

# The zref-clever package implementation\*

Gustavo Barros<sup>†</sup>

2021-09-29

## Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>2</b>
<b>3</b>	<b>zref setup</b>	<b>3</b>
<b>4</b>	<b>Plumbing</b>	<b>7</b>
4.1	Messages . . . . .	7
4.2	Reference format . . . . .	8
4.3	Languages . . . . .	10
4.4	Dictionaries . . . . .	11
4.5	Options . . . . .	17
<b>5</b>	<b>Configuration</b>	<b>29</b>
5.1	\zcsetup . . . . .	29
5.2	\zcRefTypeSetup . . . . .	29
5.3	\zcLanguageSetup . . . . .	31
<b>6</b>	<b>User interface</b>	<b>33</b>
6.1	\zceref . . . . .	33
6.2	\zcpageref . . . . .	35
<b>7</b>	<b>Sorting</b>	<b>35</b>
<b>8</b>	<b>Typesetting</b>	<b>43</b>
<b>9</b>	<b>Compatibility</b>	<b>68</b>
9.1	Appendix . . . . .	68
9.2	enumitem package . . . . .	69

---

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

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

<b>10</b>	<b>Dictionaries</b>	<b>70</b>
10.1	English	70
10.2	German	73
10.3	French	77
10.4	Portuguese	80
10.5	Spanish	83
<b>Index</b>		<b>87</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

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

Identify the package.

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

## 2 Dependencies

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

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

### 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` property is provided by `zref-base`, while `zref-abspage` provides the `abspage` property which gives us a safe and easy way to sort labels for page references.

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

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

The reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the<counter>` and store it “clean” in `zc@thecnt` for reserved use. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@...` prefix.

```
22 \zref@newprop { zc@thecnt }
23 { \use:c { the \l__zrefclever_current_counter_tl } }
24 \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`.

```
25 \zref@newprop { zc@type }
26 {
27   \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
28     \l__zrefclever_current_counter_tl
29     {
30       \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
31         { \l__zrefclever_current_counter_tl }
32     }
33   { \l__zrefclever_current_counter_tl }
34 }
35 \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’).

```
36 \zref@newprop { zc@cntval } [0]
37 { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
```

```

38 \zref@addprop \ZREF@mainlist { zc@cntval }
39 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
40 \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 ‘ltxcounts.dtx’ in ‘source2e’). Besides, there may be a chain of resetting counters, which must be taken into account: if ‘counterC’ gets reset by ‘counterB’, and ‘counterB’ gets reset by ‘counterA’, stepping the latter affects all three of them.

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

has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

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

Recursively generate a *sequence* of “enclosing counters” and values, for a given  $\langle counter \rangle$  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 {\langle counter \rangle}
    \__zrefclever_get_enclosing_counters_value:n {\langle counter \rangle}

41 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
42 {
43   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
44   {
45     { \__zrefclever_counter_reset_by:n {#1} }
46     \__zrefclever_get_enclosing_counters:e
47     { \__zrefclever_counter_reset_by:n {#1} }
48   }
49 }
50 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
51 {
52   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
53   {
54     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
55     \__zrefclever_get_enclosing_counters_value:e
56     { \__zrefclever_counter_reset_by:n {#1} }
57   }
58 }

```

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’).

```

59 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { e }
60 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

```

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

`\__zrefclever_counter_reset_by:n`

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

```

    \__zrefclever_counter_reset_by:n {\langle counter \rangle}

61 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
62 {
63   \bool_if:nTF
64   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }

```

```

65     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
66     {
67       \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
68       { \__zrefclever_counter_reset_by_aux:nn {#1} }
69     }
70   }
71   \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
72   {
73     \cs_if_exist:cT { c@ #2 }
74     {
75       \tl_if_empty:cF { cl@ #2 }
76       {
77         \tl_map_tokens:cn { cl@ #2 }
78         { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
79       }
80     }
81   }
82   \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
83   {
84     \str_if_eq:nnT {#2} {#3}
85     { \tl_map_break:n { \seq_map_break:n {#1} } }
86   }

```

(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.

```

87   \zref@newprop { zc@enclcnt }
88   { \__zrefclever_get_enclosing_counters:e \l__zrefclever_current_counter_tl }
89   \zref@newprop { zc@enclval }
90   { \__zrefclever_get_enclosing_counters_value:e \l__zrefclever_current_counter_tl }
91   \zref@addprop \ZREF@mainlist { zc@enclcnt }
92   \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}`.

```

93   \tl_new:N \g__zrefclever_page_format_tl

```

```

94 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
95 \AddToHook { shipout / before }
96 {
97   \group_begin:
98   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
99   \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
100   \group_end:
101 }
102 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
103 \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

```

104 \msg_new:nnn { zref-clever } { option-not-type-specific }
105 {
106   Option~'#1'~is-not-type-specific~\msg_line_context:..~
107   Set-it-in~'\iow_char:N\zcLanguageSetup'~before~first~'type'
108   ~switch-or-as~package~option.
109 }
110 \msg_new:nnn { zref-clever } { option-only-type-specific }
111 {
112   No-type-specified-for-option~'#1'~\msg_line_context:..~
113   Set-it-after~'type'~switch-or-in~'\iow_char:N\zcRefTypeSetup'.
114 }
115 \msg_new:nnn { zref-clever } { key-requires-value }
116 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:. }
117 \msg_new:nnn { zref-clever } { language-declared }
118 { Language~'#1'~is-already-declared.~Nothing-to-do. }
119 \msg_new:nnn { zref-clever } { unknown-language-alias }
120 {
121   Language~'#1'~is-unknown,~cannot~alias~to~it.~See~documentation~for~
122   '\iow_char:N\zcDeclareLanguage'~and~
123   '\iow_char:N\zcDeclareLanguageAlias'.
124 }
125 \msg_new:nnn { zref-clever } { unknown-language-transl }
126 {
127   Language~'#1'~is-unknown,~cannot~declare~translations~to~it.~
128   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
129   '\iow_char:N\zcDeclareLanguageAlias'.
130 }
131 \msg_new:nnn { zref-clever } { unknown-language-opt }
132 {
133   Language~'#1'~is-unknown~\msg_line_context:..~Using~default.~
134   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
135   '\iow_char:N\zcDeclareLanguageAlias'.
136 }
137 \msg_new:nnn { zref-clever } { dict-loaded }

```

```

138 { Loaded~'#1'~dictionary. }
139 \msg_new:nnn { zref-clever } { dict-not-available }
140 { Dictionary~for~'#1'~not~available~\msg_line_context:. }
141 \msg_new:nnn { zref-clever } { unknown-language-load }
142 {
143   Language~'#1'~is~unknown~\msg_line_context:.~Unable~to~load~dictionary.~
144   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
145   '\iow_char:N\zcDeclareLanguageAlias'.
146 }
147 \msg_new:nnn { zref-clever } { missing-zref-titleref }
148 {
149   Option~'ref=title'~requested~\msg_line_context:.~
150   But~package~'zref-titleref'~is~not~loaded,~falling-back~to~default~'ref'.
151 }
152 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
153 {
154   Option~'hyperref'~only~available~in~the~preamble.~
155   Use~the~starred~version~of~'\iow_char:N\zcRef'~instead.
156 }
157 \msg_new:nnn { zref-clever } { missing-hyperref }
158 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
159 \msg_new:nnn { zref-check } { check-document-only }
160 { Option~'check'~only~available~in~the~document. }
161 \msg_new:nnn { zref-clever } { missing-zref-check }
162 {
163   Option~'check'~requested~\msg_line_context:.~
164   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
165 }
166 \msg_new:nnn { zref-clever } { counters-not-nested }
167 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:. }
168 \msg_new:nnn { zref-clever } { missing-type }
169 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
170 \msg_new:nnn { zref-clever } { missing-name }
171 { Name~undefined~for~type~'#1'~\msg_line_context:. }
172 \msg_new:nnn { zref-clever } { missing-string }
173 {
174   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:.~
175   But~we~should~have:~throw~a~rock~at~the~maintainer.
176 }
177 \msg_new:nnn { zref-clever } { single-element-range }
178 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }

```

## 4.2 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, these precedence rules are handled / enforced in `\__zrefclever_get_ref_string:nN`, `\__zrefclever_get_ref_font:nN`, and `\__zrefclever_type_name_setup:` which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in `\g_zrefclever_fallback_dict_prop`.

`\l_zrefclever_setup_type_tl` Store “current” type and language in different places for option and translation handling, notably in `\__zrefclever_provide_dictionary:n`, `\zcRefTypeSetup`, and



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

```
179 \tl_new:N \l__zrefclever_setup_type_tl
180 \tl_new:N \l__zrefclever_dict_language_tl
```

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

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.

```
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_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq
181 \seq_const_from_clist:Nn
182 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
183 {
184     tpairsep ,
185     tlistsep ,
186     tlastsep ,
187     notesep ,
188 }
189 \seq_const_from_clist:Nn
190 \c__zrefclever_ref_options_possibly_type_specific_seq
191 {
192     namesep ,
193     pairsep ,
194     listsep ,
195     lastsep ,
196     rangesep ,
197     refpre ,
198     refpos ,
199     refpre-in ,
200     refpos-in ,
201 }
```

Only “type names” are “necessarily type-specific”, which makes them somewhat special on the retrieval side of things. In short, they don’t have their values queried by \\_\_zrefclever\_get\_ref\_string:nN, but by \\_\_zrefclever\_type\_name\_setup:.

```
202 \seq_const_from_clist:Nn
203 \c__zrefclever_ref_options_necessarily_type_specific_seq
204 {
205     Name-sg ,
206     name-sg ,
207     Name-pl ,
208     name-pl ,
209     Name-sg-ab ,
210     name-sg-ab ,
211     Name-pl-ab ,
212     name-pl-ab ,
213 }
```

\c\_\_zrefclever\_ref\_options\_font\_seq are technically “possibly type-specific”, but are not “language-specific”, so we separate them.

```
214 \seq_const_from_clist:Nn
215 \c__zrefclever_ref_options_font_seq
216 {
217     namefont ,
```

```

218     reffont ,
219     reffont-in ,
220 }
221 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
222 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
223   \c__zrefclever_ref_options_possibly_type_specific_seq
224   \c__zrefclever_ref_options_necessarily_type_specific_seq
225 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
226   \c__zrefclever_ref_options_typesetup_seq
227   \c__zrefclever_ref_options_font_seq
228 \seq_new:N \c__zrefclever_ref_options_reference_seq
229 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
230   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
231   \c__zrefclever_ref_options_possibly_type_specific_seq
232 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
233   \c__zrefclever_ref_options_reference_seq
234   \c__zrefclever_ref_options_font_seq

```

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

### 4.3 Languages

`\g__zrefclever_languages_prop` Stores the names of known languages and the mapping from “language name” to “dictionary name”. Whether or not a language or alias is known to `zref-clever` is decided by its presence in this property list. A “base language” (loose concept here, meaning just “the name we gave for the dictionary in that particular language”) is just like any other one, the only difference is that the “language name” happens to be the same as the “dictionary name”, in other words, it is an “alias to itself”.

```

235 \prop_new:N \g__zrefclever_languages_prop

```

(End definition for `\g__zrefclever_languages_prop`.)

`\zcDeclareLanguage` Declare a new language for use with `zref-clever`.  $\langle language \rangle$  is taken to be both the “language name” and the “dictionary name”. If  $\langle language \rangle$  is already known, just warn. `\zcDeclareLanguage` is preamble only.

```

\zcDeclareLanguage { $\langle language \rangle$ }

236 \NewDocumentCommand \zcDeclareLanguage { m }
237 {
238   \tl_if_empty:nF {#1}
239   {
240     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
241       { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
242       { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
243   }
244 }
245 \@onlypreamble \zcDeclareLanguage

```

(End definition for `\zcDeclareLanguage`.)

`\zcDeclareLanguageAlias` Declare  $\langle language alias \rangle$  to be an alias of  $\langle aliased language \rangle$ .  $\langle aliased language \rangle$  must be already known to `zref-clever`, as stored in `\g__zrefclever_languages_prop`. `\zcDeclareLanguageAlias` is preamble only.

```

\zcDeclareLanguageAlias {\language alias} {\aliased language}

246 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
247 {
248   \tl_if_empty:nF {#1}
249   {
250     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
251     {
252       \exp_args:NnNx
253       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
254       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
255     }
256     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
257   }
258 }
259 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for `\zcDeclareLanguageAlias`.)

## 4.4 Dictionaries

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

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

This design decision has also implications to the *form* the dictionary files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.ltx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load dictionaries on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot

afford to do it. Hence, zref-clever’s built-in dictionary files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/dictionary}` by `\__zrefclever_provide_dictionary:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The dictionary file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`\__zrefclever_provide_dictionary:n` is only meant to load the built-in dictionaries. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a variable `\g__zrefclever_dict_⟨language⟩_prop`, created as needed. Hence, there is no need to “load” anything in this case: definitions and assignments made by the user are performed immediately.

## Provide

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

260 `\seq_new:N \g__zrefclever_loaded_dictionaries_seq`

(End definition for `\g__zrefclever_loaded_dictionaries_seq`.)

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

261 `\bool_new:N \l__zrefclever_load_dict_verbose_bool`

(End definition for `\l__zrefclever_load_dict_verbose_bool`.)

`\__zrefclever_provide_dictionary:n` Load dictionary for known `⟨language⟩` if it is available and if it has not already been loaded.

```

\__zrefclever_provide_dictionary:n {⟨language⟩}

262 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
263 {
264   \group_begin:
265   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
266   \l__zrefclever_dict_language_tl
267   {
268     \seq_if_in:NVF
269     \g__zrefclever_loaded_dictionaries_seq
270     \l__zrefclever_dict_language_tl
271     {
272       \exp_args:Nx \file_get:nnNTF
273       { zref-clever- \l__zrefclever_dict_language_tl .dict }
274       { \ExplSyntaxOn }
275       \l_tmpa_tl
276       {
277         \prop_if_exist:cF
278         {
279           g__zrefclever_dict_
280           \l__zrefclever_dict_language_tl _prop
281         }
282         {
283           \prop_new:c
284           {
285             g__zrefclever_dict_
286             \l__zrefclever_dict_language_tl _prop

```

```

287         }
288     }
289     \tl_clear:N \l__zrefclever_setup_type_tl
290     \exp_args:NnV
291         \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
292     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
293         \l__zrefclever_dict_language_tl
294     \msg_note:nnx { zref-clever } { dict-loaded }
295         { \l__zrefclever_dict_language_tl }
296 }
297 {
298     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
299     {
300         \msg_warning:nnx { zref-clever } { dict-not-available }
301         { \l__zrefclever_dict_language_tl }
302     }

```

Even if we don't have the actual dictionary, we register it as “loaded”. At this point, it is a known language, properly declared. There is no point in trying to load it multiple times, because users cannot really provide the dictionary files (well, technically they could, but we are working so they don't need to, and have better ways to do what they want). And if the users had provided some translations themselves, by means of `\zcLanguageSetup`, everything would be in place, and they could use the `lang` option multiple times, and the `dict-not-available` warning would never go away.

```

303         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
304         \l__zrefclever_dict_language_tl
305     }
306 }
307 }
308 {
309     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
310     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
311 }
312 \group_end:
313 }
314 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }

```

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

`\__zrefclever_provide_dictionary_verbose:n` Does the same as `\__zrefclever_provide_dictionary:n`, but warns if the loading of the dictionary has failed.

```

    \__zrefclever_provide_dictionary_verbose:n {<language>}

315 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
316 {
317     \group_begin:
318     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
319     \__zrefclever_provide_dictionary:n {#1}
320     \group_end:
321 }
322 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

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

\\_zrefclever\_provide\_dict\_type\_transl:nn  
\\_zrefclever\_provide\_dict\_default\_transl:nn

A couple of auxiliary functions for the of zref-clever/dictionary keys set in \\_zrefclever\_provide\_dictionary:n. They respectively “provide” (i.e. set if it value does not exist, do nothing if it already does) “type-specific” and “default” translations. Both receive  $\langle key \rangle$  and  $\langle translation \rangle$  as arguments, but \\_zrefclever\_provide\_dict\_type\_transl:nn relies on the current value of \l\\_zrefclever\_setup\_type\_tl, as set by the type key.

```

\__zrefclever_provide_dict_type_transl:nn {\langle key \rangle} {\langle translation \rangle}
\__zrefclever_provide_dict_default_transl:nn {\langle key \rangle} {\langle translation \rangle}

323 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
324 {
325   \exp_args:Nnx \prop_gput_if_new:cnn
326   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
327   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
328 }
329 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
330 {
331   \prop_gput_if_new:cnn
332   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
333   { default- #1 } {#2}
334 }
```

(End definition for \\_zrefclever\_provide\_dict\_type\_transl:nn and \\_zrefclever\_provide\_dict\_default\_transl:nn.)

The set of keys for zref-clever/dictionary, which is used to process the dictionary files in \\_zrefclever\_provide\_dictionary:n. The no-op cases for each category have their messages sent to “info”. These messages should not occur, as long as the dictionaries are well formed, but they’re placed there nevertheless, and can be leveraged in regression tests.

```

335 \keys_define:nn { zref-clever / dictionary }
336 {
337   type .code:n =
338   {
339     \tl_if_empty:NTF {#1}
340     { \tl_clear:N \l__zrefclever_setup_type_tl }
341     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
342   } ,
343 }
344 \seq_map_inline:Nn
345 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
346 {
347   \keys_define:nn { zref-clever / dictionary }
348   {
349     #1 .value_required:n = true ,
350     #1 .code:n =
351     {
352       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
353       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
354       {
355         \msg_info:nnn { zref-clever }
356         { option-not-type-specific } {#1}
357       }
358     } ,

```

```

359     }
360   }
361   \seq_map_inline:Nn
362     \c__zrefclever_ref_options_possibly_type_specific_seq
363   {
364     \keys_define:nn { zref-clever / dictionary }
365     {
366       #1 .value_required:n = true ,
367       #1 .code:n =
368       {
369         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
370         { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
371         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
372       } ,
373     }
374   }
375   \seq_map_inline:Nn
376     \c__zrefclever_ref_options_necessarily_type_specific_seq
377   {
378     \keys_define:nn { zref-clever / dictionary }
379     {
380       #1 .value_required:n = true ,
381       #1 .code:n =
382       {
383         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
384         {
385           \msg_info:nnn { zref-clever }
386             { option-only-type-specific } {#1}
387         }
388         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
389       } ,
390     }
391   }

```

## Fallback

All “strings” queried with `\__zrefclever_get_ref_string:nN` – in practice, those in either `\c__zrefclever_ref_options_necessarily_not_type_specific_seq` or `\c__zrefclever_ref_options_possibly_type_specific_seq` – must have their values set for “fallback”, even if to empty ones, since this is what will be retrieved in the absence of a proper translation, which will be the case if `babel` or `polyglossia` is loaded and sets a language which `zref-clever` does not know. On the other hand, “type names” are not looked for in “fallback”, since it is indeed impossible to provide any reasonable value for them for a “specified but unknown language”. Also “font” options – those in `\c__zrefclever_ref_options_font_seq`, and queried with `\__zrefclever_get_ref_font:nN` – do not need to be provided here, since the later function sets an empty value if the option is not found.

