,
3132 tpairsep = {\~y\nobreakspace} ,
3133 tlistsep = {,~} ,
3134 tlastsep = {\~y\nobreakspace} ,
3135 notesep = {\~} ,
3136 rangesep = {\~a\nobreakspace} ,
3137
3138   type = part ,
3139   Name-sg = Parte ,
3140   name-sg = parte ,
3141   Name-pl = Partes ,
3142   name-pl = partes ,
3143
3144   type = chapter ,
3145   Name-sg = Capítulo ,
3146   name-sg = capítulo ,
3147   Name-pl = Capítulos ,
3148   name-pl = capítulos ,
3149
3150   type = section ,
3151   Name-sg = Sección ,
3152   name-sg = sección ,
3153   Name-pl = Secciones ,

```

```

3154     name-pl = secciones ,
3155
3156 type = paragraph ,
3157     Name-sg = Párrafo ,
3158     name-sg = párrafo ,
3159     Name-pl = Párrafos ,
3160     name-pl = párrafos ,
3161
3162 type = appendix ,
3163     Name-sg = Apéndice ,
3164     name-sg = apéndice ,
3165     Name-pl = Apéndices ,
3166     name-pl = apéndices ,
3167
3168 type = page ,
3169     Name-sg = Página ,
3170     name-sg = página ,
3171     Name-pl = Páginas ,
3172     name-pl = páginas ,
3173
3174 type = line ,
3175     Name-sg = Línea ,
3176     name-sg = línea ,
3177     Name-pl = Líneas ,
3178     name-pl = líneas ,
3179
3180 type = figure ,
3181     Name-sg = Figura ,
3182     name-sg = figura ,
3183     Name-pl = Figuras ,
3184     name-pl = figuras ,
3185
3186 type = table ,
3187     Name-sg = Cuadro ,
3188     name-sg = cuadro ,
3189     Name-pl = Cuadros ,
3190     name-pl = cuadros ,
3191
3192 type = item ,
3193     Name-sg = Punto ,
3194     name-sg = punto ,
3195     Name-pl = Puntos ,
3196     name-pl = puntos ,
3197
3198 type = footnote ,
3199     Name-sg = Nota ,
3200     name-sg = nota ,
3201     Name-pl = Notas ,
3202     name-pl = notas ,
3203
3204 type = note ,
3205     Name-sg = Nota ,
3206     name-sg = nota ,
3207     Name-pl = Notas ,

```

```

3208     name-pl = notas ,
3209
3210 type = equation ,
3211     Name-sg = Ecuación ,
3212     name-sg = ecuación ,
3213     Name-pl = Ecuaciones ,
3214     name-pl = ecuaciones ,
3215     refpre-in = {()} ,
3216     refpos-in = {} ,
3217
3218 type = theorem ,
3219     Name-sg = Teorema ,
3220     name-sg = teorema ,
3221     Name-pl = Teoremas ,
3222     name-pl = teoremas ,
3223
3224 type = lemma ,
3225     Name-sg = Lema ,
3226     name-sg = lema ,
3227     Name-pl = Lemas ,
3228     name-pl = lemas ,
3229
3230 type = corollary ,
3231     Name-sg = Corolario ,
3232     name-sg = corolario ,
3233     Name-pl = Corolarios ,
3234     name-pl = corolarios ,
3235
3236 type = proposition ,
3237     Name-sg = Proposición ,
3238     name-sg = proposición ,
3239     Name-pl = Propositiones ,
3240     name-pl = proposiciones ,
3241
3242 type = definition ,
3243     Name-sg = Definición ,
3244     name-sg = definición ,
3245     Name-pl = Definiciones ,
3246     name-pl = definiciones ,
3247
3248 type = proof ,
3249     Name-sg = Demostración ,
3250     name-sg = demostración ,
3251     Name-pl = Demostraciones ,
3252     name-pl = demostraciones ,
3253
3254 type = result ,
3255     Name-sg = Resultado ,
3256     name-sg = resultado ,
3257     Name-pl = Resultados ,
3258     name-pl = resultados ,
3259
3260 type = example ,
3261     Name-sg = Ejemplo ,

```

```

3262   name-sg = ejemplo ,
3263   Name-pl = Ejemplos ,
3264   name-pl = ejemplos ,
3265
3266   type = remark ,
3267   Name-sg = Observación ,
3268   name-sg = observación ,
3269   Name-pl = Observaciones ,
3270   name-pl = observaciones ,
3271
3272   type = algorithm ,
3273   Name-sg = Algoritmo ,
3274   name-sg = algoritmo ,
3275   Name-pl = Algoritmos ,
3276   name-pl = algoritmos ,
3277
3278   type = listing ,
3279   Name-sg = Listado ,
3280   name-sg = listado ,
3281   Name-pl = Listados ,
3282   name-pl = listados ,
3283
3284   type = exercise ,
3285   Name-sg = Ejercicio ,
3286   name-sg = ejercicio ,
3287   Name-pl = Ejercicios ,
3288   name-pl = ejercicios ,
3289
3290   type = solution ,
3291   Name-sg = Solución ,
3292   name-sg = solución ,
3293   Name-pl = Soluciones ,
3294   name-pl = soluciones ,
3295 </dict-spanish>

```

## Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

| Symbols                                    |   |
|--|---|
| <code>\</code> . . . . .                   | <a href="#">103</a> , <a href="#">109</a> , <a href="#">120</a> , <a href="#">125</a> , <a href="#">126</a> , <a href="#">135</a> , <a href="#">145</a>   |
| A  |   |
| <code>\AddToHook</code> . . . . .          | <a href="#">91</a> ,<br><a href="#">517</a> , <a href="#">532</a> , <a href="#">642</a> , <a href="#">710</a> , <a href="#">734</a> , <a href="#">763</a> , <a href="#">765</a> , <a href="#">816</a>   |
| <code>\appendix</code> . . . . .           | <a href="#">59</a>  |
| <code>\appendixname</code> . . . . .       | <a href="#">59</a>  |
| <code>\Arg</code> . . . . .                | <a href="#">2277</a>  |
| B  |   |
| <code>\babelname</code> . . . . .          | <a href="#">720</a>   |
| <code>\babelprovide</code> . . . . .       | <a href="#">21</a>  |
| bool commands:                             |   |
| <code>\bool_case_true:</code> . . . . .    | <a href="#">2</a>   |
| <code>\bool_if:NTF</code> . . . . .        | <a href="#">271</a> , <a href="#">280</a> , <a href="#">646</a> , <a href="#">650</a> ,<br><a href="#">1489</a> , <a href="#">1562</a> , <a href="#">1687</a> , <a href="#">1708</a> , <a href="#">1739</a> , <a href="#">1794</a> ,<br><a href="#">1861</a> , <a href="#">1865</a> , <a href="#">1872</a> , <a href="#">1881</a> , <a href="#">1887</a> , <a href="#">2141</a>   |
| <code>\bool_if:nTF</code> . . . . .        | <a href="#">59</a> , <a href="#">1128</a> , <a href="#">1137</a> , <a href="#">1146</a> ,<br><a href="#">1217</a> , <a href="#">1245</a> , <a href="#">1268</a> , <a href="#">1357</a> , <a href="#">1365</a> , <a href="#">1503</a> ,<br><a href="#">1511</a> , <a href="#">1720</a> , <a href="#">1727</a> , <a href="#">1734</a> , <a href="#">1983</a> , <a href="#">2219</a> |
| <code>\bool_lazy_all:nTF</code> . . . . .  | <a href="#">2284</a>  |
| <code>\bool_lazy_and:nnTF</code> . . . . . | <a href="#">1021</a> , <a href="#">1032</a> , <a href="#">1836</a> , <a href="#">2045</a> , <a href="#">2326</a>  |

|  |                                |  |
|--|--------------------------------|--|
| \bool_lazy_any:nTF . . . . .           | 2109, 2118                     | 2210, 2213, 2225, 2228, 2231, 2254,          |
| \bool_lazy_or:nnTF . . . . .           | 1025, 2033                     | 2256, 2259, 2262, 2268, 2270, 2273           |
| \bool_new:N . . . . .                  | 241, 553,                      | \exp_not:n . . . . . 1593, 1609, 1621, 1625, |
| 554, 579, 603, 612, 619, 620, 675,     |                                | 1645, 1658, 1661, 1673, 1676, 1709,          |
| 676, 693, 694, 809, 810, 1043, 1060,   |                                | 1710, 1742, 1763, 1768, 1769, 1901,          |
| 1397, 1398, 1409, 1410, 1411, 1430     |                                | 1914, 1919, 1939, 1950, 1953, 1963,          |
| \bool_set:Nn . . . . .                 | 1019                           | 1966, 1987, 1988, 1990, 2000, 2002,          |
| \bool_set_false:N . . . . .            |                                | 2005, 2010, 2011, 2013, 2014, 2016,          |
| 566, 570, 627, 636, 637,               |                                | 2018, 2166, 2167, 2169, 2171, 2172,          |
| 652, 831, 1206, 1456, 1495, 1509,      |                                | 2174, 2175, 2178, 2190, 2191, 2196,          |
| 1520, 1699, 1834, 1835, 2116, 2133     |                                | 2197, 2199, 2207, 2211, 2212, 2214,          |
| \bool_set_true:N . . . . .             |                                | 2226, 2227, 2229, 2249, 2252, 2255,          |
| 290, 560, 561, 565, 571, 626, 631,     |                                | 2260, 2261, 2263, 2264, 2267, 2269           |
| 632, 820, 825, 1229, 1240, 1257,       |                                | \ExplSyntaxOn . . . . . 257                  |
| 1263, 1280, 1286, 1312, 1324, 1463,    |                                |  |
| 1490, 1496, 1500, 1521, 1524, 2132,    |                                |  |
| 2352, 2359, 2360, 2378, 2385, 2386     |                                |  |
| \bool_until_do:Nn . . . . .            | 1210, 1457                     |  |
| <b>C</b>                               |                                |  |
| clist commands:                        |                                |  |
| \clist_map_inline:nn . . . . .         | 462                            |  |
| \counterwithin . . . . .               | 4                              |  |
| \cs . . . . .                          | 1336, 1685, 2280, 2322         |  |
| cs commands:                           |                                |  |
| \cs_generate_variant:Nn . . . . .      |                                |  |
| 55, 56, 285, 294, 937, 945, 1061, 2024 |                                |  |
| \cs_if_exist:NnTF . . . . .            | 39, 48, 69                     |  |
| \cs_new:Npn . . . . .                  | 37, 46, 57, 67, 78, 1979, 2136 |  |
| \cs_new_protected:Npn . . . . .        | 244, 287,                      |  |
| 297, 305, 423, 932, 940, 1014, 1062,   |                                |  |
| 1078, 1121, 1187, 1332, 1388, 1433,    |                                |  |
| 1573, 1829, 2025, 2278, 2320, 2340     |                                |  |
| \cs_new_protected:Npx . . . . .        | 90                             |  |
| \cs_set_eq:NN . . . . .                | 94                             |  |
| <b>E</b>                               |                                |  |
| \endinput . . . . .                    | 12                             |  |
| exp commands:                          |                                |  |
| \exp_args:NNe . . . . .                | 27                             |  |
| \exp_args:NNnx . . . . .               | 234                            |  |
| \exp_args:NnV . . . . .                | 263                            |  |
| \exp_args:NNx . . . . .                | 95, 1254, 1277                 |  |
| \exp_args:Nnx . . . . .                | 299                            |  |
| \exp_args:Nx . . . . .                 | 255                            |  |
| \exp_args:Nxx . . . . .                |                                |  |
| 1171, 1225, 2344, 2366, 2370           |                                |  |
| \exp_not:N . . . . .                   |                                |  |
| 1741, 1744, 1764, 1767, 1770,          |                                |  |
| 1986, 1989, 1992, 2004, 2006, 2009,    |                                |  |
| 2012, 2017, 2019, 2022, 2139, 2147,    |                                |  |
| 2165, 2168, 2170, 2173, 2179, 2186,    |                                |  |
| 2188, 2192, 2195, 2198, 2200, 2206,    |                                |  |
| 2210, 2213, 2225, 2228, 2231, 2254,    |                                |  |
| 2256, 2259, 2262, 2268, 2270, 2273     |                                |  |
| \exp_not:n . . . . .                   | 1593, 1609, 1621, 1625,        |  |
| 1645, 1658, 1661, 1673, 1676, 1709,    |                                |  |
| 1710, 1742, 1763, 1768, 1769, 1901,    |                                |  |
| 1914, 1919, 1939, 1950, 1953, 1963,    |                                |  |
| 1966, 1987, 1988, 1990, 2000, 2002,    |                                |  |
| 2005, 2010, 2011, 2013, 2014, 2016,    |                                |  |
| 2018, 2166, 2167, 2169, 2171, 2172,    |                                |  |
| 2174, 2175, 2178, 2190, 2191, 2196,    |                                |  |
| 2197, 2199, 2207, 2211, 2212, 2214,    |                                |  |