TODO Add regression test to ensure all fallback “translations” are indeed present.

```

392   \prop_new:N \g__zrefclever_fallback_dict_prop
393   \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
394   {
395     tpairsep = {,-} ,
396     tlistsep = {,-} ,

```

```

397     tlastsep = {,~} ,
398     notesep  = {~} ,
399     namesep  = {\nobreakspace} ,
400     pairsep  = {,~} ,
401     listsep  = {,~} ,
402     lastsep  = {,~} ,
403     rangesep = {\textendash} ,
404     refpre   = {} ,
405     refpos   = {} ,
406     refpre-in = {} ,
407     refpos-in = {} ,
408 }

```

### Get translations

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

```

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

409 \prg_new_protected_conditional:Npnn
410 \_zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
411 {
412   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
413   \l__zrefclever_dict_language_tl
414   {
415     \prop_get:cnNTF
416     { \g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
417     { type- #2 - #3 } #4
418     { \prg_return_true: }
419     { \prg_return_false: }
420   }
421   { \prg_return_false: }
422 }
423 \prg_generate_conditional_variant:Nnn
424 \_zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }

```

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

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

```

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

425 \prg_new_protected_conditional:Npnn
426 \_zrefclever_get_default_transl:nnN #1#2#3 { F }
427 {
428   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
429   \l__zrefclever_dict_language_tl
430   {
431     \prop_get:cnNTF

```



```

432         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
433         { default- #2 } #3
434         { \prg_return_true: }
435         { \prg_return_false: }
436     }
437     { \prg_return_false: }
438 }
439 \prg_generate_conditional_variant:Nnn
440   \__zrefclever_get_default_transl:nnN { xnN } { F }

```

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

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

```

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

441 % {<key>}<tl var to set>
442 \prg_new_protected_conditional:Npnn
443   \__zrefclever_get_fallback_transl:nN #1#2 { F }
444   {
445     \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
446     { #1 } #2
447     { \prg_return_true: }
448     { \prg_return_false: }
449   }

```

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

## 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>}

450 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
451   {
452     \tl_if_empty:nTF {#3}
453       { \prop_remove:Nn #1 {#2} }
454       { \prop_put:Nnn #1 {#2} {#3} }
455   }

```

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

### ref option

\l\_\_zrefclever\_ref\_property\_tl stores the property to which the reference is being made. Currently, we restrict `ref=` to these 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.

```

456 \tl_new:N \l__zrefclever_ref_property_tl
457 \keys_define:nn { zref-clever / reference }
458 {
459   ref .choice: ,
460   ref / zc@thecnt .code:n =
461     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
462   ref / page .code:n =
463     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
464   ref / title .code:n =
465     {
466       \AddToHook { begindocument }
467       {
468         \ifpackageloaded { zref-titleref }
469         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
470         {
471           \msg_warning:nn { zref-clever } { missing-zref-titleref }
472           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
473         }
474       }
475     } ,
476   ref .initial:n = zc@thecnt ,
477   ref .default:n = zc@thecnt ,
478   page .meta:n = { ref = page },
479   page .value_forbidden:n = true ,
480 }
481 \AddToHook { begindocument }
482 {
483   \ifpackageloaded { zref-titleref }
484   {
485     \keys_define:nn { zref-clever / reference }
486     {
487       ref / title .code:n =
488       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
489     }
490   }
491   {
492     \keys_define:nn { zref-clever / reference }
493     {
494       ref / title .code:n =
495       {
496         \msg_warning:nn { zref-clever } { missing-zref-titleref }
497         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
498       }
499     }
500   }
501 }
```

### typeset option

```
502 \bool_new:N \l__zrefclever_typeset_ref_bool
503 \bool_new:N \l__zrefclever_typeset_name_bool
504 \keys_define:nn { zref-clever / reference }
505 {
506   typeset .choice: ,
507   typeset / both .code:n =
508   {
509     \bool_set_true:N \l__zrefclever_typeset_ref_bool
510     \bool_set_true:N \l__zrefclever_typeset_name_bool
511   } ,
512   typeset / ref .code:n =
513   {
514     \bool_set_true:N \l__zrefclever_typeset_ref_bool
515     \bool_set_false:N \l__zrefclever_typeset_name_bool
516   } ,
517   typeset / name .code:n =
518   {
519     \bool_set_false:N \l__zrefclever_typeset_ref_bool
520     \bool_set_true:N \l__zrefclever_typeset_name_bool
521   } ,
522   typeset .initial:n = both ,
523   typeset .value_required:n = true ,
524
525   noname .meta:n = { typeset = ref } ,
526   noname .value_forbidden:n = true ,
527 }
```

### sort option

```
528 \bool_new:N \l__zrefclever_typeset_sort_bool
529 \keys_define:nn { zref-clever / reference }
530 {
531   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
532   sort .initial:n = true ,
533   sort .default:n = true ,
534   nosort .meta:n = { sort = false } ,
535   nosort .value_forbidden:n = true ,
536 }
```

### 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`.

```
537 \seq_new:N \l__zrefclever_typesort_seq
538 \keys_define:nn { zref-clever / reference }
539 {
540   typesort .code:n =
541   {
542     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
543     \seq_reverse:N \l__zrefclever_typesort_seq
544   } ,
545   typesort .initial:n =
```

```

546     { part , chapter , section , paragraph },
547     typesort .value_required:n = true ,
548     notypesort .code:n =
549     { \seq_clear:N \l__zrefclever_typesort_seq } ,
550     notypesort .value_forbidden:n = true ,
551 }

```

#### comp option

```

552 \bool_new:N \l__zrefclever_typeset_compress_bool
553 \keys_define:nn { zref-clever / reference }
554 {
555     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
556     comp .initial:n = true ,
557     comp .default:n = true ,
558     nocomp .meta:n = { comp = false },
559     nocomp .value_forbidden:n = true ,
560 }

```

#### range option

```

561 \bool_new:N \l__zrefclever_typeset_range_bool
562 \keys_define:nn { zref-clever / reference }
563 {
564     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
565     range .initial:n = false ,
566     range .default:n = true ,
567 }

```

#### cap and capfirst options

```

568 \bool_new:N \l__zrefclever_capitalize_bool
569 \bool_new:N \l__zrefclever_capitalize_first_bool
570 \keys_define:nn { zref-clever / reference }
571 {
572     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
573     cap .initial:n = false ,
574     cap .default:n = true ,
575     nocap .meta:n = { cap = false },
576     nocap .value_forbidden:n = true ,
577
578     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
579     capfirst .initial:n = false ,
580     capfirst .default:n = true ,
581 }

```

#### abbrev and noabbrevfirst options

```

582 \bool_new:N \l__zrefclever_abbrev_bool
583 \bool_new:N \l__zrefclever_noabbrev_first_bool
584 \keys_define:nn { zref-clever / reference }
585 {
586     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
587     abbrev .initial:n = false ,
588     abbrev .default:n = true ,
589     noabbrev .meta:n = { abbrev = false },
590     noabbrev .value_forbidden:n = true ,

```

```

591
592     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
593     noabbrevfirst .initial:n = false ,
594     noabbrevfirst .default:n = true ,
595 }

```

## S option

```

596 \keys_define:nn { zref-clever / reference }
597 {
598     S .meta:n =
599     { capfirst = true , noabbrevfirst = true },
600     S .value_forbidden:n = true ,
601 }

```

## hyperref option

```

602 \bool_new:N \l__zrefclever_use_hyperref_bool
603 \bool_new:N \l__zrefclever_warn_hyperref_bool
604 \keys_define:nn { zref-clever / reference }
605 {
606     hyperref .choice: ,
607     hyperref / auto .code:n =
608     {
609         \bool_set_true:N \l__zrefclever_use_hyperref_bool
610         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
611     } ,
612     hyperref / true .code:n =
613     {
614         \bool_set_true:N \l__zrefclever_use_hyperref_bool
615         \bool_set_true:N \l__zrefclever_warn_hyperref_bool
616     } ,
617     hyperref / false .code:n =
618     {
619         \bool_set_false:N \l__zrefclever_use_hyperref_bool
620         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
621     } ,
622     hyperref .initial:n = auto ,
623     hyperref .default:n = auto
624 }
625 \AddToHook { begindocument }
626 {
627     \@ifpackageloaded { hyperref }
628     {
629         \bool_if:NT \l__zrefclever_use_hyperref_bool
630         { \RequirePackage { zref-hyperref } }
631     }
632     {
633         \bool_if:NT \l__zrefclever_warn_hyperref_bool
634         { \msg_warning:nn { zref-clever } { missing-hyperref } }
635         \bool_set_false:N \l__zrefclever_use_hyperref_bool
636     }
637     \keys_define:nn { zref-clever / reference }
638     {
639         hyperref .code:n =
640         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }

```

```

641     }
642 }

```

### nameinlink option

```

643 \str_new:N \l__zrefclever_nameinlink_str
644 \keys_define:nn { zref-clever / reference }
645 {
646   nameinlink .choice: ,
647   nameinlink / true .code:n =
648     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
649   nameinlink / false .code:n =
650     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
651   nameinlink / single .code:n =
652     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
653   nameinlink / tsingle .code:n =
654     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
655   nameinlink .initial:n = tsingle ,
656   nameinlink .default:n = true ,
657 }

```

### lang option

`\l__zrefclever_current_language_tl` is an internal alias for `babel`’s `\language` or `polyglossia`’s `\mainbabelname` and, if none of them is loaded, we set it to `english`. `\l__zrefclever_main_language_tl` is an internal alias for `babel`’s `\bbl@main@language` or for `polyglossia`’s `\mainbabelname`, as the case may be. Note that for `polyglossia` we get `babel`’s language names, so that we only need to handle those internally. `\l__zrefclever_ref_language_tl` is the internal variable which stores the language in which the reference is to be made.

The overall setup here seems a little roundabout, but this is actually required. In the preamble, we (potentially) don’t yet have values for the “main” and “current” document languages, this must be retrieved at a `begindocument` hook. The `begindocument` hook is responsible to get values for `\l__zrefclever_main_language_tl` and `\l__zrefclever_current_language_tl`, and to set the default for `\l__zrefclever_ref_language_tl`. Package options, or preamble calls to `\zcsetup` are also hooked at `begindocument`, but come after the first hook, so that the pertinent variables have been set when they are executed. Finally, we set a third `begindocument` hook, at `begindocument/before`, so that it runs after any options set in the preamble. This hook redefines the `lang` option for immediate execution in the document body, and ensures the `main` language’s dictionary gets loaded, if it hadn’t been already.

For the `babel` and `polyglossia` variables which store the “main” and “current” languages, see <https://tex.stackexchange.com/a/233178>, including comments, particularly the one by Javier Bezos. For the `babel` and `polyglossia` variables which store the list of loaded languages, see <https://tex.stackexchange.com/a/281220>, including comments, particularly PLK’s. Note, however, that languages loaded by `\babelprovide`, either directly, “on the fly”, or with the `provide` option, do not get included in `\bbl@loaded`.

```

658 \tl_new:N \l__zrefclever_ref_language_tl
659 \tl_new:N \l__zrefclever_main_language_tl
660 \tl_new:N \l__zrefclever_current_language_tl
661 \AddToHook { begindocument }
662 {

```

```

663 \@@ifpackageloaded { babel }
664 {
665   \tl_set:Nn \l__zrefclever_current_language_tl { \language }
666   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
667 }
668 {
669   \@@ifpackageloaded { polyglossia }
670   {
671     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
672     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
673   }
674   {
675     \tl_set:Nn \l__zrefclever_current_language_tl { english }
676     \tl_set:Nn \l__zrefclever_main_language_tl { english }
677   }
678 }

```

Provide default value for `\l__zrefclever_ref_language_tl` corresponding to option `main`, but do so outside of the `l3keys` machinery (that is, instead of using `.initial:n`), so that we are able to distinguish when the user actually gave the option, in which case the dictionary loading is done verbosely, from when we are setting the default value (here), in which case the dictionary loading is done silently.

```

679   \tl_set:Nn \l__zrefclever_ref_language_tl
680   { \l__zrefclever_main_language_tl }
681 }
682 \keys_define:nn { zref-clever / reference }
683 {
684   lang .code:n =
685   {
686     \AddToHook { begindocument }
687     {
688       \str_case:nnF {#1}
689       {
690         { main }
691         {
692           \tl_set:Nn \l__zrefclever_ref_language_tl
693           { \l__zrefclever_main_language_tl }
694           \__zrefclever_provide_dictionary_verbosely:x
695           { \l__zrefclever_ref_language_tl }
696         }
697
698         { current }
699         {
700           \tl_set:Nn \l__zrefclever_ref_language_tl
701           { \l__zrefclever_current_language_tl }
702           \__zrefclever_provide_dictionary_verbosely:x
703           { \l__zrefclever_ref_language_tl }
704         }
705       }
706     }
707     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
708     {
709       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
710     }

```

```

711         {
712             \msg_warning:nnn { zref-clever }
713             { unknown-language-opt } {#1}
714             \tl_set:Nn \l__zrefclever_ref_language_tl
715             { \l__zrefclever_main_language_tl }
716         }
717         \__zrefclever_provide_dictionary_verbose:x
718         { \l__zrefclever_ref_language_tl }
719     }
720 }
721 },
722 lang .value_required:n = true ,
723 }
724 \AddToHook { begindocument / before }
725 {
726     \AddToHook { begindocument }
727     {
728         \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
729     }
730 }
731 lang .code:n =
732 {
733     \str_case:nnF {#1}
734     {
735         { main }
736         {
737             \tl_set:Nn \l__zrefclever_ref_language_tl
738             { \l__zrefclever_main_language_tl }
739             \__zrefclever_provide_dictionary:x
740             { \l__zrefclever_ref_language_tl }
741         }
742     }
743     { current }
744     {
745         \tl_set:Nn \l__zrefclever_ref_language_tl
746         { \l__zrefclever_current_language_tl }
747         \__zrefclever_provide_dictionary:x
748         { \l__zrefclever_ref_language_tl }
749     }
750 }
751 {
752     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
753     {
754         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
755     }
756     {

```

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.

Redefinition of the `lang` key option for the document body. Also, drop the verbose dictionary loading in the document body, as it can become intrusive depending on the use case, and does not provide much “juice” anyway: in `\zcref` missing names warnings will already ensue.



```

757         \msg_warning:nnn { zref-clever }
758         { unknown-language-opt } {#1}
759         \tl_set:Nn \l__zrefclever_ref_language_tl
760         { \l__zrefclever_main_language_tl }
761     }
762     \__zrefclever_provide_dictionary:x
763     { \l__zrefclever_ref_language_tl }
764 }
765 },
766 lang .value_required:n = true ,
767 }
768 }
769 }

```

### font option

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

```

770 \tl_new:N \l__zrefclever_ref_typeset_font_tl
771 \keys_define:nn { zref-clever / reference }
772 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

### note option

```

773 \tl_new:N \l__zrefclever_zcref_note_tl
774 \keys_define:nn { zref-clever / reference }
775 {
776     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
777     note .value_required:n = true ,
778 }

```

### check option

Integration with `zref-check`.

```

779 \bool_new:N \l__zrefclever_zrefcheck_available_bool
780 \bool_new:N \l__zrefclever_zcref_with_check_bool
781 \keys_define:nn { zref-clever / reference }
782 {
783     check .code:n =
784     { \msg_warning:nn { zref-clever } { check-document-only } } ,
785 }
786 \AddToHook { begindocument }
787 {
788     \@ifpackageloaded { zref-check }
789     {
790         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
791         \keys_define:nn { zref-clever / reference }
792         {
793             check .code:n =
794             {
795                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
796                 \keys_set:nn { zref-check / zcheck } {#1}
797             }

```

```

798     }
799   }
800   {
801     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
802     \keys_define:nn { zref-clever / reference }
803     {
804       check .code:n =
805         { \msg_warning:nn { zref-clever } { missing-zref-check } }
806     }
807   }
808 }

```

### 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`.

```

809 \prop_new:N \l__zrefclever_counter_type_prop
810 \keys_define:nn { zref-clever / label }
811 {
812   countertype .code:n =
813   {
814     \keyval_parse:nnn
815     {
816       \msg_warning:nnnn { zref-clever }
817       { key-requires-value } { countertype }
818     }
819     {
820       \__zrefclever_prop_put_non_empty:Nnn
821       \l__zrefclever_counter_type_prop
822     }
823     {#1}
824   } ,
825   countertype .value_required:n = true ,
826   countertype .initial:n =
827   {
828     subsection    = section ,
829     subsubsection = section ,
830     subparagraph  = paragraph ,
831     enumi         = item ,
832     enumii        = item ,
833     enumiii       = item ,
834     enumiv        = item ,
835   } ,
836 }

```

### 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_resettters_seq` with the `counterresetby` option.

```

837 \seq_new:N \l__zrefclever_counter_resettters_seq
838 \keys_define:nn { zref-clever / label }
839 {
840   counterresettters .code:n =
841   {
842     \clist_map_inline:nn {#1}
843     {
844       \seq_if_in:NnF \l__zrefclever_counter_resettters_seq {##1}
845       {
846         \seq_put_right:Nn
847         \l__zrefclever_counter_resettters_seq {##1}
848       }
849     }
850   } ,
851   counterresettters .initial:n =
852   {
853     part ,
854     chapter ,
855     section ,
856     subsection ,
857     subsubsection ,
858     paragraph ,
859     subparagraph ,
860   },
861   counterresettters .value_required:n = true ,
862 }

```

### **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_resettters_seq`.

```

863 \prop_new:N \l__zrefclever_counter_resetby_prop
864 \keys_define:nn { zref-clever / label }
865 {
866   counterresetby .code:n =
867   {
868     \keyval_parse:nnn
869     {
870       \msg_warning:nnn { zref-clever }
871       { key-requires-value } { counterresetby }
872     }
873     {
874       \__zrefclever_prop_put_non_empty:Nnn
875       \l__zrefclever_counter_resetby_prop
876     }
877     {#1}

```

```

878     } ,
879     counterresetby .value_required:n = true ,
880     counterresetby .initial:n =
881     {

```

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.

```

882         enumii = enumi ,
883         enumiii = enumii ,
884         enumiv = enumiii ,
885     } ,
886 }

```

### currentcounter option

`\l_zrefclever_current_counter_tl` is pretty much the starting point of all of the data specification for label setting done by `zref` with our setup for it. It exists because we must provide some “handle” to specify the current counter for packages/features that do not set `\@currentcounter` appropriately.

```

887 \tl_new:N \l_zrefclever_current_counter_tl
888 \keys_define:nn { zref-clever / label }
889 {
890     currentcounter .tl_set:N = \l_zrefclever_current_counter_tl ,
891     currentcounter .value_required:n = true ,
892     currentcounter .initial:n = \@currentcounter ,
893 }

```

### 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 `\zceref` or to `\zcsetup` or at load time, only “not necessarily type-specific” options are pertinent here. However, they *may* either be type-specific or language-specific, and thus must be stored in a property list, `\l_zrefclever_ref_options_prop`, in order to be retrieved from the option *name* by `\_zrefclever_get_ref_string:nN` and `\_zrefclever_get_ref_font:nN` according to context and precedence rules.