| 2226, 2227, 2229, 2249, 2252, 2255,    |                                |  |
| 2260, 2261, 2263, 2264, 2267, 2269     |                                |  |
| \ExplSyntaxOn . . . . .                | 257                            |  |
| <b>F</b>                               |                                |  |
| file commands:                         |                                |  |
| \file_get:nnNTF . . . . .              | 255                            |  |
| \fmtversion . . . . .                  | 3                              |  |
| <b>G</b>                               |                                |  |
| group commands:                        |                                |  |
| \group_begin: . . . . .                | 93, 246, 289,                  |  |
| 911, 1016, 1741, 1767, 1986, 1989,     |                                |  |
| 2009, 2012, 2165, 2170, 2173, 2188,    |                                |  |
| 2195, 2210, 2225, 2228, 2259, 2262     |                                |  |
| \group_end: . . . . .                  | 96, 283, 292,                  |  |
| 919, 1040, 1764, 1770, 2004, 2006,     |                                |  |
| 2017, 2019, 2168, 2179, 2186, 2192,    |                                |  |
| 2198, 2213, 2254, 2256, 2268, 2270     |                                |  |
| <b>H</b>                               |                                |  |
| \hyperlink . . . . .                   | 50                             |  |
| <b>I</b>                               |                                |  |
| \IfBooleanTF . . . . .                 | 1046                           |  |
| \IfFormatAtLeastTF . . . . .           | 3, 4                           |  |
| int commands:                          |                                |  |
| \int_case:nnTF . . . . .               |                                |  |
| 1576, 1602, 1633, 1797, 1894, 1930     |                                |  |
| \int_compare:nNnTF . . . . .           | 1175,                          |  |
| 1230, 1297, 1313, 1343, 1345, 1390,    |                                |  |
| 1544, 1589, 1623, 1786, 1788, 1851,    |                                |  |
| 1875, 1917, 2348, 2354, 2374, 2380     |                                |  |
| \int_compare_p:nNn . . . . .           |                                |  |
| 1359, 1367, 2037, 2048, 2129           |                                |  |
| \int_eval:n . . . . .                  | 90                             |  |
| \int_incr:N . . . . .                  | 1824, 1864,                    |  |
| 1866, 1880, 1882, 1886, 1888, 1977     |                                |  |
| \int_new:N . . . . .                   |                                |  |
| 1058, 1059, 1404, 1405, 1406, 1407     |                                |  |
| \int_set:Nn . . . . .                  | 1344, 1346, 1350, 1353         |  |
| \int_use:N . . . . .                   | 33, 35, 50                     |  |



|   |                    |  |                    |
|---|--------------------|--|--------------------|
| <code>\seq_reverse:N</code> .....                   | 594                | <code>\zref@ifrefundefined</code> .....                  |                    |
| <code>\seq_set_eq:NN</code> .....                   | 1435               | 1088, 1090, 1102, 1492, 1494, 1499,                      |                    |
| <code>\seq_set_from_clist:Nn</code> ....            | 593, 1018          | 1532, 1691, 1700, 1842, 2027, 2138                       |                    |
| <code>\seq_sort:Nn</code> .....                     | 1086               | <code>\ZREF@mainlist</code> 22, 32, 34, 36, 87, 88, 99   |                    |
| sort commands:                                      |                    | <code>\zref@newprop</code> 4, 21, 23, 33, 35, 83, 85, 98 |                    |
| <code>\sort_return_same:</code> .....               |                    | <code>\zref@refused</code> .....                         | 1531               |
| ..... 32, 38, 1093, 1098, 1135,                     |                    | <code>\zref@wrapper@babel</code> .....                   | 29, 1013           |
| 1180, 1182, 1235, 1241, 1258, 1264,                 |                    | <code>\textendash</code> .....                           | 417                |
| 1287, 1318, 1325, 1363, 1379, 1395                  |                    | <code>\the</code> .....                                  | 3                  |
| <code>\sort_return_swapped:</code> .....            |                    | <code>\thechapter</code> .....                           | 59                 |
| ..... 32, 38, 1106, 1144, 1179,                     |                    | <code>\thepage</code> .....                              | 6, 95              |
| 1234, 1281, 1317, 1371, 1382, 1394                  |                    | <code>\thesection</code> .....                           | 59                 |
| str commands:                                       |                    | tl commands:   |                    |
| <code>\str_case:nnTF</code> .....                   | 736, 772           | <code>\c_empty_tl</code> .....                           | 1065, 1124, 1126,  |
| <code>\str_if_eq:nnTF</code> .....                  | 80, 216            | 1190, 1194, 1198, 1202, 1476, 1481                       |                    |
| <code>\str_if_eq_p:nn</code> 2114, 2120, 2122, 2126 |                    | <code>\c_novalue_tl</code> .....                         | 845, 886           |
| <code>\str_new:N</code> .....                       | 660                | <code>\tl_clear:N</code> .....                           | 254, 262,          |
| <code>\str_set:Nn</code> .....                      | 665, 667, 669, 671 | 316, 915, 927, 1208, 1209, 1436,                         |                    |