The keys are set so that any value, including an empty one, is added to `\l_zrefclever_ref_options_prop`, while a key with *no value* removes the property from the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.2.

```

894 \prop_new:N \l_zrefclever_ref_options_prop
895 \seq_map_inline:Nn
896   \c_zrefclever_ref_options_reference_seq
897   {
898     \keys_define:nn { zref-clever / reference }
899     {
900         #1 .default:V = \c_novalue_tl ,
901         #1 .code:n =
902         {
903             \tl_if_novalue:nTF {##1}
904             { \prop_remove:Nn \l_zrefclever_ref_options_prop {#1} }

```

```

905         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
906     } ,
907 }
908 }

```

## Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: `label` and `reference`. Currently, the only use of this selection is the ability to exclude label related options from `\zcref`’s options. Anyway, for load-time package options and for `\zcsetup` we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```

909 \keys_define:nn { }
910 {
911     zref-clever / zcsetup .inherit:n =
912     {
913         zref-clever / label ,
914         zref-clever / reference ,
915     }
916 }

```

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

```

917 \ProcessKeysOptions { zref-clever / zcsetup }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

```

```

918 \NewDocumentCommand \zcsetup { m }
919 { \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 `\zcLanguageSetup` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The `<options>` should be given in the usual `key=val` format. The `<type>` does not need to pre-exist, the property list variable to store the properties for the type gets created if need be.

```

\zcRefTypeSetup      \zcRefTypeSetup {<type>} {<options>}

920 \NewDocumentCommand \zcRefTypeSetup { m m }
921 {
922     \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
923     { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }

```

```

924 \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
925 \keys_set:nn { zref-clever / typesetup } {#2}
926 }

```

(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>.

```

927 \seq_map_inline:Nn
928 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
929 {
930   \keys_define:nn { zref-clever / typesetup }
931   {
932     #1 .code:n =
933     {
934       \msg_warning:nnn { zref-clever }
935       { option-not-type-specific } {#1}
936     } ,
937   }
938 }

939 \seq_map_inline:Nn
940 \c__zrefclever_ref_options_typesetup_seq
941 {
942   \keys_define:nn { zref-clever / typesetup }
943   {
944     #1 .default:V = \c_novaluel_tl ,
945     #1 .code:n =
946     {
947       \tl_if_novalue:nTF {##1}
948       {
949         \prop_remove:cn
950         {
951           l__zrefclever_type_
952           \l__zrefclever_setup_type_tl _options_prop
953         }
954         {#1}
955       }
956       {
957         \prop_put:cnn
958         {
959           l__zrefclever_type_

```

```

960         \l__zrefclever_setup_type_tl _options_prop
961     }
962     {#1} {##1}
963 }
964 } ,
965 }
966 }

```

### 5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `<options>` argument of \zcLanguageSetup, any options made before the first `type` key declare “default” (non type-specific) translations. When the `type` key is given with a value, the options following it will set “type-specific” translations for that type. The current type can be switched off by an empty `type` key. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
967 \NewDocumentCommand \zcLanguageSetup { m m }
968 {
969   \group_begin:
970   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
971   \l__zrefclever_dict_language_tl
972   {
973     \tl_clear:N \l__zrefclever_setup_type_tl
974     \keys_set:nn { zref-clever / langsetup } {#2}
975   }
976   { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
977   \group_end:
978 }
979 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

\\_zrefclever\_declare\_type\_transl:nnnn A couple of auxiliary functions for the of `zref-clever/translation` keys set in  
\\_zrefclever\_declare\_default\_transl:nnn \zcLanguageSetup. They respectively declare (unconditionally set) “type-specific” and  
“default” translations.

```

    \_zrefclever_declare_type_transl:nnnn {<language>} {<type>}
    {<key>} {<translation>}
    \_zrefclever_declare_default_transl:nnn {<language>}
    {<key>} {<translation>}
980 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
981 {
982   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
983   { type- #2 - #3 } {#4}
984 }
985 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
986 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
987 {
988   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
989   { default- #2 } {#3}

```

```

990 }
991 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }

```

(End definition for \\_\_zrefclever\_declare\_type\_transl:nnnn and \\_\_zrefclever\_declare\_default\_transl:nnn.)

The set of keys for zref-clever/langsetup, which is used to set language-specific translations in \zcLanguageSetup.

```

992 \keys_define:nn { zref-clever / langsetup }
993 {
994   type .code:n =
995   {
996     \tl_if_empty:NTF {#1}
997     { \tl_clear:N \l__zrefclever_setup_type_tl }
998     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
999   } ,
1000 }
1001 \seq_map_inline:Nn
1002 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1003 {
1004   \keys_define:nn { zref-clever / langsetup }
1005   {
1006     #1 .value_required:n = true ,
1007     #1 .code:n =
1008     {
1009       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1010       {
1011         \__zrefclever_declare_default_transl:Vnn
1012         \l__zrefclever_dict_language_tl
1013         {#1} {##1}
1014       }
1015       {
1016         \msg_warning:nnn { zref-clever }
1017         { option-not-type-specific } {#1}
1018       }
1019     } ,
1020   }
1021 }
1022 \seq_map_inline:Nn
1023 \c__zrefclever_ref_options_possibly_type_specific_seq
1024 {
1025   \keys_define:nn { zref-clever / langsetup }
1026   {
1027     #1 .value_required:n = true ,
1028     #1 .code:n =
1029     {
1030       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1031       {
1032         \__zrefclever_declare_default_transl:Vnn
1033         \l__zrefclever_dict_language_tl
1034         {#1} {##1}
1035       }
1036       {
1037         \__zrefclever_declare_type_transl:VVnn
1038         \l__zrefclever_dict_language_tl

```



```

1039         \l__zrefclever_setup_type_tl
1040         {#1} {##1}
1041     }
1042 } ,
1043 }
1044 }
1045 \seq_map_inline:Nn
1046 \c__zrefclever_ref_options_necessarily_type_specific_seq
1047 {
1048     \keys_define:nn { zref-clever / langsetup }
1049     {
1050         #1 .value_required:n = true ,
1051         #1 .code:n =
1052         {
1053             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1054             {
1055                 \msg_warning:nnn { zref-clever }
1056                 { option-only-type-specific } {#1}
1057             }
1058             {
1059                 \__zrefclever_declare_type_transl:VVnn
1060                 \l__zrefclever_dict_language_tl
1061                 \l__zrefclever_setup_type_tl
1062                 {#1} {##1}
1063             }
1064         } ,
1065     }
1066 }

```

## 6 User interface

### 6.1 \zcref

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

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

```

1067 \NewDocumentCommand \zcref { s O { } m }
1068 { \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>}
```

```

1069 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1070 {
1071     \group_begin:

```

Set options.

```

1072     \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1073     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1074     \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.

```

1075     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with zref-check.

```

1076     \bool_lazy_and:nnT
1077     { \l__zrefclever_zrefcheck_available_bool }
1078     { \l__zrefclever_zcref_with_check_bool }
1079     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1080     \bool_lazy_or:nnT
1081     { \l__zrefclever_typeset_sort_bool }
1082     { \l__zrefclever_typeset_range_bool }
1083     { \__zrefclever_sort_labels: }

```

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

```

1084     \group_begin:
1085     \l__zrefclever_ref_typeset_font_tl
1086     \__zrefclever_typeset_refs:
1087     \group_end:

```

Typeset note.

```

1088     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1089     {
1090         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1091         \l_tmpa_tl
1092         \l__zrefclever_zcref_note_tl
1093     }

```

Integration with zref-check.

```

1094     \bool_lazy_and:nnT
1095     { \l__zrefclever_zrefcheck_available_bool }
1096     { \l__zrefclever_zcref_with_check_bool }
1097     {
1098         \zrefcheck_zcref_end_label_maybe:
1099         \zrefcheck_zcref_run_checks_on_labels:n
1100         { \l__zrefclever_zcref_labels_seq }
1101     }
1102     \group_end:
1103 }

```

(End definition for `\__zrefclever_zcref:nnnn`.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1104 \seq_new:N \l__zrefclever_zcref_labels_seq
1105 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for `\l__zrefclever_zcref_labels_seq` and `\l__zrefclever_link_star_bool`.)

## 6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```

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

1106 \NewDocumentCommand \zcpageref { s O { } m }
1107 {
1108   \IfBooleanTF {#1}
1109     { \zcref*[#2, ref = page] {#3} }
1110     { \zcref [ #2, ref = page] {#3} }
1111 }

```

(End definition for \zcpageref.)

## 7 Sorting

Sorting is certainly a “big task” for zref-clever but, in the end, it boils down to “carefully done branching”, and quite some of it. The sorting of “page” references is very much lightened by the availability of `abspage`, from the `zref-abspage` module, which offers “just what we need” for our purposes. The sorting of “default” references falls on two main cases: i) labels of the same type; ii) labels of different types. The first case is sorted according to the priorities set by the `typesort` option or, if that is silent for the case, by the order in which labels were given by the user in `\zcref`. The second case is the most involved one, since it is possible for multiple counters to be bundled together in a single reference type. Because of this, sorting must take into account the whole chain of “enclosing counters” for the counters of the labels at hand.

<pre> \l_zrefclever_label_type_a_tl \l_zrefclever_label_type_b_tl \l_zrefclever_label_enclcnt_a_tl \l_zrefclever_label_enclcnt_b_tl \l_zrefclever_label_enclval_a_tl \l_zrefclever_label_enclval_b_tl </pre>	<pre> 1112 \tl_new:N \l__zrefclever_label_type_a_tl 1113 \tl_new:N \l__zrefclever_label_type_b_tl 1114 \tl_new:N \l__zrefclever_label_enclcnt_a_tl 1115 \tl_new:N \l__zrefclever_label_enclcnt_b_tl 1116 \tl_new:N \l__zrefclever_label_enclval_a_tl 1117 \tl_new:N \l__zrefclever_label_enclval_b_tl </pre>	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p>
--	--	--

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

<pre> \l_zrefclever_sort_decided_bool </pre>	<pre> 1118 \bool_new:N \l__zrefclever_sort_decided_bool </pre>	<p>Auxiliary variable for \l__zrefclever_sort_default_same_type:nn, signals if the sorting between two labels has been decided or not.</p>
--	--	--

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

<pre> \l_zrefclever_sort_prior_a_int \l_zrefclever_sort_prior_b_int </pre>	<pre> 1119 \int_new:N \l__zrefclever_sort_prior_a_int 1120 \int_new:N \l__zrefclever_sort_prior_b_int </pre>	<p>Auxiliary variables for \l__zrefclever_sort_default_different_types:nn. Store the sort priority of the “current” and “next” labels.</p>
--	--	--

(End definition for \l\_zrefclever\_sort\_prior\_a\_int and \l\_zrefclever\_sort\_prior\_b\_int.)

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

```
1121 \seq_new:N \l__zrefclever_label_types_seq
```

(End definition for `\l__zrefclever_label_types_seq`.)

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

```
1122 \cs_new_protected:Npn \__zrefclever_sort_labels:
1123 {
```

Store label types sequence.

```
1124   \seq_clear:N \l__zrefclever_label_types_seq
1125   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1126   {
1127     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1128     \__zrefclever_label_type_put_new_right:n
1129   }
```

Sort.

```
1130   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1131   {
1132     \zref@ifrefundefined {##1}
1133     {
1134       \zref@ifrefundefined {##2}
1135       {
1136         % Neither label is defined.
1137         \sort_return_same:
1138       }
1139       {
1140         % The second label is defined, but the first isn't, leave the
1141         % undefined first (to be more visible).
1142         \sort_return_same:
1143       }
1144     }
1145     {
1146       \zref@ifrefundefined {##2}
1147       {
1148         % The first label is defined, but the second isn't, bring the
1149         % second forward.
1150         \sort_return_swapped:
1151       }
1152       {
1153         % The interesting case: both labels are defined. References
1154         % to the "default" property or to the "page" are quite
1155         % different with regard to sorting, so we branch them here to
1156         % specialized functions.
1157         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
```

```

1158         { \_zrefclever_sort_page:nn {##1} {##2} }
1159         { \_zrefclever_sort_default:nn {##1} {##2} }
1160     }
1161 }
1162 }
1163 }

```

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

\\_zrefclever\_label\_type\_put\_new\_right:n Auxiliary function used to store the order in which reference types appear in the label list supplied by the user in \zceref. It is expected to be run inside \\_zrefclever\_sort\_labels:, and stores the types sequence in \l\\_zrefclever\_label\_types\_seq. I have tried to handle the same task inside \seq\_sort:Nn in \\_zrefclever\_sort\_labels: to spare mapping over \l\\_zrefclever\_zceref\_labels\_seq, but it turned out it not to be easy to rely on the order the labels get processed at that point, since the variable is being sorted there. Besides, the mapping is simple, not a particularly expensive operation. Anyway, this keeps things clean.

```

\_zrefclever_label_type_put_new_right:n {<label>}

1164 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1165 {
1166   \tl_set:Nx \l\_zrefclever_label_type_a_tl
1167   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1168   \seq_if_in:NVF \l\_zrefclever_label_types_seq
1169   \l\_zrefclever_label_type_a_tl
1170   {
1171     \seq_put_right:NV \l\_zrefclever_label_types_seq
1172     \l\_zrefclever_label_type_a_tl
1173   }
1174 }

```

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

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

```

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

1175 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
1176 {
1177   \tl_set:Nx \l\_zrefclever_label_type_a_tl
1178   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1179   \tl_set:Nx \l\_zrefclever_label_type_b_tl
1180   { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1181
1182   \bool_if:nTF
1183   {
1184     % The second label has a type, but the first doesn't, leave the
1185     % undefined first (to be more visible).
1186     \tl_if_empty_p:N \l\_zrefclever_label_type_a_tl &&
1187     ! \tl_if_empty_p:N \l\_zrefclever_label_type_b_tl

```

```

1188 }
1189 { \sort_return_same: }
1190 {
1191   \bool_if:nTF
1192   {
1193     % The first label has a type, but the second doesn't, bring the
1194     % second forward.
1195     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1196     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1197   }
1198   { \sort_return_swapped: }
1199   {
1200     \bool_if:nTF
1201     {
1202       % The interesting case: both labels have a type...
1203       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1204       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1205     }
1206     {
1207       \tl_if_eq:NNTF
1208       \l__zrefclever_label_type_a_tl
1209       \l__zrefclever_label_type_b_tl
1210       % ...and it's the same type.
1211       { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1212       % ...and they are different types.
1213       { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1214     }
1215     {
1216       % Neither label has a type. We can't do much of meaningful
1217       % here, but if it's the same counter, compare it.
1218       \exp_args:Nxx \tl_if_eq:nnTF
1219       { \zref@extractdefault {#1} { zc@counter } { } }
1220       { \zref@extractdefault {#2} { zc@counter } { } }
1221       {
1222         \int_compare:nNnTF
1223         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1224         >
1225         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1226         { \sort_return_swapped: }
1227         { \sort_return_same: }
1228       }
1229       { \sort_return_same: }
1230     }
1231   }
1232 }
1233 }

```

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

Variant not provided by the kernel, for use in \\_\_zrefclever\_sort\_default\_-same\_type:nn.

```

1234 \cs_generate_variant:Nn \tl_reverse_items:n { V }

```

```

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

```

```

1235 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2

```

```

1236 {
1237   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1238     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1239   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1240     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1241   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1242     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1243   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1244     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1245   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1246     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1247   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1248     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1249   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1250     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1251   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1252     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1253
1254   \bool_set_false:N \l__zrefclever_sort_decided_bool
1255   \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1256   {
1257     \bool_if:nTF
1258     {
1259       % Both are empty: neither label has any (further) "enclosing
1260       % counters" (left).
1261       \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1262       \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1263     }
1264     {
1265       \exp_args:Nxx \tl_if_eq:nnTF
1266       { \zref@extractdefault {#1} { zc@counter } { } }
1267       { \zref@extractdefault {#2} { zc@counter } { } }
1268       {
1269         \bool_set_true:N \l__zrefclever_sort_decided_bool
1270         \int_compare:nNnTF
1271         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1272         >
1273         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1274         { \sort_return_swapped: }
1275         { \sort_return_same: }
1276       }
1277     }
1278     \msg_warning:nnnn { zref-clever }
1279     { counters-not-nested } {#1} {#2}
1280     \bool_set_true:N \l__zrefclever_sort_decided_bool
1281     \sort_return_same:
1282   }
1283 }
1284 {
1285   \bool_if:nTF
1286   {
1287     % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1288     \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1289   }

```

```

1290 {
1291   \int_zero:N \l_tmpb_int
1292   \tl_map_inline:Nn \l__zrefclever_label_enclcnt_b_tl
1293   {
1294     \int_incr:N \l_tmpb_int
1295     \exp_args:Nnx \tl_if_eq:nnT {##1}
1296     { \zref@extractdefault {#1} { zc@counter } { } }
1297     {
1298       \tl_map_break:n
1299       {
1300         \int_show:N \l_tmpb_int
1301         \int_compare:nNnTF
1302         { \zref@extractdefault {#1} { zc@cntval } { } }
1303         >
1304         {
1305           \tl_item:Nn \l__zrefclever_label_enclval_b_tl
1306           { \l_tmpb_int }
1307         }
1308         { \sort_return_swapped: }
1309         { \sort_return_same: }
1310         \bool_set_true:N \l__zrefclever_sort_decided_bool
1311       }
1312     }
1313   }
1314   \bool_if:NF \l__zrefclever_sort_decided_bool
1315   {
1316     \msg_warning:nnnn { zref-clever }
1317     { counters-not-nested } {#1} {#2}
1318     \bool_set_true:N \l__zrefclever_sort_decided_bool
1319     \sort_return_same:
1320   }
1321 }
1322 {
1323   \bool_if:nTF
1324   {
1325     % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1326     \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1327   }
1328   {
1329     \int_zero:N \l_tmpa_int
1330     \tl_map_inline:Nn \l__zrefclever_label_enclcnt_a_tl
1331     {
1332       \int_incr:N \l_tmpa_int
1333       \exp_args:Nnx \tl_if_eq:nnT {##1}
1334       { \zref@extractdefault {#2} { zc@counter } { } }
1335       {
1336         \tl_map_break:n
1337         {
1338           \int_compare:nNnTF
1339           {
1340             \tl_item:Nn
1341             \l__zrefclever_label_enclval_a_tl
1342             { \l_tmpa_int }
1343           }

```



```

1344         <
1345     {
1346         \zref@extractdefault {#2}
1347         { zc@cntval } { }
1348     }
1349     { \sort_return_same: }
1350     { \sort_return_swapped: }
1351     \bool_set_true:N
1352     \l__zrefclever_sort_decided_bool
1353 }
1354 }
1355 }
1356 \bool_if:NF \l__zrefclever_sort_decided_bool
1357 {
1358     \msg_warning:nnnn { zref-clever }
1359     { counters-not-nested } {#1} {#2}
1360     \bool_set_true:N \l__zrefclever_sort_decided_bool
1361     \sort_return_same:
1362 }
1363 }
1364 {
1365     % Neither is empty: we can (possibly) compare the values
1366     % of the current enclosing counter in the loop, if they
1367     % are equal, we are still in the loop, if they are not, a
1368     % sorting decision can be made directly.
1369     \exp_args:Nxx \tl_if_eq:nnTF
1370     { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1371     { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1372     {
1373         \int_compare:nNnTF
1374         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1375         =
1376         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1377         {
1378             \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1379             { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1380             \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1381             { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1382             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1383             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1384             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1385             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1386         }
1387         {
1388             \bool_set_true:N \l__zrefclever_sort_decided_bool
1389             \int_compare:nNnTF
1390             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1391             >
1392             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1393             { \sort_return_swapped: }
1394             { \sort_return_same: }
1395         }
1396     }
1397 }

```

```

1398         \msg_warning:nnnn { zref-clever }
1399         { counters-not-nested } {#1} {#2}
1400         \bool_set_true:N \l__zrefclever_sort_decided_bool
1401         \sort_return_same:
1402     }
1403 }
1404 }
1405 }
1406 }
1407 }

```

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

```

__zrefclever_sort_default_different_types:nn
    \__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
1408 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1409 {

```

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

```

1410     \int_zero:N \l__zrefclever_sort_prior_a_int
1411     \int_zero:N \l__zrefclever_sort_prior_b_int
1412     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1413     {
1414         \tl_if_eq:nnTF {##2} {{othertypes}}
1415         {
1416             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1417             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1418             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1419             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1420         }
1421         {
1422             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1423             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1424             {
1425                 \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1426                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1427             }
1428         }
1429     }

```

Then do the actual sorting.