|   |                    | 1437, 1438, 1439, 1440, 1462, 1819,                      |                    |
|   |                    | 1820, 1821, 1822, 1863, 2028, 2031,                      |                    |
|   |                    | 2055, 2070, 2099, 2314, 2335, 2337                       |                    |
|   |                    | <code>\tl_gset:Nn</code> .....                           | 95                 |
|   |                    | <code>\tl_head:N</code> .....                            |                    |
|   |                    | .. 1213, 1215, 1298, 1300, 1314, 1316                    |                    |
|   |                    | <code>\tl_if_empty:N</code> .....                        |                    |
|   |                    | ..... 71, 328, 345, 359, 954, 975,                       |                    |
|   |                    | 998, 1066, 1535, 1689, 2042, 2057, 2204                  |                    |
|   |                    | <code>\tl_if_empty:nTF</code> .....                      |                    |
|   |                    | .. 212, 315, 425, 926, 1641, 1656,                       |                    |
|   |                    | 1671, 1912, 1937, 1948, 1961, 2030                       |                    |
|   |                    | <code>\tl_if_empty_p:N</code> . 1132, 1133, 1141,        |                    |
|   |                    | 1142, 1149, 1150, 1506, 1507, 1514,                      |                    |
|   |                    | 1515, 2113, 2123, 2127, 2286, 2327                       |                    |
|   |                    | <code>\tl_if_empty_p:n</code> 1221, 1222, 1249, 1272     |                    |
|   |                    | <code>\tl_if_eq:NNTF</code> .....                        | 1156, 1295, 1518   |
|   |                    | <code>\tl_if_eq:NnTF</code> .....                        | 1081, 1114,        |
|   |                    | 1349, 1352, 1378, 1381, 1467, 2342                       |                    |
|   |                    | <code>\tl_if_eq:nnTF</code> .....                        |                    |
|   |                    | .. 1171, 1225, 1341, 2344, 2366, 2370                    |                    |
|   |                    | <code>\tl_if_in:NnTF</code> .....                        | 1254, 1277         |
|   |                    | <code>\tl_if_novalue:nTF</code> .....                    | 848, 889           |
|   |                    | <code>\tl_map_break:n</code> .....                       | 81                 |
|   |                    | <code>\tl_map_tokens:Nn</code> .....                     | 73                 |
|   |                    | <code>\tl_new:N</code> .....                             | 89, 164, 165, 507, |
|   |                    | 707, 708, 709, 800, 803, 1050, 1051,                     |                    |
|   |                    | 1052, 1053, 1054, 1055, 1056, 1057,                      |                    |
|   |                    | 1400, 1401, 1402, 1403, 1408, 1412,                      |                    |
|   |                    | 1413, 1414, 1416, 1417, 1418, 1419,                      |                    |
|   |                    | 1420, 1421, 1422, 1423, 1424, 1425,                      |                    |
|   |                    | 1426, 1427, 1428, 1429, 1431, 1432                       |                    |
|   |                    | <code>\tl_put_left:Nn</code> ....                        | 1723, 1730, 1779   |
|   |                    | <code>\tl_put_right:Nn</code> .....                      | 1591, 1607,        |
|   |                    | 1616, 1643, 1653, 1668, 1899, 1910,                      |                    |
|   |                    | 1935, 1946, 1959, 2043, 2044, 2053                       |                    |

## T

T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> 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> .....                  |                        |
| ..... 519, 534, 644, 712, 718, 818                    |                        |
| <code>\@onlypreamble</code> .....                     | 227, 239, 921          |
| <code>\bbl@loaded</code> .....                        | 21                     |
| <code>\bbl@main@language</code> .....                 | 21, 715                |
| <code>\c@</code> .....                                | 3                      |
| <code>\c@page</code> .....                            | 6, 94                  |
| <code>\cl@</code> .....                               | 4                      |
| <code>\hyper@@link</code> ..                          | 1744, 1992, 2147, 2231 |
| <code>\p@...</code> .....                             | 3                      |
| <code>\zref@addprop</code> 22, 32, 34, 36, 87, 88, 99 |                        |
| <code>\zref@default</code> .....                      |                        |
| ..... 50, 2022, 2139, 2200, 2206, 2273                |                        |
| <code>\zref@extractdefault</code> .....               |                        |
| ..... 50, 1065, 1124, 1126, 1172,                     |                        |
| 1173, 1176, 1178, 1190, 1194, 1198,                   |                        |
| 1202, 1226, 1227, 1231, 1233, 1253,                   |                        |
| 1276, 1391, 1393, 1475, 1480, 1749,                   |                        |
| 1754, 1760, 1995, 1996, 1998, 2001,                   |                        |
| 2015, 2152, 2156, 2161, 2176, 2236,                   |                        |
| 2240, 2245, 2250, 2265, 2345, 2346,                   |                        |
| 2349, 2351, 2355, 2357, 2367, 2368,                   |                        |
| 2371, 2372, 2375, 2377, 2381, 2383                    |                        |
| <code>\zref@ifpropundefined</code> .....              | 16                     |
| <code>\zref@ifrefcontainsprop</code> . 16, 1746,      |                        |
| 1981, 1994, 2143, 2149, 2216, 2233                    |                        |

|  |                        |   |   |
|--|------------------------|---|---|
| <code>\tl_reverse_items:n</code> .....             |                        | <code>\l_zrefclever_counter_resetby_</code>       |   |
| ..... 1061, 1192, 1196, 1200, 1204                 |                        | prop .....  | 4, 15, 60, 61, 483, 495                 |
| <code>\tl_set:Nn</code> . 317, 512, 514, 520, 523, |                        | <code>\l_zrefclever_counter_resettters_</code>    |   |
| 539, 548, 714, 715, 720, 721, 724,                 |                        | seq .....   | 4, 15, 63, 457, 464, 467                |
| 725, 728, 740, 748, 755, 776, 784,                 |                        | <code>\l_zrefclever_counter_type_prop</code>      |   |
| 791, 866, 928, 1064, 1123, 1125,                   |                        | .....   | 3, 14, 25, 28, 429, 441                 |
| 1189, 1191, 1193, 1195, 1197, 1199,                |                        | <code>\l_zrefclever_current_language_</code>      |   |
| 1201, 1203, 1212, 1214, 1252, 1275,                |                        | tl ..   | 21, 709, 714, 720, 724, 749, 785        |
| 1302, 1304, 1306, 1308, 1469, 1470,                |                        | <code>\__zrefclever_declare_default_</code>       |   |
| 1473, 1478, 1582, 1583, 1706, 1737,                |                        | transl:nnn ....                                   | 940, 945, 956, 977                      |
| 1855, 1856, 1879, 2039, 2040, 2052                 |                        | <code>\__zrefclever_declare_type_</code>          |   |
| <code>\tl_set_eq:NN</code> .....                   | 1817                   | transl:nnnn ...                                   | 932, 937, 982, 1004                     |
| <code>\tl_tail:N</code> ....                       | 1303, 1305, 1307, 1309 | <code>\l_zrefclever_dict_language_tl</code> .     |   |
| <code>\l_tmpa_tl</code> ....                       | 254, 258, 264, 1208,   | .....   | 164, 248, 252, 256, 260,                |
| 1212, 1221, 1249, 1252, 1255, 1295                 |                        |   | 261, 266, 268, 274, 300, 308, 372,      |
| <code>\l_tmpb_tl</code> .....                      | 1209,                  |   | 374, 387, 389, 913, 957, 978, 983, 1005 |
| 1214, 1222, 1272, 1275, 1278, 1295                 |                        | <code>\g_zrefclever_fallback_dict_</code>         |   |
|  |                        | prop .....  | 401, 406, 407                           |
|  |                        | <code>\__zrefclever_get_default_</code>           |   |
|  |                        | transl:nnN .....                                  | 9, 384, 396                             |
|  |                        | <code>\__zrefclever_get_default_</code>           |   |
|  |                        | transl:nnNTF .....                                | 2308                                    |
|  |                        | <code>\__zrefclever_get_enclosing_</code>         |   |
|  |                        | counters:n .....                                  | 4, 5, 37, 42, 84                        |
|  |                        | <code>\__zrefclever_get_enclosing_</code>         |   |
|  |                        | counters_value:n .                                | 4, 5, 37, 51, 86                        |
|  |                        | <code>\__zrefclever_get_fallback_</code>          |   |
|  |                        | transl:nN .....                                   | 399                                     |
|  |                        | <code>\__zrefclever_get_fallback_</code>          |   |
|  |                        | transl:nNTF .....                                 | 2313                                    |
|  |                        | <code>\__zrefclever_get_option_</code>            |   |
|  |                        | plain:nN  | 8, 24, 1546, 1547, 1548, 2320           |
|  |                        | <code>\__zrefclever_get_option_with_</code>       |   |
|  |                        | transl:nN   | 8, 13, 24, 1447, 1448,                  |
|  |                        |   | 1449, 1450, 1549, 1550, 1551, 1552,     |
|  |                        |   | 1553, 1554, 1555, 1556, 1557, 2277      |
|  |                        | <code>\__zrefclever_get_ref:n</code>              | 1594, 1610,                             |
|  |                        |   | 1622, 1626, 1646, 1659, 1662, 1674,     |
|  |                        |   | 1677, 1711, 1731, 1902, 1915, 1920,     |
|  |                        |   | 1940, 1951, 1954, 1964, 1967, 1979      |
|  |                        | <code>\__zrefclever_get_ref_first:</code> ...     |   |
|  |                        | .....   | 40, 50, 1724, 1780, 2136                |
|  |                        | <code>\__zrefclever_get_type_transl:nnnN</code>   |   |
|  |                        | .....   | 9, 369, 381                             |
|  |                        | <code>\__zrefclever_get_type_transl:nnnNTF</code> |   |
|  |                        | .....   | 2064, 2087, 2093, 2302                  |
|  |                        | <code>\l_zrefclever_label_a_tl</code> .....       |   |
|  |                        | ...   | 1050, 1459, 1476, 1492, 1531,           |
|  |                        |   | 1532, 1538, 1582, 1594, 1610, 1626,     |
|  |                        |   | 1662, 1677, 1704, 1711, 1842, 1846,     |
|  |                        |   | 1855, 1879, 1902, 1920, 1954, 1967      |
|  |                        | <code>\l_zrefclever_label_b_tl</code> .           | 1050,                                   |
|  |                        |   | 1462, 1465, 1481, 1494, 1499, 1846      |

U

use commands:

`\use:N` .....