```

1430     \bool_if:nTF
1431     {
1432         \int_compare_p:nNn
1433         { \l__zrefclever_sort_prior_a_int } <
1434         { \l__zrefclever_sort_prior_b_int }
1435     }
1436     { \sort_return_same: }
1437     {
1438         \bool_if:nTF
1439         {
1440             \int_compare_p:nNn
1441             { \l__zrefclever_sort_prior_a_int } >

```

```

1442         { \l__zrefclever_sort_prior_b_int }
1443     }
1444     { \sort_return_swapped: }
1445     {
1446         % Sort priorities are equal: the type that occurs first in
1447         % ‘labels’, as given by the user, is kept (or brought) forward.
1448         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1449         {
1450             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1451             { \seq_map_break:n { \sort_return_same: } }
1452             {
1453                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1454                 { \seq_map_break:n { \sort_return_swapped: } }
1455             }
1456         }
1457     }
1458 }
1459 }

```

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

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

```

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

1460 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1461 {
1462     \int_compare:nNnTF
1463     { \zref@extractdefault {#1} { abspage } {-1} }
1464     >
1465     { \zref@extractdefault {#2} { abspage } {-1} }
1466     { \sort_return_swapped: }
1467     { \sort_return_same: }
1468 }

```

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

## 8 Typesetting

“Typesetting” the reference, which here includes the parsing of the labels and eventual compression of labels in sequence into ranges, is definitely the “crux” of `zref-clever`. This because we process the label set as a stack, in a single pass, and hence “parsing”, “compressing”, and “typesetting” must be decided upon at the same time, making it difficult to slice the job into more specific and self-contained tasks. So, do bear this in mind before you curse me for the length of some of the functions below, or before a more orthodox “docstripper” complains about me not sticking to code commenting conventions to keep the code more readable in the `.dtx` file.

While processing the label stack (kept in `\l__zrefclever_typeset_labels_seq`), `\__zrefclever_typeset_refs:` “sees” two labels, and two labels only, the “current” one

(kept in `\l__zrefclever_label_a_tl`), and the “next” one (kept in `\l__zrefclever_label_b_tl`). However, the typesetting needs (a lot) more information than just these two immediate labels to make a number of critical decisions. Some examples: i) We cannot know if labels “current” and “next” of the same type are a “pair”, or just “elements in a list”, until we examine the label after “next”; ii) If the “next” label is of the same type as the “current”, and it is in immediate sequence to it, it potentially forms a “range”, but we cannot know if “next” is actually the end of the range until we examined an arbitrary number of labels, and found one which is not in sequence from the previous one; iii) When processing a type block, the “name” comes first, however, we only know if that name should be plural, or if it should be included in the hyperlink, after processing an arbitrary number of labels and find one of a different type. One could naively assume that just examining “next” would be enough for this, since we can know if it is of the same type or not. Alas, “there be ranges”, and a compression operation may boil down to a single element, so we have to process the whole type block to know how its name should be typeset; iv) Similar issues apply to lists of type blocks, each of which is of arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

We handle this by storing the reference “pieces” in “queues”, instead of typesetting them immediately upon processing. The “queues” get typeset at the point where all the information needed is available, which usually happens when a type block finishes (we see something of a different type in “next”, signaled by `\l__zrefclever_last_of_type_bool`), or the stack itself finishes (has no more elements, signaled by `\l__zrefclever_typeset_last_bool`). And, in processing a type block, the type “name” gets added last (on the left) of the queue. The very first reference of its type always follows the name, since it may form a hyperlink with it (so we keep it stored separately, in `\l__zrefclever_type_first_label_tl`, with `\l__zrefclever_type_first_label_type_tl` being its type). And, since we may need up to two type blocks in storage before typesetting, we have two of these “queues”: `\l__zrefclever_typeset_queue_curr_tl` and `\l__zrefclever_typeset_queue_prev_tl`.

Some of the relevant cases (e.g., distinguishing “pair” from “list”) are handled by counters, the main ones are: one for the “type” (`\l__zrefclever_type_count_int`) and one for the “label in the current type block” (`\l__zrefclever_label_count_int`).

Range compression, in particular, relies heavily on counting to be able to distinguish relevant cases. `\l__zrefclever_range_count_int` counts the number of elements in the current sequential “streak”, and `\l__zrefclever_range_same_count_int` counts the number of *equal* elements in that same “streak”. The difference between the two allows us to distinguish the cases in which a range actually “skips” a number in the sequence, in which case we should use a range separator, from when they are after all just contiguous, in which case a pair separator is called for. Since, as usual, we can only know this when an arbitrary long “streak” finishes, we have to store the label which (potentially) begins a range (kept in `\l__zrefclever_range_beg_label_tl`). `\l__zrefclever_next_maybe_range_bool` signals when “next” is potentially a range with “current”, and `\l__zrefclever_next_is_same_bool` when their values are actually equal.

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this (and good ones at that) see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be an option receiving the label(s) not to be compressed, this would be a repetition, but would keep the syntax clean. All in all, probably the best is simply not to

allow individual inhibition of compression. We can already control compression of each `\zcref` call with existing options, this should be enough. I don't think the small extra flexibility individual label control for this would grant is worth the syntax disruption it would entail. Anyway, it would be easy to deal with this in case the need arose, by just adding another condition (coming from whatever the chosen syntax was) when we check for `\__zrefclever_labels_in_sequence:nn` in `\__zrefclever_typeset_refs_not_last_of_type:.` But I remain unconvinced of the pertinence of doing so.

## Variables

<code>\l_zrefclever_typeset_labels_seq</code>	Auxiliary variables for <code>\__zrefclever_typeset_refs</code> : main stack control.
<code>\l_zrefclever_typeset_last_bool</code>	1469 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l_zrefclever_last_of_type_bool</code>	1470 <code>\bool_new:N \l_zrefclever_typeset_last_bool</code>
	1471 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>
	(End definition for <code>\l_zrefclever_typeset_labels_seq</code> , <code>\l_zrefclever_typeset_last_bool</code> , and <code>\l__zrefclever_last_of_type_bool</code> .)
 <code>\l_zrefclever_type_count_int</code>	 Auxiliary variables for <code>\__zrefclever_typeset_refs</code> : main counters.
<code>\l_zrefclever_label_count_int</code>	1472 <code>\int_new:N \l_zrefclever_type_count_int</code>
	1473 <code>\int_new:N \l_zrefclever_label_count_int</code>
	(End definition for <code>\l_zrefclever_type_count_int</code> and <code>\l_zrefclever_label_count_int</code> .)
 <code>\l__zrefclever_label_a_tl</code>	 Auxiliary variables for <code>\__zrefclever_typeset_refs</code> : main “queue” control and storage.
<code>\l_zrefclever_label_b_tl</code>	
<code>\l_zrefclever_typeset_queue_prev_tl</code>	1474 <code>\tl_new:N \l_zrefclever_label_a_tl</code>
<code>\l_zrefclever_typeset_queue_curr_tl</code>	1475 <code>\tl_new:N \l_zrefclever_label_b_tl</code>
<code>\l_zrefclever_type_first_label_tl</code>	1476 <code>\tl_new:N \l_zrefclever_typeset_queue_prev_tl</code>
<code>\l__zrefclever_type_first_label_type_tl</code>	1477 <code>\tl_new:N \l_zrefclever_typeset_queue_curr_tl</code>
	1478 <code>\tl_new:N \l_zrefclever_type_first_label_tl</code>
	1479 <code>\tl_new:N \l_zrefclever_type_first_label_type_tl</code>
	(End definition for <code>\l__zrefclever_label_a_tl</code> and others.)
 <code>\l__zrefclever_type_name_tl</code>	 Auxiliary variables for <code>\__zrefclever_typeset_refs</code> : type name handling.
<code>\l_zrefclever_name_in_link_bool</code>	1480 <code>\tl_new:N \l__zrefclever_type_name_tl</code>
<code>\l_zrefclever_name_format_tl</code>	1481 <code>\bool_new:N \l_zrefclever_name_in_link_bool</code>
<code>\l_zrefclever_name_format_fallback_tl</code>	1482 <code>\tl_new:N \l_zrefclever_name_format_tl</code>
	1483 <code>\tl_new:N \l_zrefclever_name_format_fallback_tl</code>
	(End definition for <code>\l__zrefclever_type_name_tl</code> and others.)
 <code>\l_zrefclever_range_count_int</code>	 Auxiliary variables for <code>\__zrefclever_typeset_refs</code> : range handling.
<code>\l_zrefclever_range_same_count_int</code>	1484 <code>\int_new:N \l__zrefclever_range_count_int</code>
<code>\l_zrefclever_range_beg_label_tl</code>	1485 <code>\int_new:N \l_zrefclever_range_same_count_int</code>
<code>\l_zrefclever_next_maybe_range_bool</code>	1486 <code>\tl_new:N \l_zrefclever_range_beg_label_tl</code>
<code>\l_zrefclever_next_is_same_bool</code>	1487 <code>\bool_new:N \l_zrefclever_next_maybe_range_bool</code>
	1488 <code>\bool_new:N \l_zrefclever_next_is_same_bool</code>
	(End definition for <code>\l_zrefclever_range_count_int</code> and others.)

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

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangeseq_tl
\l__zrefclever_refpre_out_tl
\l__zrefclever_refpos_out_tl
\l__zrefclever_refpre_in_tl
\l__zrefclever_refpos_in_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_out_tl
\l__zrefclever_reffont_in_tl

```

```

1489 \tl_new:N \l__zrefclever_tpairsep_tl
1490 \tl_new:N \l__zrefclever_tlistsep_tl
1491 \tl_new:N \l__zrefclever_tlastsep_tl
1492 \tl_new:N \l__zrefclever_namesep_tl
1493 \tl_new:N \l__zrefclever_pairsep_tl
1494 \tl_new:N \l__zrefclever_listsep_tl
1495 \tl_new:N \l__zrefclever_lastsep_tl
1496 \tl_new:N \l__zrefclever_rangeseq_tl
1497 \tl_new:N \l__zrefclever_refpre_out_tl
1498 \tl_new:N \l__zrefclever_refpos_out_tl
1499 \tl_new:N \l__zrefclever_refpre_in_tl
1500 \tl_new:N \l__zrefclever_refpos_in_tl
1501 \tl_new:N \l__zrefclever_namefont_tl
1502 \tl_new:N \l__zrefclever_reffont_out_tl
1503 \tl_new:N \l__zrefclever_reffont_in_tl

```

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

## Main functions

`\__zrefclever_typeset_refs`: Main typesetting function for `\zcref`.

```

1504 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1505 {
1506   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1507   \l__zrefclever_zcref_labels_seq
1508   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1509   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1510   \tl_clear:N \l__zrefclever_type_first_label_tl
1511   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1512   \tl_clear:N \l__zrefclever_range_beg_label_tl
1513   \int_zero:N \l__zrefclever_label_count_int
1514   \int_zero:N \l__zrefclever_type_count_int
1515   \int_zero:N \l__zrefclever_range_count_int
1516   \int_zero:N \l__zrefclever_range_same_count_int
1517
1518   % Get type block options (not type-specific).
1519   \__zrefclever_get_ref_string:nN { tpairsep }
1520   \l__zrefclever_tpairsep_tl
1521   \__zrefclever_get_ref_string:nN { tlistsep }
1522   \l__zrefclever_tlistsep_tl
1523   \__zrefclever_get_ref_string:nN { tlastsep }
1524   \l__zrefclever_tlastsep_tl
1525
1526   % Process label stack.
1527   \bool_set_false:N \l__zrefclever_typeset_last_bool
1528   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1529   {
1530     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1531     \l__zrefclever_label_a_tl
1532     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1533     {
1534       \tl_clear:N \l__zrefclever_label_b_tl

```

```

1535         \bool_set_true:N \l__zrefclever_typeset_last_bool
1536     }
1537     {
1538         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1539         \l__zrefclever_label_b_tl
1540     }
1541
1542 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1543 {
1544     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1545     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1546 }
1547 {
1548     \tl_set:Nx \l__zrefclever_label_type_a_tl
1549     {
1550         \zref@extractdefault
1551         { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1552     }
1553     \tl_set:Nx \l__zrefclever_label_type_b_tl
1554     {
1555         \zref@extractdefault
1556         { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1557     }
1558 }
1559
1560 % First, we establish whether the "current label" (i.e. 'a') is the
1561 % last one of its type. This can happen because the "next label"
1562 % (i.e. 'b') is of a different type (or different definition status),
1563 % or because we are at the end of the list.
1564 \bool_if:NTF \l__zrefclever_typeset_last_bool
1565 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1566 {
1567     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1568     {
1569         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1570         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1571         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1572     }
1573     {
1574         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1575         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1576         {
1577             % Neither is undefined, we must check the types.
1578             \bool_if:nTF
1579             {
1580                 % Both empty: same "type".
1581                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1582                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1583             }
1584             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1585             {
1586                 \bool_if:nTF
1587                 {
1588                     % Neither empty: compare types.

```

```

1589         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1590         &&
1591         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1592     }
1593     {
1594         \tl_if_eq:NNTF
1595             \l__zrefclever_label_type_a_tl
1596             \l__zrefclever_label_type_b_tl
1597         {
1598             \bool_set_false:N
1599                 \l__zrefclever_last_of_type_bool
1600         }
1601         {
1602             \bool_set_true:N
1603                 \l__zrefclever_last_of_type_bool
1604         }
1605     }
1606     % One empty, the other not: different "types".
1607     {
1608         \bool_set_true:N
1609             \l__zrefclever_last_of_type_bool
1610     }
1611 }
1612 }
1613 }
1614 }
1615
1616 % Handle warnings in case of reference or type undefined.
1617 \zref@refused { \l__zrefclever_label_a_tl }
1618 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1619 {}
1620 {
1621     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1622     {
1623         \msg_warning:nxx { zref-clever } { missing-type }
1624         { \l__zrefclever_label_a_tl }
1625     }
1626 }
1627
1628 % Get type-specific separators, refpre/pos and font options, once per
1629 % type.
1630 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1631 {
1632     \__zrefclever_get_ref_string:nN { namesep }
1633     \l__zrefclever_namesep_tl
1634     \__zrefclever_get_ref_string:nN { rangesep }
1635     \l__zrefclever_rangesep_tl
1636     \__zrefclever_get_ref_string:nN { pairsep }
1637     \l__zrefclever_pairsep_tl
1638     \__zrefclever_get_ref_string:nN { listsep }
1639     \l__zrefclever_listsep_tl
1640     \__zrefclever_get_ref_string:nN { lastsep }
1641     \l__zrefclever_lastsep_tl
1642     \__zrefclever_get_ref_string:nN { refpre }

```



```

1643         \l__zrefclever_refpre_out_tl
1644         \__zrefclever_get_ref_string:nN { refpos      }
1645         \l__zrefclever_refpos_out_tl
1646         \__zrefclever_get_ref_string:nN { refpre-in  }
1647         \l__zrefclever_refpre_in_tl
1648         \__zrefclever_get_ref_string:nN { refpos-in  }
1649         \l__zrefclever_refpos_in_tl
1650         \__zrefclever_get_ref_font:nN   { namefont   }
1651         \l__zrefclever_namefont_tl
1652         \__zrefclever_get_ref_font:nN   { reffont     }
1653         \l__zrefclever_reffont_out_tl
1654         \__zrefclever_get_ref_font:nN   { reffont-in  }
1655         \l__zrefclever_reffont_in_tl
1656     }
1657
1658     % Here we send this to a couple of auxiliary functions.
1659     \bool_if:NTF \l__zrefclever_last_of_type_bool
1660     % There exists no next label of the same type as the current.
1661     { \__zrefclever_typeset_refs_last_of_type: }
1662     % There exists a next label of the same type as the current.
1663     { \__zrefclever_typeset_refs_not_last_of_type: }
1664 }
1665 }

```

(End definition for \\_\_zrefclever\_typeset\_refs:.)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is *not* the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the “next” label and find something of a different “type” (loose here, maybe different definition status, maybe end of stack). So, though this is not very strict, \\_\_zrefclever\_typeset\_refs\_last\_of\_type: is more of a “wrapping up” function, and it is indeed the one which does the actual typesetting, while \\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type: is more of an “accumulation” function.

\\_\_zrefclever\_typeset\_refs\_last\_of\_type:

Handles typesetting when the current label is the last of its type.

```

1666 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1667 {
1668     % Process the current label to the current queue.
1669     \int_case:nnF { \l__zrefclever_label_count_int }
1670     {
1671         % It is the last label of its type, but also the first one, and that's
1672         % what matters here: just store it.
1673         { 0 }
1674         {
1675             \tl_set:NV \l__zrefclever_type_first_label_tl
1676             \l__zrefclever_label_a_tl
1677             \tl_set:NV \l__zrefclever_type_first_label_type_tl
1678             \l__zrefclever_label_type_a_tl
1679         }
1680
1681         % The last is the second: we have a pair (if not repeated).
1682         { 1 }
1683         {

```

```

1684 \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1685 {
1686   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1687   {
1688     \exp_not:V \l__zrefclever_pairsep_tl
1689     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1690   }
1691 }
1692 }
1693 }
1694 % Last is third or more of its type: without repetition, we'd have the
1695 % last element on a list, but control for possible repetition.
1696 {
1697   \int_case:nNnF { \l__zrefclever_range_count_int }
1698   {
1699     % There was no range going on.
1700     { 0 }
1701     {
1702       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1703       {
1704         \exp_not:V \l__zrefclever_lastsep_tl
1705         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1706       }
1707     }
1708     % Last in the range is also the second in it.
1709     { 1 }
1710     {
1711       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1712       {
1713         % We know 'range_beg_label' is not empty, since this is the
1714         % second element in the range, but the third or more in the
1715         % type list.
1716         \exp_not:V \l__zrefclever_listsep_tl
1717         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1718         \int_compare:nNnF
1719         { \l__zrefclever_range_same_count_int } = { 1 }
1720         {
1721           \exp_not:V \l__zrefclever_lastsep_tl
1722           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1723         }
1724       }
1725     }
1726   }
1727   % Last in the range is third or more in it.
1728   {
1729     \int_case:nNnF
1730     {
1731       \l__zrefclever_range_count_int -
1732       \l__zrefclever_range_same_count_int
1733     }
1734     {
1735       % Repetition, not a range.
1736       { 0 }
1737       {

```

```

1738 % If 'range_beg_label' is empty, it means it was also the
1739 % first of the type, and hence was already handled.
1740 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1741 {
1742   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1743   {
1744     \exp_not:V \l__zrefclever_lastsep_tl
1745     \__zrefclever_get_ref:V
1746       \l__zrefclever_range_beg_label_tl
1747   }
1748 }
1749 }
1750 % A 'range', but with no skipped value, treat as list.
1751 { 1 }
1752 {
1753   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1754   {
1755     % Ditto.
1756     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1757     {
1758       \exp_not:V \l__zrefclever_listsep_tl
1759       \__zrefclever_get_ref:V
1760         \l__zrefclever_range_beg_label_tl
1761     }
1762     \exp_not:V \l__zrefclever_lastsep_tl
1763     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1764   }
1765 }
1766 }
1767 {
1768   % An actual range.
1769   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1770   {
1771     % Ditto.
1772     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1773     {
1774       \exp_not:V \l__zrefclever_lastsep_tl
1775       \__zrefclever_get_ref:V
1776         \l__zrefclever_range_beg_label_tl
1777     }
1778     \exp_not:V \l__zrefclever_rangesep_tl
1779     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1780   }
1781 }
1782 }
1783 }
1784
1785 % Handle "range" option. The idea is simple: if the queue is not empty,
1786 % we replace it with the end of the range (or pair). We can still
1787 % retrieve the end of the range from 'label_a' since we know to be
1788 % processing the last label of its type at this point.
1789 \bool_if:NT \l__zrefclever_typeset_range_bool
1790 {
1791   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl

```

```

1792 {
1793   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1794   { }
1795   {
1796     \msg_warning:nxx { zref-clever } { single-element-range }
1797     { \l__zrefclever_type_first_label_type_tl }
1798   }
1799 }
1800 {
1801   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1802   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1803   { }
1804   {
1805     \__zrefclever_labels_in_sequence:nn
1806     { \l__zrefclever_type_first_label_tl }
1807     { \l__zrefclever_label_a_tl }
1808   }
1809   \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1810   {
1811     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1812     { \exp_not:V \l__zrefclever_pairsep_tl }
1813     { \exp_not:V \l__zrefclever_rangesep_tl }
1814     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1815   }
1816 }
1817 }
1818
1819 % Now that the type block is finished, we can add the name and the first
1820 % ref to the queue. Also, if "typeset" option is not "both", handle it
1821 % here as well.
1822 \__zrefclever_type_name_setup:
1823 \bool_if:nTF
1824 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1825 {
1826   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1827   { \__zrefclever_get_ref_first: }
1828 }
1829 {
1830   \bool_if:nTF
1831   { \l__zrefclever_typeset_ref_bool }
1832   {
1833     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1834     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1835   }
1836   {
1837     \bool_if:nTF
1838     { \l__zrefclever_typeset_name_bool }
1839     {
1840       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1841       {
1842         \bool_if:NTF \l__zrefclever_name_in_link_bool
1843         {
1844           \exp_not:N \group_begin:
1845           \exp_not:V \l__zrefclever_namefont_tl