Z

`\zcDeclareLanguage` .....

210, 227, 2394, 2585, 2765, 2939, 3126

`\zcDeclareLanguageAlias` .....

..... 230, 239, 2395, 2396,

2397, 2398, 2399, 2400, 2401, 2586,

2587, 2588, 2589, 2590, 2591, 2766,

2767, 2768, 2769, 2940, 2941, 2942

`\zcDeclareTranslations` . 8, 9, 25–27, 909

`\zcpageref` .....

30, 1044

`\zceref` 8, 24, 29, 31, 38, 40, 1012, 1047, 1048

`\zcRefTypeSetup` .....

8, 25, 862

`\zcsetup` .....

8, 21, 24, 25, 860

zrefcheck commands:

`\zrefcheck_zceref_beg_label:` ..

1024

`\zrefcheck_zceref_end_label_`

maybe: .....

1036

`\zrefcheck_zceref_run_checks_on_`

labels:n .....

1037

zrefclever internal commands:

`\l_zrefclever_abbrev_bool` .....

..... 693, 697, 2046

`\l_zrefclever_capitalize_bool` ..

..... 675, 679, 2034

`\l_zrefclever_capitalize_first_`

bool .....

676, 685, 2036

`\__zrefclever_counter_reset_by:n`

..... 5, 15, 39, 41, 43, 48, 50, 52, 57

`\__zrefclever_counter_reset_by_`

aux:nn .....

64, 67

`\__zrefclever_counter_reset_by_`

auxi:nnn .....

74, 78



```

\l_zrefclever_label_count_int . .
    . . . . . 39, 1404,
    1441, 1544, 1576, 1823, 1851, 1977
\l_zrefclever_label_enclcnt_a-
    tl . . . . . 1050, 1189,
    1191, 1192, 1213, 1278, 1302, 1303
\l_zrefclever_label_enclcnt_b-
    tl . . . . . 1050, 1193,
    1195, 1196, 1215, 1255, 1304, 1305
\l_zrefclever_label_enclval_a-
    tl . . . . . 1050, 1197,
    1199, 1200, 1298, 1306, 1307, 1314
\l_zrefclever_label_enclval_b-
    tl . . . . . 1050, 1201,
    1203, 1204, 1300, 1308, 1309, 1316
\l_zrefclever_label_type_a_tl . .
    . . . . . 1050, 1064, 1066, 1070,
    1073, 1123, 1132, 1141, 1149, 1157,
    1349, 1378, 1469, 1473, 1506, 1514,
    1519, 1535, 1583, 1856, 2286, 2289,
    2293, 2298, 2304, 2327, 2330, 2334
\l_zrefclever_label_type_b_tl . .
    . . . . . 1050,
    1125, 1133, 1142, 1150, 1158, 1352,
    1381, 1470, 1478, 1507, 1515, 1519
\__zrefclever_label_type_put_-
    new_right:n . . . . . 31, 1062, 1084
\l_zrefclever_label_types_seq . .
    . . . . . 31, 1069, 1072, 1077, 1080, 1376
\__zrefclever_labels_in_sequence:nn
    . . . . . 1703, 1845, 2340
\g_zrefclever_language_aliases_-
    prop . . . . . 207, 214, 217, 221,
    224, 232, 234, 235, 247, 371, 386, 912
\l_zrefclever_last_of_type_bool
    . . . . . 38, 1397, 1490, 1495, 1496,
    1500, 1509, 1520, 1521, 1524, 1562
\l_zrefclever_lastsep_tl . 1420,
    1553, 1609, 1625, 1645, 1661, 1673
\l_zrefclever_link_star_bool . .
    . . . . . 1019, 1042, 1984, 2112, 2222
\l_zrefclever_listsep_tl . . . .
    . . . . 1419, 1552, 1621, 1658, 1901,
    1914, 1919, 1939, 1950, 1953, 1963
\l_zrefclever_load_dict_-
    verbose_bool . . . 241, 271, 280, 290
\g_zrefclever_loaded_dictionaries_-
    seq . . . . . 240, 251, 265
\l_zrefclever_main_language_tl .
    . 21, 708, 715, 721, 725, 728, 741, 777
\l_zrefclever_name_format_-
    fallback_tl . . . . .
    . . 1429, 2052, 2055, 2057, 2084, 2096
\l_zrefclever_name_format_tl . .
    . . . 1429, 2039, 2040, 2043, 2044,
    2052, 2053, 2061, 2067, 2079, 2090
\l_zrefclever_name_in_link_bool
    . . 1429, 1739, 2116, 2132, 2133, 2141
\l_zrefclever_namefont_tl 1412,
    1546, 1742, 1768, 2166, 2196, 2211
\l_zrefclever_nameinlink_str . .
    . . . . . 660, 665,
    667, 669, 671, 2114, 2120, 2122, 2126
\l_zrefclever_namesep_tl . . . .
    . . 1416, 1549, 2169, 2199, 2207, 2214
\l_zrefclever_next_is_same_bool
    . . . . . 39, 57, 1406,
    1835, 1865, 1881, 1887, 2360, 2386
\l_zrefclever_next_maybe_range_-
    bool . . . . .
    . . 39, 57, 1406, 1699, 1708, 1834,
    1861, 1872, 2352, 2359, 2378, 2385
\l_zrefclever_noabbrev_first_-
    bool . . . . . 694, 703, 2049
\l_zrefclever_noteseptl . . . .
    . . . . . 1030, 1424, 1450
\__zrefclever_page_format_aux: . .
    . . . . . 90, 94
\g_zrefclever_page_format_tl . .
    . . . . . 6, 89, 95, 98
\l_zrefclever_pairsep_tl . . . .
    . . . . . 1418, 1551, 1593, 1709
\__zrefclever_prop_put_non_-
    empty:Nnn . . . . . 14, 423, 440, 494
\__zrefclever_provide_dict_-
    default_transl:nn . . 305, 329, 346
\__zrefclever_provide_dict_type_-
    transl:nn . . . . . 297, 347, 364
\__zrefclever_provide_dictionary:n
    . . . . . 8, 29, 244, 285, 291, 767, 1020
\__zrefclever_provide_dictionary_-
    verbose:n . . . . . 287,
    294, 742, 750, 756, 778, 786, 792
\l_zrefclever_range_beg_label_-
    tl . . . . . 39, 1406, 1440,
    1622, 1641, 1646, 1656, 1659, 1671,
    1674, 1822, 1863, 1879, 1912, 1915,
    1937, 1940, 1948, 1951, 1961, 1964
\l_zrefclever_range_count_int . .
    . . . . . 39,
    1406, 1443, 1602, 1634, 1825, 1864,
    1876, 1880, 1886, 1894, 1931, 1972
\l_zrefclever_range_inhibit_-
    next_bool . . . . . 38, 39, 1406, 1840
\l_zrefclever_range_same_count_-
    int . . . . . 39,

```

[1406](#), [1444](#), [1589](#), [1623](#), [1634](#), [1826](#),  
[1866](#), [1882](#), [1888](#), [1917](#), [1931](#), [1973](#)  
\l\_zrefclever\_rangeseq\_tl .....  
..... [1417](#), [1550](#), [1676](#), [1710](#), [1966](#)  
\l\_zrefclever\_ref\_language\_tl ..  
..... [21](#),  
[707](#), [728](#), [740](#), [743](#), [748](#), [751](#), [755](#),  
[757](#), [767](#), [776](#), [779](#), [784](#), [787](#), [791](#),  
[793](#), [1020](#), [2065](#), [2088](#), [2094](#), [2303](#), [2309](#)  
\c\_zrefclever\_ref\_options\_-  
necessarily\_not\_type\_specific\_-  
seq ..... [166](#), [321](#), [870](#), [947](#)  
\c\_zrefclever\_ref\_options\_-  
necessarily\_type\_specific\_seq  
..... [166](#), [352](#), [991](#)  
\c\_zrefclever\_ref\_options\_not\_-  
type\_specific\_seq ..... [166](#), [841](#)  
\c\_zrefclever\_ref\_options\_-  
possibly\_type\_specific\_seq ..  
..... [166](#), [338](#), [968](#)  
\l\_zrefclever\_ref\_options\_prop .  
.... [24](#), [25](#), [839](#), [849](#), [850](#), [2281](#), [2323](#)  
\c\_zrefclever\_ref\_options\_type\_-  
specific\_seq ..... [166](#), [882](#)  
\l\_zrefclever\_ref\_property\_tl ..  
..... [16](#),  
[507](#), [512](#), [514](#), [520](#), [523](#), [539](#), [548](#),  
[1081](#), [1114](#), [1467](#), [1981](#), [2001](#), [2015](#),  
[2144](#), [2177](#), [2217](#), [2251](#), [2266](#), [2342](#)  
\l\_zrefclever\_ref\_typeset\_font\_-  
tl ..... [800](#), [802](#), [1453](#)  
\l\_zrefclever\_reffont\_in\_tl [1414](#),  
[1548](#), [1990](#), [2013](#), [2174](#), [2229](#), [2263](#)  
\l\_zrefclever\_reffont\_out\_tl ...  
..... [1413](#), [1547](#),  
[1987](#), [2010](#), [2171](#), [2190](#), [2226](#), [2260](#)  
\l\_zrefclever\_refpos\_in\_tl [1428](#),  
[1557](#), [2002](#), [2016](#), [2178](#), [2252](#), [2267](#)  
\l\_zrefclever\_refpos\_out\_tl [1426](#),  
[1555](#), [2005](#), [2018](#), [2191](#), [2255](#), [2269](#)  
\l\_zrefclever\_refpre\_in\_tl [1427](#),  
[1556](#), [2000](#), [2014](#), [2175](#), [2249](#), [2264](#)  
\l\_zrefclever\_refpre\_out\_tl [1425](#),  
[1554](#), [1988](#), [2011](#), [2172](#), [2227](#), [2261](#)  
\l\_zrefclever\_setup\_type\_tl ...  
..... [164](#), [262](#), [301](#), [316](#),  
[317](#), [328](#), [345](#), [359](#), [866](#), [894](#), [902](#),  
[915](#), [927](#), [928](#), [954](#), [975](#), [984](#), [998](#), [1006](#)  
\l\_zrefclever\_sort\_decided\_bool  
... [1060](#), [1206](#), [1210](#), [1229](#), [1240](#),  
[1257](#), [1263](#), [1280](#), [1286](#), [1312](#), [1324](#)  
\\_zrefclever\_sort\_default:nn ...  