```

```

1846 % It's two '@s', but escaped for DocStrip.
1847 \exp_not:N \hyper@@link
1848 {
1849   \zref@ifrefcontainsprop
1850   { \l__zrefclever_type_first_label_tl }
1851   { urluse }
1852   {
1853     \zref@extractdefault
1854     { \l__zrefclever_type_first_label_tl }
1855     { urluse } {}
1856   }
1857   {
1858     \zref@extractdefault
1859     { \l__zrefclever_type_first_label_tl }
1860     { url } {}
1861   }
1862 }
1863 {
1864   \zref@extractdefault
1865   { \l__zrefclever_type_first_label_tl }
1866   { anchor } {}
1867 }
1868 { \exp_not:V \l__zrefclever_type_name_tl }
1869 \exp_not:N \group_end:
1870 }
1871 {
1872   \exp_not:N \group_begin:
1873   \exp_not:V \l__zrefclever_namefont_tl
1874   \exp_not:V \l__zrefclever_type_name_tl
1875   \exp_not:N \group_end:
1876 }
1877 }
1878 }
1879 {
1880   % Logically, this case would correspond to "typeset=none", but
1881   % it should not occur, given that the options are set up to
1882   % typeset either "ref" or "name". Still, leave here a
1883   % sensible fallback, equal to the behavior of "both".
1884   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1885     { \l__zrefclever_get_ref_first: }
1886 }
1887 }
1888 }
1889
1890 % Typeset the previous type, if there is one.
1891 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1892 {
1893   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1894   { \l__zrefclever_tlistsep_tl }
1895   \l__zrefclever_typeset_queue_prev_tl
1896 }
1897
1898 % Wrap up loop, or prepare for next iteration.
1899 \bool_if:NTF \l__zrefclever_typeset_last_bool

```

```

1900 {
1901 % We are finishing, typeset the current queue.
1902 \int_case:nnF { \l__zrefclever_type_count_int }
1903 {
1904 % Single type.
1905 { 0 }
1906 { \l__zrefclever_typeset_queue_curr_tl }
1907 % Pair of types.
1908 { 1 }
1909 {
1910 \l__zrefclever_tpairsep_tl
1911 \l__zrefclever_typeset_queue_curr_tl
1912 }
1913 }
1914 {
1915 % Last in list of types.
1916 \l__zrefclever_tlastsep_tl
1917 \l__zrefclever_typeset_queue_curr_tl
1918 }
1919 }
1920 {
1921 % There are further labels, set variables for next iteration.
1922 \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1923 \l__zrefclever_typeset_queue_curr_tl
1924 \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1925 \tl_clear:N \l__zrefclever_type_first_label_tl
1926 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1927 \tl_clear:N \l__zrefclever_range_beg_label_tl
1928 \int_zero:N \l__zrefclever_label_count_int
1929 \int_incr:N \l__zrefclever_type_count_int
1930 \int_zero:N \l__zrefclever_range_count_int
1931 \int_zero:N \l__zrefclever_range_same_count_int
1932 }
1933 }

```

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

```

\__zrefclever_typeset_refs_not_last_of_type: Handles typesetting when the current label is not the last of its type.
1934 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1935 {
1936 % Signal if next label may form a range with the current one (only
1937 % considered if compression is enabled in the first place).
1938 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1939 \bool_set_false:N \l__zrefclever_next_is_same_bool
1940 \bool_if:NT \l__zrefclever_typeset_compress_bool
1941 {
1942 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1943 { }
1944 {
1945 \__zrefclever_labels_in_sequence:nn
1946 { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1947 }
1948 }
1949

```

```

1950 % Process the current label to the current queue.
1951 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1952 {
1953   % Current label is the first of its type (also not the last, but it
1954   % doesn't matter here): just store the label.
1955   \tl_set:NV \l__zrefclever_type_first_label_tl
1956     \l__zrefclever_label_a_tl
1957   \tl_set:NV \l__zrefclever_type_first_label_type_tl
1958     \l__zrefclever_label_type_a_tl
1959
1960   % If the next label may be part of a range, we set 'range_beg_label'
1961   % to "empty" (we deal with it as the "first", and must do it there, to
1962   % handle hyperlinking), but also step the range counters.
1963   \bool_if:NT \l__zrefclever_next_maybe_range_bool
1964   {
1965     \tl_clear:N \l__zrefclever_range_beg_label_tl
1966     \int_incr:N \l__zrefclever_range_count_int
1967     \bool_if:NT \l__zrefclever_next_is_same_bool
1968       { \int_incr:N \l__zrefclever_range_same_count_int }
1969   }
1970 }
1971 {
1972   % Current label is neither the first (nor the last) of its type.
1973   \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
1974   {
1975     % Starting, or continuing a range.
1976     \int_compare:nNnTF
1977       { \l__zrefclever_range_count_int } = { 0 }
1978     {
1979       % There was no range going, we are starting one.
1980       \tl_set:NV \l__zrefclever_range_beg_label_tl
1981         \l__zrefclever_label_a_tl
1982       \int_incr:N \l__zrefclever_range_count_int
1983       \bool_if:NT \l__zrefclever_next_is_same_bool
1984         { \int_incr:N \l__zrefclever_range_same_count_int }
1985     }
1986     {
1987       % Second or more in the range, but not the last.
1988       \int_incr:N \l__zrefclever_range_count_int
1989       \bool_if:NT \l__zrefclever_next_is_same_bool
1990         { \int_incr:N \l__zrefclever_range_same_count_int }
1991     }
1992   }
1993 }
1994 % Next element is not in sequence: there was no range, or we are
1995 % closing one.
1996 \int_case:nnF { \l__zrefclever_range_count_int }
1997 {
1998   % There was no range going on.
1999   { 0 }
2000   {
2001     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2002     {
2003       \exp_not:V \l__zrefclever_listsep_tl

```

```

2004         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2005     }
2006 }
2007 % Last is second in the range: if 'range_same_count' is also
2008 % '1', it's a repetition (drop it), otherwise, it's a "pair
2009 % within a list", treat as list.
2010 { 1 }
2011 {
2012     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2013     {
2014         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2015         {
2016             \exp_not:V \l__zrefclever_listsep_tl
2017             \l__zrefclever_get_ref:V
2018             \l__zrefclever_range_beg_label_tl
2019         }
2020         \int_compare:nNnF
2021         { \l__zrefclever_range_same_count_int } = { 1 }
2022         {
2023             \exp_not:V \l__zrefclever_listsep_tl
2024             \l__zrefclever_get_ref:V
2025             \l__zrefclever_label_a_tl
2026         }
2027     }
2028 }
2029 }
2030 {
2031 % Last is third or more in the range: if 'range_count' and
2032 % 'range_same_count' are the same, its a repetition (drop it),
2033 % if they differ by '1', its a list, if they differ by more,
2034 % it is a real range.
2035 \int_case:nnF
2036 {
2037     \l__zrefclever_range_count_int -
2038     \l__zrefclever_range_same_count_int
2039 }
2040 {
2041     { 0 }
2042     {
2043         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2044         {
2045             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2046             {
2047                 \exp_not:V \l__zrefclever_listsep_tl
2048                 \l__zrefclever_get_ref:V
2049                 \l__zrefclever_range_beg_label_tl
2050             }
2051         }
2052     }
2053     { 1 }
2054     {
2055         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2056         {
2057             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl

```



```

2058         {
2059             \exp_not:V \l__zrefclever_listsep_tl
2060             \__zrefclever_get_ref:V
2061             \l__zrefclever_range_beg_label_tl
2062         }
2063         \exp_not:V \l__zrefclever_listsep_tl
2064         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2065     }
2066 }
2067 }
2068 {
2069     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2070     {
2071         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2072         {
2073             \exp_not:V \l__zrefclever_listsep_tl
2074             \__zrefclever_get_ref:V
2075             \l__zrefclever_range_beg_label_tl
2076         }
2077         \exp_not:V \l__zrefclever_rangesep_tl
2078         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2079     }
2080 }
2081 }
2082 % Reset counters.
2083 \int_zero:N \l__zrefclever_range_count_int
2084 \int_zero:N \l__zrefclever_range_same_count_int
2085 }
2086 }
2087 % Step label counter for next iteration.
2088 \int_incr:N \l__zrefclever_label_count_int
2089 }

```

(End definition for `\__zrefclever_typeset_refs_not_last_of_type:.`)

## Aux functions

`\__zrefclever_get_ref:n` and `\__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `\__zrefclever_get_ref:n` handles all references but the first of its type, and `\__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `\__zrefclever_typeset_refs_last_of_type:` and `\__zrefclever_typeset_refs_not_last_of_type:.` And this difference results quite crucial for the  $\TeX$ nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `\__zrefclever_get_ref:n` and `\__zrefclever_get_ref_first:` get called, as they must, in the context of  $x$  type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to

use the `n` signature jargon). We also need to prevent premature expansion of material that can't be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```
2090 \cs_new_protected:Npn \__zrefclever_ref_default:
2091   { \zref@default }
2092 \cs_new_protected:Npn \__zrefclever_name_default:
2093   { \zref@default }
```

(End definition for `\__zrefclever_ref_default:` and `\__zrefclever_name_default:.`)

`\__zrefclever_get_ref:n` Handles a complete reference block to be accumulated in the “queue”, including “pre” and “pos” elements, and hyperlinking. For use with all labels, except the first of its type, which is done by `\__zrefclever_get_ref_first:.`

```
\__zrefclever_get_ref:n {<label>}

2094 \cs_new:Npn \__zrefclever_get_ref:n #1
2095   {
2096     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2097     {
2098       \bool_if:nTF
2099       {
2100         \l__zrefclever_use_hyperref_bool &&
2101         ! \l__zrefclever_link_star_bool
2102       }
2103       {
2104         \exp_not:N \group_begin:
2105         \exp_not:V \l__zrefclever_reffont_out_tl
2106         \exp_not:V \l__zrefclever_refpre_out_tl
2107         \exp_not:N \group_begin:
2108         \exp_not:V \l__zrefclever_reffont_in_tl
2109         % It's two '@s', but escaped for DocStrip.
2110         \exp_not:N \hyper@@link
2111         {
2112           \zref@ifrefcontainsprop {#1} { urluse }
2113           { \zref@extractdefault {#1} { urluse } { } }
2114           { \zref@extractdefault {#1} { url } { } }
2115         }
2116         { \zref@extractdefault {#1} { anchor } { } }
2117         {
2118           \exp_not:V \l__zrefclever_refpre_in_tl
2119           \zref@extractdefault {#1}
2120           { \l__zrefclever_ref_property_tl } { }
2121           \exp_not:V \l__zrefclever_refpos_in_tl
2122         }
2123         \exp_not:N \group_end:
2124         \exp_not:V \l__zrefclever_refpos_out_tl
2125         \exp_not:N \group_end:
```

```

2126     }
2127     {
2128         \exp_not:N \group_begin:
2129         \exp_not:V \l__zrefclever_reffont_out_tl
2130         \exp_not:V \l__zrefclever_refpre_out_tl
2131         \exp_not:N \group_begin:
2132         \exp_not:V \l__zrefclever_reffont_in_tl
2133         \exp_not:V \l__zrefclever_refpre_in_tl
2134         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2135         \exp_not:V \l__zrefclever_refpos_in_tl
2136         \exp_not:N \group_end:
2137         \exp_not:V \l__zrefclever_refpos_out_tl
2138         \exp_not:N \group_end:
2139     }
2140 }
2141 { \__zrefclever_ref_default: }
2142 }
2143 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

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

`\__zrefclever_get_ref_first:` Handles a complete reference block for the first label of its type to be accumulated in the “queue”, including “pre” and “pos” elements, hyperlinking, and the reference type “name”. It does not receive arguments, but relies on being called in the appropriate place in `\__zrefclever_typeset_refs_last_of_type:` where a number of variables are expected to be appropriately set for it to consume. Prominently among those is `\l__zrefclever_type_first_label_tl`, but it also expected to be called right after `\__zrefclever_type_name_setup:` which sets `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool` which it uses.

```

2144 \cs_new:Npn \__zrefclever_get_ref_first:
2145 {
2146   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2147   { \__zrefclever_ref_default: }
2148   {
2149     \bool_if:NTF \l__zrefclever_name_in_link_bool
2150     {
2151       \zref@ifrefcontainsprop
2152       { \l__zrefclever_type_first_label_tl }
2153       { \l__zrefclever_ref_property_tl }
2154       {
2155         % It's two '@s', but escaped for DocStrip.
2156         \exp_not:N \hyper@@link
2157         {
2158           \zref@ifrefcontainsprop
2159           { \l__zrefclever_type_first_label_tl } { urluse }
2160           {
2161             \zref@extractdefault
2162             { \l__zrefclever_type_first_label_tl }
2163             { urluse } { }
2164           }
2165           {
2166             \zref@extractdefault
2167             { \l__zrefclever_type_first_label_tl }
2168             { url } { }

```

```

2169         }
2170     }
2171     {
2172         \zref@extractdefault
2173         { \l__zrefclever_type_first_label_tl }
2174         { anchor } { }
2175     }
2176     {
2177         \exp_not:N \group_begin:
2178         \exp_not:V \l__zrefclever_namefont_tl
2179         \exp_not:V \l__zrefclever_type_name_tl
2180         \exp_not:N \group_end:
2181         \exp_not:V \l__zrefclever_namesep_tl
2182         \exp_not:N \group_begin:
2183         \exp_not:V \l__zrefclever_reffont_out_tl
2184         \exp_not:V \l__zrefclever_refpre_out_tl
2185         \exp_not:N \group_begin:
2186         \exp_not:V \l__zrefclever_reffont_in_tl
2187         \exp_not:V \l__zrefclever_refpre_in_tl
2188         \zref@extractdefault
2189         { \l__zrefclever_type_first_label_tl }
2190         { \l__zrefclever_ref_property_tl } { }
2191         \exp_not:V \l__zrefclever_refpos_in_tl
2192         \exp_not:N \group_end:
2193         % hyperlink makes it's own group, we'd like to close the
2194         % 'refpre-out' group after 'refpos-out', but... we close
2195         % it here, and give the trailing 'refpos-out' its own
2196         % group. This will result that formatting given to
2197         % 'refpre-out' will not reach 'refpos-out', but I see no
2198         % alternative, and this has to be handled specially.
2199         \exp_not:N \group_end:
2200     }
2201     \exp_not:N \group_begin:
2202     % Ditto: special treatment.
2203     \exp_not:V \l__zrefclever_reffont_out_tl
2204     \exp_not:V \l__zrefclever_refpos_out_tl
2205     \exp_not:N \group_end:
2206 }
2207 {
2208     \exp_not:N \group_begin:
2209     \exp_not:V \l__zrefclever_namefont_tl
2210     \exp_not:V \l__zrefclever_type_name_tl
2211     \exp_not:N \group_end:
2212     \exp_not:V \l__zrefclever_namesep_tl
2213     \__zrefclever_ref_default:
2214 }
2215 }
2216 {
2217     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2218     {
2219         \__zrefclever_name_default:
2220         \exp_not:V \l__zrefclever_namesep_tl
2221     }
2222     {

```

```

2223 \exp_not:N \group_begin:
2224 \exp_not:V \l__zrefclever_namefont_tl
2225 \exp_not:V \l__zrefclever_type_name_tl
2226 \exp_not:N \group_end:
2227 \exp_not:V \l__zrefclever_namesep_tl
2228 }
2229 \zref@ifrefcontainsprop
2230 { \l__zrefclever_type_first_label_tl }
2231 { \l__zrefclever_ref_property_tl }
2232 {
2233 \bool_if:nTF
2234 {
2235 \l__zrefclever_use_hyperref_bool &&
2236 ! \l__zrefclever_link_star_bool
2237 }
2238 {
2239 \exp_not:N \group_begin:
2240 \exp_not:V \l__zrefclever_reffont_out_tl
2241 \exp_not:V \l__zrefclever_refpre_out_tl
2242 \exp_not:N \group_begin:
2243 \exp_not:V \l__zrefclever_reffont_in_tl
2244 % It's two '@s', but escaped for DocStrip.
2245 \exp_not:N \hyper@@link
2246 {
2247 \zref@ifrefcontainsprop
2248 { \l__zrefclever_type_first_label_tl } { urluse }
2249 {
2250 \zref@extractdefault
2251 { \l__zrefclever_type_first_label_tl }
2252 { urluse } { }
2253 }
2254 {
2255 \zref@extractdefault
2256 { \l__zrefclever_type_first_label_tl }
2257 { url } { }
2258 }
2259 }
2260 {
2261 \zref@extractdefault
2262 { \l__zrefclever_type_first_label_tl }
2263 { anchor } { }
2264 }
2265 {
2266 \exp_not:V \l__zrefclever_refpre_in_tl
2267 \zref@extractdefault
2268 { \l__zrefclever_type_first_label_tl }
2269 { \l__zrefclever_ref_property_tl } { }
2270 \exp_not:V \l__zrefclever_refpos_in_tl
2271 }
2272 \exp_not:N \group_end:
2273 \exp_not:V \l__zrefclever_refpos_out_tl
2274 \exp_not:N \group_end:
2275 }
2276 {

```

```

2277         \exp_not:N \group_begin:
2278         \exp_not:V \l__zrefclever_reffont_out_tl
2279         \exp_not:V \l__zrefclever_refpre_out_tl
2280         \exp_not:N \group_begin:
2281         \exp_not:V \l__zrefclever_reffont_in_tl
2282         \exp_not:V \l__zrefclever_refpre_in_tl
2283         \zref@extractdefault
2284         { \l__zrefclever_type_first_label_tl }
2285         { \l__zrefclever_ref_property_tl } { }
2286         \exp_not:V \l__zrefclever_refpos_in_tl
2287         \exp_not:N \group_end:
2288         \exp_not:V \l__zrefclever_refpos_out_tl
2289         \exp_not:N \group_end:
2290     }
2291 }
2292 { \__zrefclever_ref_default: }
2293 }
2294 }
2295 }

```

(End definition for `\__zrefclever_get_ref_first:`)

`\__zrefclever_type_name_setup:` Auxiliary function to `\__zrefclever_typeset_refs_last_of_type:`. It is responsible for setting the type name variable `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool`. If a type name can't be found, `\l__zrefclever_type_name_tl` is cleared. The function takes no arguments, but is expected to be called in `\__zrefclever_typeset_refs_last_of_type:` right before `\__zrefclever_get_ref_first:`, which is the main consumer of the variables it sets, though not the only one (and hence this cannot be moved into `\__zrefclever_get_ref_first:` itself). It also expects a number of relevant variables to have been appropriately set, and which it uses, prominently `\l__zrefclever_type_first_label_type_tl`, but also the queue itself in `\l__zrefclever_typeset_queue_curr_tl`, which should be “ready except for the first label”, and the type counter `\l__zrefclever_type_count_int`.

```

2296 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2297 {
2298   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2299   { \tl_clear:N \l__zrefclever_type_name_tl }
2300   {
2301     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2302     { \tl_clear:N \l__zrefclever_type_name_tl }
2303     {
2304       % Determine whether we should use capitalization, abbreviation,
2305       % and plural.
2306       \bool_lazy_or:nnTF
2307       { \l__zrefclever_capitalize_bool }
2308       {
2309         \l__zrefclever_capitalize_first_bool &&
2310         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2311       }
2312       { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2313       { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2314       % If the queue is empty, we have a singular, otherwise, plural.
2315       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl

```

```

2316 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2317 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2318 \bool_lazy_and:nnTF
2319 { \l__zrefclever_abbrev_bool }
2320 {
2321   ! \int_compare_p:nNn
2322     { \l__zrefclever_type_count_int } = { 0 } ||
2323   ! \l__zrefclever_noabbrev_first_bool
2324 }
2325 {
2326   \tl_set:NV \l__zrefclever_name_format_fallback_tl
2327     \l__zrefclever_name_format_tl
2328   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2329 }
2330 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2331
2332 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2333 {
2334   \prop_get:cVNF
2335   {
2336     l__zrefclever_type_
2337     \l__zrefclever_type_first_label_type_tl _options_prop
2338   }
2339   \l__zrefclever_name_format_tl
2340   \l__zrefclever_type_name_tl
2341   {
2342     \__zrefclever_get_type_transl:xxxNF
2343     { \l__zrefclever_ref_language_tl }
2344     { \l__zrefclever_type_first_label_type_tl }
2345     { \l__zrefclever_name_format_tl }
2346     \l__zrefclever_type_name_tl
2347     {
2348       \tl_clear:N \l__zrefclever_type_name_tl
2349       \msg_warning:nxx { zref-clever } { missing-name }
2350       { \l__zrefclever_type_first_label_type_tl }
2351     }
2352   }
2353 }
2354 {
2355   \prop_get:cVNF
2356   {
2357     l__zrefclever_type_
2358     \l__zrefclever_type_first_label_type_tl _options_prop
2359   }
2360   \l__zrefclever_name_format_tl
2361   \l__zrefclever_type_name_tl
2362   {
2363     \prop_get:cVNF
2364     {
2365       l__zrefclever_type_
2366       \l__zrefclever_type_first_label_type_tl _options_prop
2367     }
2368     \l__zrefclever_name_format_fallback_tl
2369     \l__zrefclever_type_name_tl

```

```

2370 {
2371     \_zrefclever_get_type_transl:xxxNF
2372     { \l__zrefclever_ref_language_tl }
2373     { \l__zrefclever_type_first_label_type_tl }
2374     { \l__zrefclever_name_format_tl }
2375     \l__zrefclever_type_name_tl
2376     {
2377         \_zrefclever_get_type_transl:xxxNF
2378         { \l__zrefclever_ref_language_tl }
2379         { \l__zrefclever_type_first_label_type_tl }
2380         { \l__zrefclever_name_format_fallback_tl }
2381         \l__zrefclever_type_name_tl
2382         {
2383             \tl_clear:N \l__zrefclever_type_name_tl
2384             \msg_warning:nmx { zref-clever }
2385             { missing-name }
2386             { \l__zrefclever_type_first_label_type_tl }
2387         }
2388     }
2389 }
2390 }
2391 }
2392 }
2393 }
2394
2395 % Signal whether the type name is to be included in the hyperlink or not.
2396 \bool_lazy_any:nTF
2397 {
2398     { ! \l__zrefclever_use_hyperref_bool }
2399     { \l__zrefclever_link_star_bool }
2400     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2401     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2402 }
2403 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2404 {
2405     \bool_lazy_any:nTF
2406     {
2407         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2408         {
2409             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2410             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2411         }
2412         {
2413             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2414             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2415             \l__zrefclever_typeset_last_bool &&
2416             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2417         }
2418     }
2419     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2420     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2421 }
2422 }

```

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



`\_zrefclever_labels_in_sequence:nn` Auxiliary function to `\_zrefclever_typeset_refs_not_last_of_type:`. Sets `\l\_zrefclever_next_maybe_range_bool` to true if  $\langle label\ b \rangle$  comes in immediate sequence from  $\langle label\ a \rangle$ . And sets both `\l\_zrefclever_next_maybe_range_bool` and `\l\_zrefclever_next_is_same_bool` to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside `\_zrefclever_typeset_refs_not_last_of_type:`, so this function is expected to be called at its beginning, if compression is enabled.

```

\__zrefclever_labels_in_sequence:nn {\langle label\ a \rangle} {\langle label\ b \rangle}

2423 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2424 {
2425   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2426   {
2427     \exp_args:Nxx \tl_if_eq:nnT
2428     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2429     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2430     {
2431       \int_compare:nNnTF
2432       { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2433       =
2434       { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2435       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2436       {
2437         \int_compare:nNnT
2438         { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2439         =
2440         { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2441         {
2442           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2443           \bool_set_true:N \l__zrefclever_next_is_same_bool
2444         }
2445       }
2446     }
2447   }
2448   {
2449     \exp_args:Nxx \tl_if_eq:nnT
2450     { \zref@extractdefault {#1} { zc@counter } { } }
2451     { \zref@extractdefault {#2} { zc@counter } { } }
2452     {
2453       \exp_args:Nxx \tl_if_eq:nnT
2454       { \zref@extractdefault {#1} { zc@enclval } { } }
2455       { \zref@extractdefault {#2} { zc@enclval } { } }
2456       {
2457         \int_compare:nNnTF
2458         { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2459         =
2460         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2461         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2462         {
2463           \int_compare:nNnT
2464           { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2465           =
2466           { \zref@extractdefault {#2} { zc@cntval } { -1 } }

```

```

2467         {
2468             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2469             \bool_set_true:N \l__zrefclever_next_is_same_bool
2470         }
2471     }
2472 }
2473 }
2474 }
2475 }

```

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

Finally, a couple of functions for retrieving options values, according to the relevant precedence rules. They both receive an *⟨option⟩* as argument, and store the retrieved value in *⟨tl variable⟩*. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of `\l__zrefclever_label_type_a_tl`, as set during the processing of the label stack. This could be easily generalized, of course, but I don't think it is worth it, `\l__zrefclever_label_type_a_tl` is indeed what we want in all practical cases. The difference between `\__zrefclever_get_ref_string:nN` and `\__zrefclever_get_ref_font:nN` is the kind of option each should be used for. `\__zrefclever_get_ref_string:nN` is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fall-back”). `\__zrefclever_get_ref_font:nN` is intended for “font” options, which cannot be “language-specific”, thus for these we just search general options and type options.

```

\__zrefclever_get_ref_string:nN      \__zrefclever_get_ref_string:nN {⟨option⟩} {⟨tl variable⟩}
2476 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2477 {
2478     % First attempt: general options.
2479     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2480     {
2481         % If not found, try type specific options.
2482         \bool_lazy_all:nTF
2483         {
2484             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2485             {
2486                 \prop_if_exist_p:c
2487                 {
2488                     l__zrefclever_type_
2489                     \l__zrefclever_label_type_a_tl _options_prop
2490                 }
2491             }
2492             {
2493                 \prop_if_in_p:cn
2494                 {
2495                     l__zrefclever_type_
2496                     \l__zrefclever_label_type_a_tl _options_prop
2497                 }
2498                 {#1}
2499             }
2500         }
2501         {
2502             \prop_get:cnN
2503             {

```

```

2504         l__zrefclever_type_
2505         \l__zrefclever_label_type_a_tl _options_prop
2506     }
2507     {#1} #2
2508 }
2509 {
2510     % If not found, try type specific translations.
2511     \__zrefclever_get_type_transl:xxnNF
2512     { \l__zrefclever_ref_language_tl }
2513     { \l__zrefclever_label_type_a_tl }
2514     {#1} #2
2515     {
2516         % If not found, try default translations.
2517         \__zrefclever_get_default_transl:xxnNF
2518         { \l__zrefclever_ref_language_tl }
2519         {#1} #2
2520         {
2521             % If not found, try fallback.
2522             \__zrefclever_get_fallback_transl:nNF {#1} #2
2523             {
2524                 \tl_clear:N #2
2525                 \msg_warning:nnn { zref-clever }
2526                 { missing-string } {#1}
2527             }
2528         }
2529     }
2530 }
2531 }
2532 }

```

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

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2533 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2534 {
2535     % First attempt: general options.
2536     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2537     {
2538         % If not found, try type specific options.
2539         \bool_lazy_and:nnTF
2540         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2541         {
2542             \prop_if_exist_p:c
2543             {
2544                 l__zrefclever_type_
2545                 \l__zrefclever_label_type_a_tl _options_prop
2546             }
2547         }
2548         {
2549             \prop_get:cnNF
2550             {
2551                 l__zrefclever_type_
2552                 \l__zrefclever_label_type_a_tl _options_prop
2553             }

```

```

2554         {#1} #2
2555         { \tl_clear:N #2 }
2556     }
2557     { \tl_clear:N #2 }
2558 }
2559 }

```

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

## 9 Compatibility

This section is meant to aggregate any “special handling” needed for L<sup>A</sup>T<sub>E</sub>X kernel features, document classes, and packages, needed for `zref-clever` to work properly with them. It is not meant to be a “kitchen sink of workarounds”. Rather, I intend to keep this as lean as possible, trying to add things selectively when they are safe and reasonable. And, hopefully, doing so by proper setting of `zref-clever`’s options, not by messing with other packages’ code. In particular, I do not mean to compensate for “lack of support for `zref`” by individual packages here, unless there is really no alternative.

### 9.1 Appendix

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

The standard `\appendix` command is a one way switch, in other words, it cannot be reverted (see <https://tex.stackexchange.com/a/444057>). So, even if the fact that it is a “switch” rather than an environment complicates things, because we have to make ungrouped settings to correspond to its effects, in practice this is not a big deal, since these settings are never really reverted (by default, at least). Hence, hooking into `\appendix` is a viable and natural alternative. The `appendix` package defines the `appendices` environment, which provides for a way for the appendix to “end”, but in this case, of course, we can hook into the environment instead.

```

2560 \AddToHook { begindocument }
2561 {
2562     \AddToHook { cmd / appendix / before }
2563     {
2564         \zcsetup
2565         {
2566             countertype =
2567             {
2568                 chapter      = appendix ,
2569                 section      = appendix ,
2570                 subsection   = appendix ,
2571                 subsubsection = appendix ,
2572             }
2573         }
2574     }

```

```

2574     }
2575   }
2576   % \begin{macrocode}
2577   %
2578   %
2579   %
2580   % \subsection{\pkg{listings} package}
2581   %
2582   %
2583   % \begin{macrocode}
2584   \AddToHook { begindocument }
2585   {
2586     \@ifpackageloaded { listings }
2587     {
2588       \zcsetup
2589       {
2590         countertype =
2591         {
2592           lstlisting = listing ,
2593           lstnumber = line ,
2594         } ,
2595         counterresetby = { lstnumber = lstlisting } ,
2596       }
2597       \lst@AddToHook { Init }
2598       {

```

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

```

2599         \tl_if_empty:NF \lst@label
2600         { \zlabel { \lst@label } }

```

The correct place to set `currentcounter` to `lstnumber` is indeed the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` in the same hook. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Note that `listings` *does use* `\refstepcounter{lstnumber}`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. Indeed, the fact that `listings` manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```

2601         \zcsetup { currentcounter = lstnumber }
2602     }
2603   }
2604   {}
2605 }

```

## 9.2 enumitem package

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

```

2606 </package>

```

## 10 Dictionaries

### 10.1 English

```
2607 <package>\zcDeclareLanguage { english }
2608 <package>\zcDeclareLanguageAlias { american } { english }
2609 <package>\zcDeclareLanguageAlias { australian } { english }
2610 <package>\zcDeclareLanguageAlias { british } { english }
2611 <package>\zcDeclareLanguageAlias { canadian } { english }
2612 <package>\zcDeclareLanguageAlias { newzealand } { english }
2613 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2614 <package>\zcDeclareLanguageAlias { USenglish } { english }
2615 <*dict-english>

2616 namesep = {\nobreakspace} ,
2617 pairsep = {\and\nobreakspace} ,
2618 listsep = {,~} ,
2619 lastsep = {\and\nobreakspace} ,
2620 tpairsep = {\and\nobreakspace} ,
2621 tlistsep = {,~} ,
2622 tlastsep = {,~\and\nobreakspace} ,
2623 notesep = {~} ,
2624 rangesep = {\to\nobreakspace} ,
2625
2626 type = part ,
2627   Name-sg = Part ,
2628   name-sg = part ,
2629   Name-pl = Parts ,
2630   name-pl = parts ,
2631
2632 type = chapter ,
2633   Name-sg = Chapter ,
2634   name-sg = chapter ,
2635   Name-pl = Chapters ,
2636   name-pl = chapters ,
2637
2638 type = section ,
2639   Name-sg = Section ,
2640   name-sg = section ,
2641   Name-pl = Sections ,
2642   name-pl = sections ,
2643
2644 type = paragraph ,
2645   Name-sg = Paragraph ,
2646   name-sg = paragraph ,
2647   Name-pl = Paragraphs ,
2648   name-pl = paragraphs ,
2649   Name-sg-ab = Par. ,
2650   name-sg-ab = par. ,
2651   Name-pl-ab = Par. ,
2652   name-pl-ab = par. ,
2653
2654 type = appendix ,
2655   Name-sg = Appendix ,
2656   name-sg = appendix ,
```

```

2657     Name-pl = Appendices ,
2658     name-pl = appendices ,
2659
2660     type = page ,
2661     Name-sg = Page ,
2662     name-sg = page ,
2663     Name-pl = Pages ,
2664     name-pl = pages ,
2665     name-sg-ab = p. ,
2666     name-pl-ab = pp. ,
2667
2668     type = line ,
2669     Name-sg = Line ,
2670     name-sg = line ,
2671     Name-pl = Lines ,
2672     name-pl = lines ,
2673
2674     type = figure ,
2675     Name-sg = Figure ,
2676     name-sg = figure ,
2677     Name-pl = Figures ,
2678     name-pl = figures ,
2679     Name-sg-ab = Fig. ,
2680     name-sg-ab = fig. ,
2681     Name-pl-ab = Figs. ,
2682     name-pl-ab = figs. ,
2683
2684     type = table ,
2685     Name-sg = Table ,
2686     name-sg = table ,
2687     Name-pl = Tables ,
2688     name-pl = tables ,
2689
2690     type = item ,
2691     Name-sg = Item ,
2692     name-sg = item ,
2693     Name-pl = Items ,
2694     name-pl = items ,
2695
2696     type = footnote ,
2697     Name-sg = Footnote ,
2698     name-sg = footnote ,
2699     Name-pl = Footnotes ,
2700     name-pl = footnotes ,
2701
2702     type = note ,
2703     Name-sg = Note ,
2704     name-sg = note ,
2705     Name-pl = Notes ,
2706     name-pl = notes ,
2707
2708     type = equation ,
2709     Name-sg = Equation ,
2710     name-sg = equation ,

```

```

2711 Name-pl = Equations ,
2712 name-pl = equations ,
2713 Name-sg-ab = Eq. ,
2714 name-sg-ab = eq. ,
2715 Name-pl-ab = Eqs. ,
2716 name-pl-ab = eqs. ,
2717 refpre-in = {()} ,
2718 refpos-in = {} ,
2719
2720 type = theorem ,
2721 Name-sg = Theorem ,
2722 name-sg = theorem ,
2723 Name-pl = Theorems ,
2724 name-pl = theorems ,
2725
2726 type = lemma ,
2727 Name-sg = Lemma ,
2728 name-sg = lemma ,
2729 Name-pl = Lemmas ,
2730 name-pl = lemmas ,
2731
2732 type = corollary ,
2733 Name-sg = Corollary ,
2734 name-sg = corollary ,
2735 Name-pl = Corollaries ,
2736 name-pl = corollaries ,
2737
2738 type = proposition ,
2739 Name-sg = Proposition ,
2740 name-sg = proposition ,
2741 Name-pl = Propositions ,
2742 name-pl = propositions ,
2743
2744 type = definition ,
2745 Name-sg = Definition ,
2746 name-sg = definition ,
2747 Name-pl = Definitions ,
2748 name-pl = definitions ,
2749
2750 type = proof ,
2751 Name-sg = Proof ,
2752 name-sg = proof ,
2753 Name-pl = Proofs ,
2754 name-pl = proofs ,
2755
2756 type = result ,
2757 Name-sg = Result ,
2758 name-sg = result ,
2759 Name-pl = Results ,
2760 name-pl = results ,
2761
2762 type = remark ,
2763 Name-sg = Remark ,
2764 name-sg = remark ,

```



```

2765   Name-pl = Remarks ,
2766   name-pl = remarks ,
2767
2768   type = example ,
2769   Name-sg = Example ,
2770   name-sg = example ,
2771   Name-pl = Examples ,
2772   name-pl = examples ,
2773
2774   type = algorithm ,
2775   Name-sg = Algorithm ,
2776   name-sg = algorithm ,
2777   Name-pl = Algorithms ,
2778   name-pl = algorithms ,
2779
2780   type = listing ,
2781   Name-sg = Listing ,
2782   name-sg = listing ,
2783   Name-pl = Listings ,
2784   name-pl = listings ,
2785
2786   type = exercise ,
2787   Name-sg = Exercise ,
2788   name-sg = exercise ,
2789   Name-pl = Exercises ,
2790   name-pl = exercises ,
2791
2792   type = solution ,
2793   Name-sg = Solution ,
2794   name-sg = solution ,
2795   Name-pl = Solutions ,
2796   name-pl = solutions ,
2797   </dict-english>

```

## 10.2 German

```

2798 <package>\zcDeclareLanguage { german }
2799 <package>\zcDeclareLanguageAlias { austrian      } { german }
2800 <package>\zcDeclareLanguageAlias { germanb       } { german }
2801 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2802 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2803 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2804 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2805 <*dict-german>

2806 namesep = {\nobreakspace} ,
2807 pairsep  = {\simund\nobreakspace} ,
2808 listsep  = {,~} ,
2809 lastsep  = {\simund\nobreakspace} ,
2810 tpairsep = {\simund\nobreakspace} ,
2811 tlistsep = {,~} ,
2812 tlastsep = {\simund\nobreakspace} ,
2813 notesep  = {~} ,
2814 rangesep = {\simbis\nobreakspace} ,
2815

```

```

2816 type = part ,
2817     Name-sg = Teil ,
2818     name-sg = Teil ,
2819     Name-pl = Teile ,
2820     name-pl = Teile ,
2821
2822 type = chapter ,
2823     Name-sg = Kapitel ,
2824     name-sg = Kapitel ,
2825     Name-pl = Kapitel ,
2826     name-pl = Kapitel ,
2827
2828 type = section ,
2829     Name-sg = Abschnitt ,
2830     name-sg = Abschnitt ,
2831     Name-pl = Abschnitte ,
2832     name-pl = Abschnitte ,
2833
2834 type = paragraph ,
2835     Name-sg = Absatz ,
2836     name-sg = Absatz ,
2837     Name-pl = Absätze ,
2838     name-pl = Absätze ,
2839
2840 type = appendix ,
2841     Name-sg = Anhang ,
2842     name-sg = Anhang ,
2843     Name-pl = Anhänge ,
2844     name-pl = Anhänge ,
2845
2846 type = page ,
2847     Name-sg = Seite ,
2848     name-sg = Seite ,
2849     Name-pl = Seiten ,
2850     name-pl = Seiten ,
2851
2852 type = line ,
2853     Name-sg = Zeile ,
2854     name-sg = Zeile ,
2855     Name-pl = Zeilen ,
2856     name-pl = Zeilen ,
2857
2858 type = figure ,
2859     Name-sg = Abbildung ,
2860     name-sg = Abbildung ,
2861     Name-pl = Abbildungen ,
2862     name-pl = Abbildungen ,
2863     Name-sg-ab = Abb. ,
2864     name-sg-ab = Abb. ,
2865     Name-pl-ab = Abb. ,
2866     name-pl-ab = Abb. ,
2867
2868 type = table ,
2869     Name-sg = Tabelle ,

```

```

2870     name-sg = Tabelle ,
2871     Name-pl = Tabellen ,
2872     name-pl = Tabellen ,
2873
2874 type = item ,
2875     Name-sg = Punkt ,
2876     name-sg = Punkt ,
2877     Name-pl = Punkte ,
2878     name-pl = Punkte ,
2879
2880 type = footnote ,
2881     Name-sg = Fußnote ,
2882     name-sg = Fußnote ,
2883     Name-pl = Fußnoten ,
2884     name-pl = Fußnoten ,
2885
2886 type = note ,
2887     Name-sg = Anmerkung ,
2888     name-sg = Anmerkung ,
2889     Name-pl = Anmerkungen ,
2890     name-pl = Anmerkungen ,
2891
2892 type = equation ,
2893     Name-sg = Gleichung ,
2894     name-sg = Gleichung ,
2895     Name-pl = Gleichungen ,
2896     name-pl = Gleichungen ,
2897     refpre-in = {() ,
2898     refpos-in = {} } ,
2899
2900 type = theorem ,
2901     Name-sg = Theorem ,
2902     name-sg = Theorem ,
2903     Name-pl = Theoreme ,
2904     name-pl = Theoreme ,
2905
2906 type = lemma ,
2907     Name-sg = Lemma ,
2908     name-sg = Lemma ,
2909     Name-pl = Lemmata ,
2910     name-pl = Lemmata ,
2911
2912 type = corollary ,
2913     Name-sg = Korollar ,
2914     name-sg = Korollar ,
2915     Name-pl = Korollare ,
2916     name-pl = Korollare ,
2917
2918 type = proposition ,
2919     Name-sg = Satz ,
2920     name-sg = Satz ,
2921     Name-pl = Sätze ,
2922     name-pl = Sätze ,
2923

```

```

2924 type = definition ,
2925     Name-sg = Definition ,
2926     name-sg = Definition ,
2927     Name-pl = Definitionen ,
2928     name-pl = Definitionen ,
2929
2930 type = proof ,
2931     Name-sg = Beweis ,
2932     name-sg = Beweis ,
2933     Name-pl = Beweise ,
2934     name-pl = Beweise ,
2935
2936 type = result ,
2937     Name-sg = Ergebnis ,
2938     name-sg = Ergebnis ,
2939     Name-pl = Ergebnisse ,
2940     name-pl = Ergebnisse ,
2941
2942 type = remark ,
2943     Name-sg = Bemerkung ,
2944     name-sg = Bemerkung ,
2945     Name-pl = Bemerkungen ,
2946     name-pl = Bemerkungen ,
2947
2948 type = example ,
2949     Name-sg = Beispiel ,
2950     name-sg = Beispiel ,
2951     Name-pl = Beispiele ,
2952     name-pl = Beispiele ,
2953
2954 type = algorithm ,
2955     Name-sg = Algorithmus ,
2956     name-sg = Algorithmus ,
2957     Name-pl = Algorithmen ,
2958     name-pl = Algorithmen ,
2959
2960 type = listing ,
2961     Name-sg = Listing ,
2962     name-sg = Listing ,
2963     Name-pl = Listings ,
2964     name-pl = Listings ,
2965
2966 type = exercise ,
2967     Name-sg = Übungsaufgabe ,
2968     name-sg = Übungsaufgabe ,
2969     Name-pl = Übungsaufgaben ,
2970     name-pl = Übungsaufgaben ,
2971
2972 type = solution ,
2973     Name-sg = Lösung ,
2974     name-sg = Lösung ,
2975     Name-pl = Lösungen ,
2976     name-pl = Lösungen ,
2977 </dict-german>

```

## 10.3 French

```
2978 <package>\zcDeclareLanguage { french }
2979 <package>\zcDeclareLanguageAlias { acadian } { french }
2980 <package>\zcDeclareLanguageAlias { canadien } { french }
2981 <package>\zcDeclareLanguageAlias { francais } { french }
2982 <package>\zcDeclareLanguageAlias { frenchb } { french }
2983 <*dict-french>

2984 namesep = {\nobreakspace} ,
2985 pairsep = {\~et\nobreakspace} ,
2986 listsep = {\~,~} ,
2987 lastsep = {\~et\nobreakspace} ,
2988 tpairsep = {\~et\nobreakspace} ,
2989 tlistsep = {\~,~} ,
2990 tlastsep = {\~et\nobreakspace} ,
2991 notesep = {\~} ,
2992 rangesep = {\~à\nobreakspace} ,
2993
2994 type = part ,
2995   Name-sg = Partie ,
2996   name-sg = partie ,
2997   Name-pl = Parties ,
2998   name-pl = parties ,
2999
3000 type = chapter ,
3001   Name-sg = Chapitre ,
3002   name-sg = chapitre ,
3003   Name-pl = Chapitres ,
3004   name-pl = chapitres ,
3005
3006 type = section ,
3007   Name-sg = Section ,
3008   name-sg = section ,
3009   Name-pl = Sections ,
3010   name-pl = sections ,
3011
3012 type = paragraph ,
3013   Name-sg = Paragraphe ,
3014   name-sg = paragraphe ,
3015   Name-pl = Paragraphes ,
3016   name-pl = paragraphes ,
3017
3018 type = appendix ,
3019   Name-sg = Annexe ,
3020   name-sg = annexe ,
3021   Name-pl = Annexes ,
3022   name-pl = annexes ,
3023
3024 type = page ,
3025   Name-sg = Page ,
3026   name-sg = page ,
3027   Name-pl = Pages ,
3028   name-pl = pages ,
3029
```

```

3030 type = line ,
3031     Name-sg = Ligne ,
3032     name-sg = ligne ,
3033     Name-pl = Lignes ,
3034     name-pl = lignes ,
3035
3036 type = figure ,
3037     Name-sg = Figure ,
3038     name-sg = figure ,
3039     Name-pl = Figures ,
3040     name-pl = figures ,
3041
3042 type = table ,
3043     Name-sg = Table ,
3044     name-sg = table ,
3045     Name-pl = Tables ,
3046     name-pl = tables ,
3047
3048 type = item ,
3049     Name-sg = Point ,
3050     name-sg = point ,
3051     Name-pl = Points ,
3052     name-pl = points ,
3053
3054 type = footnote ,
3055     Name-sg = Note ,
3056     name-sg = note ,
3057     Name-pl = Notes ,
3058     name-pl = notes ,
3059
3060 type = note ,
3061     Name-sg = Note ,
3062     name-sg = note ,
3063     Name-pl = Notes ,
3064     name-pl = notes ,
3065
3066 type = equation ,
3067     Name-sg = Équation ,
3068     name-sg = équation ,
3069     Name-pl = Équations ,
3070     name-pl = équations ,
3071     refpre-in = {()} ,
3072     refpos-in = {} } ,
3073
3074 type = theorem ,
3075     Name-sg = Théorème ,
3076     name-sg = théorème ,
3077     Name-pl = Théorèmes ,
3078     name-pl = théorèmes ,
3079
3080 type = lemma ,
3081     Name-sg = Lemme ,
3082     name-sg = lemme ,
3083     Name-pl = Lemmes ,

```

```

3084     name-pl = lemmes ,
3085
3086 type = corollary ,
3087     Name-sg = Corollaire ,
3088     name-sg = corollaire ,
3089     Name-pl = Corollaires ,
3090     name-pl = corollaires ,
3091
3092 type = proposition ,
3093     Name-sg = Proposition ,
3094     name-sg = proposition ,
3095     Name-pl = Propositions ,
3096     name-pl = propositions ,
3097
3098 type = definition ,
3099     Name-sg = Définition ,
3100     name-sg = définition ,
3101     Name-pl = Définitions ,
3102     name-pl = définitions ,
3103
3104 type = proof ,
3105     Name-sg = Démonstration ,
3106     name-sg = démonstration ,
3107     Name-pl = Démonstrations ,
3108     name-pl = démonstrations ,
3109
3110 type = result ,
3111     Name-sg = Résultat ,
3112     name-sg = résultat ,
3113     Name-pl = Résultats ,
3114     name-pl = résultats ,
3115
3116 type = remark ,
3117     Name-sg = Remarque ,
3118     name-sg = remarque ,
3119     Name-pl = Remarques ,
3120     name-pl = remarques ,
3121
3122 type = example ,
3123     Name-sg = Exemple ,
3124     name-sg = exemple ,
3125     Name-pl = Exemples ,
3126     name-pl = exemples ,
3127
3128 type = algorithm ,
3129     Name-sg = Algorithme ,
3130     name-sg = algorithme ,
3131     Name-pl = Algorithmes ,
3132     name-pl = algorithmes ,
3133
3134 type = listing ,
3135     Name-sg = Liste ,
3136     name-sg = liste ,
3137     Name-pl = Listes ,

```

```

3138     name-pl = listes ,
3139
3140 type = exercise ,
3141     Name-sg = Exercice ,
3142     name-sg = exercice ,
3143     Name-pl = Exercices ,
3144     name-pl = exercices ,
3145
3146 type = solution ,
3147     Name-sg = Solution ,
3148     name-sg = solution ,
3149     Name-pl = Solutions ,
3150     name-pl = solutions ,
3151 </dict-french>

```

## 10.4 Portuguese

```

3152 <package>\zcDeclareLanguage { portuguese }
3153 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3154 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
3155 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3156 <*dict-portuguese>

3157 namesep = {\nobreakspace} ,
3158 pairsep = {\nobreakspace} ,
3159 listsep = {,~} ,
3160 lastsep = {\nobreakspace} ,
3161 tpairsep = {\nobreakspace} ,
3162 tlistsep = {,~} ,
3163 tlastsep = {\nobreakspace} ,
3164 notesep = {~} ,
3165 rangesep = {\nobreakspace} ,
3166
3167 type = part ,
3168     Name-sg = Parte ,
3169     name-sg = parte ,
3170     Name-pl = Partes ,
3171     name-pl = partes ,
3172
3173 type = chapter ,
3174     Name-sg = Capítulo ,
3175     name-sg = capítulo ,
3176     Name-pl = Capítulos ,
3177     name-pl = capítulos ,
3178
3179 type = section ,
3180     Name-sg = Seção ,
3181     name-sg = seção ,
3182     Name-pl = Seções ,
3183     name-pl = seções ,
3184
3185 type = paragraph ,
3186     Name-sg = Parágrafo ,
3187     name-sg = parágrafo ,
3188     Name-pl = Parágrafos ,

```



```

3189     name-pl = parágrafos ,
3190     Name-sg-ab = Par. ,
3191     name-sg-ab = par. ,
3192     Name-pl-ab = Par. ,
3193     name-pl-ab = par. ,
3194
3195     type = appendix ,
3196     Name-sg = Apêndice ,
3197     name-sg = apêndice ,
3198     Name-pl = Apêndices ,
3199     name-pl = apêndices ,
3200
3201     type = page ,
3202     Name-sg = Página ,
3203     name-sg = página ,
3204     Name-pl = Páginas ,
3205     name-pl = páginas ,
3206     name-sg-ab = p. ,
3207     name-pl-ab = pp. ,
3208
3209     type = line ,
3210     Name-sg = Linha ,
3211     name-sg = linha ,
3212     Name-pl = Linhas ,
3213     name-pl = linhas ,
3214
3215     type = figure ,
3216     Name-sg = Figura ,
3217     name-sg = figura ,
3218     Name-pl = Figuras ,
3219     name-pl = figuras ,
3220     Name-sg-ab = Fig. ,
3221     name-sg-ab = fig. ,
3222     Name-pl-ab = Figs. ,
3223     name-pl-ab = figs. ,
3224
3225     type = table ,
3226     Name-sg = Tabela ,
3227     name-sg = tabela ,
3228     Name-pl = Tabelas ,
3229     name-pl = tabelas ,
3230
3231     type = item ,
3232     Name-sg = Item ,
3233     name-sg = item ,
3234     Name-pl = Itens ,
3235     name-pl = itens ,
3236
3237     type = footnote ,
3238     Name-sg = Nota ,
3239     name-sg = nota ,
3240     Name-pl = Notas ,
3241     name-pl = notas ,
3242

```

```

3243 type = note ,
3244     Name-sg = Nota ,
3245     name-sg = nota ,
3246     Name-pl = Notas ,
3247     name-pl = notas ,
3248
3249 type = equation ,
3250     Name-sg = Equação ,
3251     name-sg = equação ,
3252     Name-pl = Equações ,
3253     name-pl = equações ,
3254     Name-sg-ab = Eq. ,
3255     name-sg-ab = eq. ,
3256     Name-pl-ab = Eqs. ,
3257     name-pl-ab = eqs. ,
3258     refpre-in = {()} ,
3259     refpos-in = {} ,
3260
3261 type = theorem ,
3262     Name-sg = Teorema ,
3263     name-sg = teorema ,
3264     Name-pl = Teoremas ,
3265     name-pl = teoremas ,
3266
3267 type = lemma ,
3268     Name-sg = Lema ,
3269     name-sg = lema ,
3270     Name-pl = Lemas ,
3271     name-pl = lemas ,
3272
3273 type = corollary ,
3274     Name-sg = Corolário ,
3275     name-sg = corolário ,
3276     Name-pl = Corolários ,
3277     name-pl = corolários ,
3278
3279 type = proposition ,
3280     Name-sg = Proposição ,
3281     name-sg = proposição ,
3282     Name-pl = Proposições ,
3283     name-pl = proposições ,
3284
3285 type = definition ,
3286     Name-sg = Definição ,
3287     name-sg = definição ,
3288     Name-pl = Definições ,
3289     name-pl = definições ,
3290
3291 type = proof ,
3292     Name-sg = Demonstração ,
3293     name-sg = demonstração ,
3294     Name-pl = Demonstrações ,
3295     name-pl = demonstrações ,
3296

```

```

3297 type = result ,
3298   Name-sg = Resultado ,
3299   name-sg = resultado ,
3300   Name-pl = Resultados ,
3301   name-pl = resultados ,
3302
3303 type = remark ,
3304   Name-sg = Observação ,
3305   name-sg = observação ,
3306   Name-pl = Observações ,
3307   name-pl = observações ,
3308
3309 type = example ,
3310   Name-sg = Exemplo ,
3311   name-sg = exemplo ,
3312   Name-pl = Exemplos ,
3313   name-pl = exemplos ,
3314
3315 type = algorithm ,
3316   Name-sg = Algoritmo ,
3317   name-sg = algoritmo ,
3318   Name-pl = Algoritmos ,
3319   name-pl = algoritmos ,
3320
3321 type = listing ,
3322   Name-sg = Listagem ,
3323   name-sg = listagem ,
3324   Name-pl = Listagens ,
3325   name-pl = listagens ,
3326
3327 type = exercise ,
3328   Name-sg = Exercício ,
3329   name-sg = exercício ,
3330   Name-pl = Exercícios ,
3331   name-pl = exercícios ,
3332
3333 type = solution ,
3334   Name-sg = Solução ,
3335   name-sg = solução ,
3336   Name-pl = Soluções ,
3337   name-pl = soluções ,
3338 </dict-portuguese>

```

## 10.5 Spanish

```

3339 <package>\zcDeclareLanguage { spanish }
3340 <*dict-spanish>
3341 namesep = {\nobreakspace} ,
3342 pairsep = {\sim\nobreakspace} ,
3343 listsep = {,~} ,
3344 lastsep = {\sim\nobreakspace} ,
3345 tpairsep = {\sim\nobreakspace} ,
3346 tlistsep = {,~} ,
3347 tlastsep = {\sim\nobreakspace} ,

```

```

3348 notesep = {~} ,
3349 rangesep = {~a\nobreakspace} ,
3350
3351 type = part ,
3352   Name-sg = Parte ,
3353   name-sg = parte ,
3354   Name-pl = Partes ,
3355   name-pl = partes ,
3356
3357 type = chapter ,
3358   Name-sg = Capítulo ,
3359   name-sg = capítulo ,
3360   Name-pl = Capítulos ,
3361   name-pl = capítulos ,
3362
3363 type = section ,
3364   Name-sg = Sección ,
3365   name-sg = sección ,
3366   Name-pl = Secciones ,
3367   name-pl = secciones ,
3368
3369 type = paragraph ,
3370   Name-sg = Párrafo ,
3371   name-sg = párrafo ,
3372   Name-pl = Párrafos ,
3373   name-pl = párrafos ,
3374
3375 type = appendix ,
3376   Name-sg = Apéndice ,
3377   name-sg = apéndice ,
3378   Name-pl = Apéndices ,
3379   name-pl = apéndices ,
3380
3381 type = page ,
3382   Name-sg = Página ,
3383   name-sg = página ,
3384   Name-pl = Páginas ,
3385   name-pl = páginas ,
3386
3387 type = line ,
3388   Name-sg = Línea ,
3389   name-sg = línea ,
3390   Name-pl = Líneas ,
3391   name-pl = líneas ,
3392
3393 type = figure ,
3394   Name-sg = Figura ,
3395   name-sg = figura ,
3396   Name-pl = Figuras ,
3397   name-pl = figuras ,
3398
3399 type = table ,
3400   Name-sg = Cuadro ,
3401   name-sg = cuadro ,

```

```

3402     Name-pl = Cuadros ,
3403     name-pl = cuadros ,
3404
3405     type = item ,
3406     Name-sg = Punto ,
3407     name-sg = punto ,
3408     Name-pl = Puntos ,
3409     name-pl = puntos ,
3410
3411     type = footnote ,
3412     Name-sg = Nota ,
3413     name-sg = nota ,
3414     Name-pl = Notas ,
3415     name-pl = notas ,
3416
3417     type = note ,
3418     Name-sg = Nota ,
3419     name-sg = nota ,
3420     Name-pl = Notas ,
3421     name-pl = notas ,
3422
3423     type = equation ,
3424     Name-sg = Ecuación ,
3425     name-sg = ecuación ,
3426     Name-pl = Ecuaciones ,
3427     name-pl = ecuaciones ,
3428     refpre-in = {(} ,
3429     refpos-in = {)} ,
3430
3431     type = theorem ,
3432     Name-sg = Teorema ,
3433     name-sg = teorema ,
3434     Name-pl = Teoremas ,
3435     name-pl = teoremas ,
3436
3437     type = lemma ,
3438     Name-sg = Lema ,
3439     name-sg = lema ,
3440     Name-pl = Lemas ,
3441     name-pl = lemas ,
3442
3443     type = corollary ,
3444     Name-sg = Corolario ,
3445     name-sg = corolario ,
3446     Name-pl = Corolarios ,
3447     name-pl = corolarios ,
3448
3449     type = proposition ,
3450     Name-sg = Proposición ,
3451     name-sg = proposición ,
3452     Name-pl = Proposiciones ,
3453     name-pl = proposiciones ,
3454
3455     type = definition ,