..... [30–32](#), [1116](#), [1121](#)  
\\_zrefclever\_sort\_default\_-  
different\_types:nn . [18](#), [1165](#), [1332](#)  
\\_zrefclever\_sort\_default\_same\_-  
type:nn ..... [1161](#), [1187](#)  
\\_zrefclever\_sort\_labels: .....  
..... [31](#), [32](#), [38](#), [1028](#), [1078](#)  
\\_zrefclever\_sort\_page:nn .....  
..... [38](#), [1115](#), [1388](#)  
\l\_zrefclever\_sort\_prior\_a\_int .  
..... [1058](#),  
[1334](#), [1343](#), [1344](#), [1350](#), [1360](#), [1368](#)  
\l\_zrefclever\_sort\_prior\_b\_int .  
..... [1059](#),  
[1335](#), [1345](#), [1346](#), [1353](#), [1361](#), [1369](#)  
\l\_zrefclever\_tlastsep\_tl .....  
..... [1423](#), [1449](#), [1811](#)  
\l\_zrefclever\_tlistsep\_tl .....  
..... [1422](#), [1448](#), [1789](#)  
\l\_zrefclever\_tpairsep\_tl .....  
..... [1421](#), [1447](#), [1805](#)  
\l\_zrefclever\_type\_<type>-  
options\_prop ..... [25](#)  
\l\_zrefclever\_type\_count\_int ...  
..... [39](#), [1404](#), [1442](#), [1786](#),  
[1788](#), [1797](#), [1824](#), [2037](#), [2048](#), [2129](#)  
\l\_zrefclever\_type\_first\_label\_-  
tl ..... [1399](#), [1438](#), [1582](#), [1691](#),  
[1700](#), [1704](#), [1731](#), [1747](#), [1750](#), [1755](#),  
[1761](#), [1820](#), [1855](#), [2027](#), [2138](#), [2144](#),  
[2150](#), [2152](#), [2156](#), [2161](#), [2176](#), [2217](#),  
[2234](#), [2236](#), [2240](#), [2245](#), [2250](#), [2265](#)  
\l\_zrefclever\_type\_first\_label\_-  
type\_tl ..... [1399](#), [1439](#), [1583](#),  
[1695](#), [1821](#), [1856](#), [2030](#), [2060](#), [2066](#),  
[2072](#), [2078](#), [2083](#), [2089](#), [2095](#), [2101](#)  
\\_zrefclever\_type\_name\_setup: ..  
..... [40](#), [1719](#), [2025](#)  
\l\_zrefclever\_type\_name\_tl . [51](#),  
[1429](#), [1763](#), [1769](#), [2028](#), [2031](#), [2062](#),  
[2068](#), [2070](#), [2080](#), [2085](#), [2091](#), [2097](#),  
[2099](#), [2113](#), [2167](#), [2197](#), [2204](#), [2212](#)  
\l\_zrefclever\_typeset\_compress\_-  
bool ..... [603](#), [606](#), [1837](#)  
\l\_zrefclever\_typeset\_labels\_-  
seq ... [1399](#), [1435](#), [1459](#), [1460](#), [1465](#)  
\l\_zrefclever\_typeset\_last\_bool  
..... [38](#), [1397](#),  
[1456](#), [1457](#), [1463](#), [1489](#), [1794](#), [2128](#)  
\l\_zrefclever\_typeset\_name\_bool  
..... [554](#), [561](#), [566](#), [571](#), [1721](#), [1735](#)  
\l\_zrefclever\_typeset\_queue\_-  
curr\_tl ..... [1399](#), [1437](#), [1591](#),  
[1607](#), [1616](#), [1643](#), [1653](#), [1668](#), [1689](#),  
[1706](#), [1723](#), [1730](#), [1737](#), [1780](#), [1801](#),

|                                      |  |
|--------------------------------------|--|
| 1806, 1812, 1818, 1819, 1899, 1910,  | ..... 18, 588, 593, 594, 600, 1339             |
| 1935, 1946, 1959, 2042, 2123, 2127   |  |
| \l_zrefclever_typeset_queue_-        | \l_zrefclever_use_hyperref_bool                |
| prev_tl .... 1399, 1436, 1790, 1818  | ..... 619, 626,                                |
| \l_zrefclever_typeset_range_-        | 631, 636, 646, 652, 1984, 2111, 2221           |
| bool ..... 612, 615, 1027, 1687      | \l_zrefclever_warn_hyperref_-                  |
| \l_zrefclever_typeset_ref_bool .     | bool ..... 620, 627, 632, 637, 650             |
| ..... 553, 560, 565, 570, 1721, 1728 | \_zrefclever_zcref:nnn .. 1013, 1014           |
| \_zrefclever_typeset_refs: ....      | \_zrefclever_zcref:nnnn 29, 31, 1014           |
| ..... 38, 39, 50, 51, 53, 1029, 1433 | \l_zrefclever_zcref_labels_seq .               |
| \_zrefclever_typeset_refs_aux_-      | 31, 1018, 1038, 1042, 1083, 1086, 1435         |
| last_of_type: ..... 1565, 1573       | \l_zrefclever_zcref_note_tl ...                |
| \_zrefclever_typeset_refs_aux_-      | ..... 803, 806, 1031                           |
| not_last_of_type: .... 1569, 1829    | \l_zrefclever_zcref_with_check_-               |
| \l_zrefclever_typeset_sort_bool      | bool ..... 810, 825, 1023, 1034                |
| ..... 579, 582, 1026                 | \l_zrefclever_zrefcheck_-                      |
| \l_zrefclever_typesort_seq ....      | available_bool ..... 809, 820, 831, 1022, 1033 |