```

```

3456     Name-sg = Definición ,
3457     name-sg = definición ,
3458     Name-pl = Definiciones ,
3459     name-pl = definiciones ,
3460
3461 type = proof ,
3462     Name-sg = Demostración ,
3463     name-sg = demostración ,
3464     Name-pl = Demostraciones ,
3465     name-pl = demostraciones ,
3466
3467 type = result ,
3468     Name-sg = Resultado ,
3469     name-sg = resultado ,
3470     Name-pl = Resultados ,
3471     name-pl = resultados ,
3472
3473 type = remark ,
3474     Name-sg = Observación ,
3475     name-sg = observación ,
3476     Name-pl = Observaciones ,
3477     name-pl = observaciones ,
3478
3479 type = example ,
3480     Name-sg = Ejemplo ,
3481     name-sg = ejemplo ,
3482     Name-pl = Ejemplos ,
3483     name-pl = ejemplos ,
3484
3485 type = algorithm ,
3486     Name-sg = Algoritmo ,
3487     name-sg = algoritmo ,
3488     Name-pl = Algoritmos ,
3489     name-pl = algoritmos ,
3490
3491 type = listing ,
3492     Name-sg = Listado ,
3493     name-sg = listado ,
3494     Name-pl = Listados ,
3495     name-pl = listados ,
3496
3497 type = exercise ,
3498     Name-sg = Ejercicio ,
3499     name-sg = ejercicio ,
3500     Name-pl = Ejercicios ,
3501     name-pl = ejercicios ,
3502
3503 type = solution ,
3504     Name-sg = Solución ,
3505     name-sg = solución ,
3506     Name-pl = Soluciones ,
3507     name-pl = soluciones ,
3508 </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	C
\\ ..... 107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155	clist commands:
† internal commands:	\clist_map_inline:nn ..... 842
\i_zrefclever_current_counter_tl ..... 4	\counterwithin ..... 4
	cs commands:
	\cs_generate_variant:Nn ..... 59, 60, 314, 322, 985, 991, 1234, 2143
<b>A</b>	\cs_if_exist:NTF ..... 43, 52, 73
\AddToHook .... 95, 466, 481, 625, 661, 686, 724, 726, 786, 2560, 2562, 2584	\cs_new:Npn 41, 50, 61, 71, 82, 2094, 2144
\appendix ..... 68	\cs_new_protected:Npn ..... ..... 262, 315, 323, 329, 450, 980, 986, 1069, 1122, 1164, 1175, 1235, 1408, 1460, 1504, 1666, 1934, 2090, 2092, 2296, 2423, 2476, 2533
\appendixname ..... 68	\cs_new_protected:Npx ..... 94
	\cs_set_eq:NN ..... 98
<b>B</b>	
\babelname ..... 671	<b>E</b>
\babelprovide ..... 11, 22	\endinput ..... 12
\begin ..... 2576, 2583	exp commands:
bool commands:	\exp_args:NNe ..... 27, 30
\bool_case_true: ..... 2	\exp_args:NNnx ..... 252
\bool_if:NTF ..... 298, 309, 629, 633, 1314, 1356, 1564, 1659, 1789, 1811, 1842, 1899, 1940, 1963, 1967, 1973, 1983, 1989, 2149	\exp_args:NnV ..... 290
\bool_if:nTF . 63, 1182, 1191, 1200, 1257, 1285, 1323, 1430, 1438, 1578, 1586, 1823, 1830, 1837, 2098, 2233	\exp_args:NNx ..... 99
\bool_lazy_all:nTF ..... 2482	\exp_args:Nnx ..... 325, 1295, 1333
\bool_lazy_and:nnTF ..... ..... 1076, 1094, 2318, 2539	\exp_args:Nxx ..... .. 1218, 1265, 1369, 2427, 2449, 2453
\bool_lazy_any:nTF ..... 2396, 2405	\exp_not:N ..... 58, 1844, 1847, 1869, 1872, 1875, 2104, 2107, 2110, 2123, 2125, 2128, 2131, 2136, 2138, 2156, 2177, 2180, 2182, 2185, 2192, 2199, 2201, 2205, 2208, 2211, 2223, 2226, 2239, 2242, 2245, 2272, 2274, 2277, 2280, 2287, 2289
\bool_lazy_or:nnTF ..... 1080, 2306	\exp_not:n . 1688, 1704, 1716, 1721, 1744, 1758, 1762, 1774, 1778, 1812, 1813, 1845, 1868, 1873, 1874, 2003, 2016, 2023, 2047, 2059, 2063, 2073, 2077, 2105, 2106, 2108, 2118, 2121, 2124, 2129, 2130, 2132, 2133, 2135, 2137, 2178, 2179, 2181, 2183, 2184, 2186, 2187, 2191, 2203, 2204, 2209, 2210, 2212, 2220, 2224, 2225, 2227, 2240, 2241, 2243, 2266, 2270, 2273, 2278, 2279, 2281, 2282, 2286, 2288
\bool_new:N ..... 261, 502, 503, 528, 552, 561, 568, 569, 582, 583, 602, 603, 779, 780, 1105, 1118, 1470, 1471, 1481, 1487, 1488	\ExplSyntaxOn ..... 12, 274
\bool_set:Nn ..... 1074	
\bool_set_false:N ..... ..... 515, 519, 610, 619, 620, 635, 801, 1254, 1527, 1570, 1584, 1598, 1801, 1938, 1939, 2403, 2420	
\bool_set_true:N ..... . 318, 509, 510, 514, 520, 609, 614, 615, 790, 795, 1269, 1280, 1310, 1318, 1351, 1360, 1388, 1400, 1535, 1565, 1571, 1575, 1602, 1608, 2419, 2435, 2442, 2443, 2461, 2468, 2469	
\bool_until_do:Nn ..... 1255, 1528	

<b>F</b>		<code>\keys_set:nn</code> ..... 12, 30, 34, 291, 796, 919, 925, 974, 1072
file commands:		keyval commands:
<code>\file_get:nnNTF</code> ..... 272		<code>\keyval_parse:nnn</code> ..... 814, 868
<code>\fmtversion</code> ..... 3		
<b>G</b>		<b>L</b>
group commands:		<code>\labelformat</code> ..... 3
<code>\group_begin:</code> .. 97, 264, 317, 969, 1071, 1084, 1844, 1872, 2104, 2107, 2128, 2131, 2177, 2182, 2185, 2201, 2208, 2223, 2239, 2242, 2277, 2280		<code>\language name</code> ..... 22, 665
<code>\group_end:</code> .... 100, 312, 320, 977, 1087, 1102, 1869, 1875, 2123, 2125, 2136, 2138, 2180, 2192, 2199, 2205, 2211, 2226, 2272, 2274, 2287, 2289		
<b>I</b>		<b>M</b>
<code>\IfBooleanTF</code> ..... 1108		<code>\mainbabelname</code> ..... 22, 672
<code>\IfFormatAtLeastTF</code> ..... 3, 4		<code>\MessageBreak</code> ..... 10
<code>\input</code> ..... 11		msg commands:
int commands:		<code>\msg_info:nnn</code> ..... 355, 385
<code>\int_case:nnTF</code> ..... .. 1669, 1697, 1729, 1902, 1996, 2035		<code>\msg_line_context:</code> ..... ..... 106, 112, 116, 133, 140, 143, 149, 163, 167, 169, 171, 174, 178
<code>\int_compare:nNnTF</code> 1222, 1270, 1301, 1338, 1373, 1389, 1416, 1418, 1462, 1630, 1684, 1718, 1891, 1893, 1951, 1976, 2020, 2431, 2437, 2457, 2463		<code>\msg_new:nnn</code> 104, 110, 115, 117, 119, 125, 131, 137, 139, 141, 147, 152, 157, 159, 161, 166, 168, 170, 172, 177
<code>\int_compare_p:nNn</code> ..... ..... 1432, 1440, 2310, 2321, 2416		<code>\msg_note:nnn</code> ..... 294
<code>\int_eval:n</code> ..... 94		<code>\msg_warning:nn</code> ..... ..... 471, 496, 634, 640, 784, 805
<code>\int_incr:N</code> 1294, 1332, 1929, 1966, 1968, 1982, 1984, 1988, 1990, 2088		<code>\msg_warning:nnn</code> ... 241, 256, 300, 310, 712, 757, 870, 934, 976, 1016, 1055, 1623, 1796, 2349, 2384, 2525
<code>\int_new:N</code> ..... .. 1119, 1120, 1472, 1473, 1484, 1485		<code>\msg_warning:nnnn</code> ..... ..... 816, 1278, 1316, 1358, 1398
<code>\int_set:Nn</code> ... 1417, 1419, 1423, 1426		
<code>\int_show:N</code> ..... 1300		<b>N</b>
<code>\int_use:N</code> ..... 37, 39, 54		<code>\newcounter</code> ..... 4
<code>\int_zero:N</code> ..... 1291, 1329, 1410, 1411, 1513, 1514, 1515, 1516, 1928, 1930, 1931, 2083, 2084		<code>\NewDocumentCommand</code> ..... .. 236, 246, 918, 920, 967, 1067, 1106
<code>\l_tmpa_int</code> ..... 1329, 1332, 1342		<code>\nobreakspace</code> ..... 399, 2616, 2617, 2619, 2620, 2622, 2624, 2806, 2807, 2809, 2810, 2812, 2814, 2984, 2985, 2987, 2988, 2990, 2992, 3157, 3158, 3160, 3161, 3163, 3165, 3341, 3342, 3344, 3345, 3347, 3349
<code>\l_tmpb_int</code> ... 1291, 1294, 1300, 1306		
iow commands:		<b>P</b>
<code>\iow_char:N</code> ..... 107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155		<code>\PackageError</code> ..... 7
<b>K</b>		<code>\pagenumbering</code> ..... 6
keys commands:		<code>\pageref</code> ..... 35
<code>\keys_define:nn</code> .... 30, 335, 347, 364, 378, 457, 485, 492, 504, 529, 538, 553, 562, 570, 584, 596, 604, 637, 644, 682, 729, 771, 774, 781, 791, 802, 810, 838, 864, 888, 898, 909, 930, 942, 992, 1004, 1025, 1048		<code>\pkg</code> ..... 2580
		prg commands:
		<code>\prg_generate_conditional_- variant:Nnn</code> ..... 423, 439
		<code>\prg_new_protected_conditional:Npnn</code> ..... 409, 425, 442
		<code>\prg_return_false:</code> ..... ..... 419, 421, 435, 437, 448
		<code>\prg_return_true:</code> ..... 418, 434, 447
		<code>\ProcessKeysOptions</code> ..... 917





<code>\zref@newprop</code> .....	341, 461,
5, 6, 20, 22, 25, 36, 39, 87, 89, 102	
<code>\zref@refused</code> .....	1617
<code>\zref@wrapper@babel</code> .....	33, 1068
<code>\textendash</code> .....	403
<code>\the</code> .....	3
<code>\thechapter</code> .....	68
<code>\thelstnumber</code> .....	69
<code>\thepage</code> .....	6, 99
<code>\thesection</code> .....	68
tl commands:	
<code>\c_empty_tl</code> .....	1167, 1178, 1180,
1238, 1242, 1246, 1250, 1551, 1556	
<code>\c_novalue_tl</code> .....	900, 944
<code>\tl_clear:N</code> 289, 340, 973, 997, 1508,	
1509, 1510, 1511, 1512, 1534, 1924,	
1925, 1926, 1927, 1965, 2299, 2302,	
2330, 2348, 2383, 2524, 2555, 2557	
<code>\tl_gset:Nn</code> .....	99
<code>\tl_head:N</code> .....	
1370, 1371, 1374, 1376, 1390, 1392	
<code>\tl_if_empty:NTF</code> .....	75, 352,
369, 383, 1009, 1030, 1053, 1088,	
1621, 1791, 2217, 2315, 2332, 2599	
<code>\tl_if_empty:nTF</code> .....	
238, 248, 339, 452, 996, 1740, 1756,	
1772, 2014, 2045, 2057, 2071, 2301	
<code>\tl_if_empty_p:N</code> 1186, 1187, 1195,	
1196, 1203, 1204, 1581, 1582, 1589,	
1591, 2400, 2410, 2414, 2484, 2540	
<code>\tl_if_empty_p:n</code> 1261, 1262, 1288, 1326	
<code>\tl_if_eq:NNTF</code> .....	1207, 1594
<code>\tl_if_eq:NnTF</code> .....	1125, 1157,
1422, 1425, 1450, 1453, 1542, 2425	
<code>\tl_if_eq:nnTF</code> .. 1218, 1265, 1295,	
1333, 1369, 1414, 2427, 2449, 2453	
<code>\tl_if_novalue:NTF</code> .....	903, 947
<code>\tl_item:Nn</code> .....	1305, 1340
<code>\tl_map_break:n</code> .....	85, 1298, 1336
<code>\tl_map_inline:Nn</code> .....	1292, 1330
<code>\tl_map_tokens:Nn</code> .....	77
<code>\tl_new:N</code> .....	93, 179, 180, 456,
658, 659, 660, 770, 773, 887, 1112,	
1113, 1114, 1115, 1116, 1117, 1474,	
1475, 1476, 1477, 1478, 1479, 1480,	
1482, 1483, 1486, 1489, 1490, 1491,	
1492, 1493, 1494, 1495, 1496, 1497,	
1498, 1499, 1500, 1501, 1502, 1503	
<code>\tl_put_left:Nn</code> .... 1826, 1833, 1884	
<code>\tl_put_right:Nn</code> .... 1686, 1702,	
1711, 1742, 1753, 1769, 2001, 2012,	
2043, 2055, 2069, 2316, 2317, 2328	
<code>\tl_reverse_items:n</code> .....	
1234, 1240, 1244, 1248, 1252	
<code>\tl_set:Nn</code> .....	341, 461,
463, 469, 472, 488, 497, 665, 666,	
671, 672, 675, 676, 679, 692, 700,	
709, 714, 737, 745, 754, 759, 924,	
998, 1166, 1177, 1179, 1237, 1239,	
1241, 1243, 1245, 1247, 1249, 1251,	
1378, 1380, 1382, 1384, 1544, 1545,	
1548, 1553, 1675, 1677, 1809, 1840,	
1955, 1957, 1980, 2312, 2313, 2326	
<code>\tl_set_eq:NN</code> .....	1922
<code>\tl_tail:N</code> .... 1379, 1381, 1383, 1385	
<code>\l_tmpa_tl</code> .....	275, 291, 1090, 1091
U	
use commands:	
<code>\use:N</code> .....	23
Z	
<code>\zcDeclareLanguage</code> .....	
10, 236, 2607, 2798, 2978, 3152, 3339	
<code>\zcDeclareLanguageAlias</code> .....	
10, 11, 246, 2608, 2609,	
2610, 2611, 2612, 2613, 2614, 2799,	
2800, 2801, 2802, 2803, 2804, 2979,	
2980, 2981, 2982, 3153, 3154, 3155	
<code>\zcLanguageSetup</code> 9, 11–13, 29, 31, 32, 967	
<code>\zcpageref</code> .....	35, 1106
<code>\zceref</code> .....	24, 25, 28,
29, 33, 35–37, 45, 46, 1067, 1109, 1110	
<code>\zcRefTypeSetup</code> .....	8, 29, 30, 920
<code>\zcsetup</code> 22, 25, 28, 29, 918, 2564, 2588, 2601	
<code>\zlabel</code> .....	69, 2600
zrefcheck commands:	
<code>\zrefcheck_zceref_beg_label:</code> .. 1079	
<code>\zrefcheck_zceref_end_label_-</code>	
maybe: .....	1098
<code>\zrefcheck_zceref_run_checks_on_-</code>	
labels:n .....	1099
zrefclever internal commands:	
<code>\l_zrefclever_abbrev_bool</code> .....	
582, 586, 2319	

`\l_zrefclever_counter_resettters_`  
`seq` . . . [4](#), [5](#), [26](#), [27](#), [67](#), [837](#), [844](#), [847](#)  
`\l_zrefclever_counter_type_prop`  
. . . . . [3](#), [26](#), [27](#), [30](#), [809](#), [821](#)  
`\l_zrefclever_current_counter_`  
`tl` . . . . . [3](#), [28](#),  
[20](#), [23](#), [28](#), [31](#), [33](#), [37](#), [88](#), [90](#), [887](#), [890](#)  
`\l_zrefclever_current_language_`  
`tl` . . . [22](#), [660](#), [665](#), [671](#), [675](#), [701](#), [746](#)  
`\_zrefclever_declare_default_`  
`transl:nnn` . . . [31](#), [980](#), [1011](#), [1032](#)  
`\_zrefclever_declare_type_`  
`transl:nnnn` . . . [31](#), [980](#), [1037](#), [1059](#)  
`\g_zrefclever_dict_⟨language⟩_prop`  
. . . . . [12](#)  
`\l_zrefclever_dict_language_tl` .  
. [179](#), [266](#), [270](#), [273](#), [280](#), [286](#), [293](#),  
[295](#), [301](#), [304](#), [326](#), [332](#), [413](#), [416](#),  
[429](#), [432](#), [971](#), [1012](#), [1033](#), [1038](#), [1060](#)  
`\g_zrefclever_fallback_dict_`  
`prop` . . . . . [8](#), [392](#), [393](#), [445](#)  
`\_zrefclever_get_default_`  
`transl:nnN` . . . . . [9](#), [426](#), [440](#)  
`\_zrefclever_get_default_`  
`transl:nnNTF` . . . . . [16](#), [425](#), [2517](#)  
`\_zrefclever_get_enclosing_`  
`counters:n` . . . . . [5](#), [41](#), [46](#), [88](#)  
`\_zrefclever_get_enclosing_`  
`counters_value:n` . . . [5](#), [41](#), [55](#), [90](#)  
`\_zrefclever_get_fallback_`  
`transl:nN` . . . . . [443](#)  
`\_zrefclever_get_fallback_`  
`transl:nNTF` . . . . . [17](#), [441](#), [2522](#)  
`\_zrefclever_get_ref:n` . . . . .  
. . . . . [57](#), [58](#), [1689](#), [1705](#),  
[1717](#), [1722](#), [1745](#), [1759](#), [1763](#), [1775](#),  
[1779](#), [1814](#), [1834](#), [2004](#), [2017](#), [2024](#),  
[2048](#), [2060](#), [2064](#), [2074](#), [2078](#), [2094](#)  
`\_zrefclever_get_ref_first:` . . .  
. . . . . [57](#), [58](#), [62](#), [1827](#), [1885](#), [2144](#)  
`\_zrefclever_get_ref_font:nN` . [8](#),  
[15](#), [28](#), [66](#), [67](#), [1650](#), [1652](#), [1654](#), [2533](#)  
`\_zrefclever_get_ref_string:nN` .  
. . . . . [8](#), [9](#), [15](#), [28](#), [66](#), [1090](#), [1519](#),  
[1521](#), [1523](#), [1632](#), [1634](#), [1636](#), [1638](#),  
[1640](#), [1642](#), [1644](#), [1646](#), [1648](#), [2476](#)  
`\_zrefclever_get_type_transl:nnnN`  
. . . . . [9](#), [410](#), [424](#)  
`\_zrefclever_get_type_transl:nnnNTF`  
. . . . . [16](#), [409](#), [2342](#), [2371](#), [2377](#), [2511](#)  
`\l_zrefclever_label_a_tl` . . . . .  
. [44](#), [1474](#), [1531](#), [1551](#), [1567](#), [1617](#),  
[1618](#), [1624](#), [1676](#), [1689](#), [1705](#), [1722](#),  
[1763](#), [1779](#), [1807](#), [1814](#), [1942](#), [1946](#),  
[1956](#), [1981](#), [2004](#), [2025](#), [2064](#), [2078](#)  
`\l_zrefclever_label_b_tl` . . . . .  
. . . . . [44](#), [1474](#),  
[1534](#), [1539](#), [1556](#), [1569](#), [1574](#), [1946](#)  
`\l_zrefclever_label_count_int` . .  
. . . . . [44](#), [1472](#),  
[1513](#), [1630](#), [1669](#), [1928](#), [1951](#), [2088](#)  
`\l_zrefclever_label_enclcnt_a_`  
`tl` . . . . . [1112](#), [1237](#), [1239](#), [1240](#),  
[1261](#), [1288](#), [1330](#), [1370](#), [1378](#), [1379](#)  
`\l_zrefclever_label_enclcnt_b_`  
`tl` . . . . . [1112](#), [1241](#), [1243](#), [1244](#),  
[1262](#), [1292](#), [1326](#), [1371](#), [1380](#), [1381](#)  
`\l_zrefclever_label_enclval_a_`  
`tl` . . . . . [1112](#), [1245](#), [1247](#),  
[1248](#), [1341](#), [1374](#), [1382](#), [1383](#), [1390](#)  
`\l_zrefclever_label_enclval_b_`  
`tl` . . . . . [1112](#), [1249](#), [1251](#),  
[1252](#), [1305](#), [1376](#), [1384](#), [1385](#), [1392](#)  
`\l_zrefclever_label_type_a_tl` . .  
. . . . . [66](#), [1112](#), [1166](#), [1169](#),  
[1172](#), [1177](#), [1186](#), [1195](#), [1203](#), [1208](#),  
[1422](#), [1450](#), [1544](#), [1548](#), [1581](#), [1589](#),  
[1595](#), [1621](#), [1678](#), [1958](#), [2484](#), [2489](#),  
[2496](#), [2505](#), [2513](#), [2540](#), [2545](#), [2552](#)  
`\l_zrefclever_label_type_b_tl` . .  
. . . . . [1112](#),  
[1179](#), [1187](#), [1196](#), [1204](#), [1209](#), [1425](#),  
[1453](#), [1545](#), [1553](#), [1582](#), [1591](#), [1596](#)  
`\_zrefclever_label_type_put_`  
`new_right:n` . . . . . [36](#), [37](#), [1128](#), [1164](#)  
`\l_zrefclever_label_types_seq` . .  
. . . . . [37](#), [1121](#), [1124](#), [1168](#), [1171](#), [1448](#)  
`\_zrefclever_labels_in_sequence:nn`  
. . . . . [45](#), [65](#), [1805](#), [1945](#), [2423](#)  
`\g_zrefclever_languages_prop` . . .  
. . . . . [10](#), [235](#), [240](#), [242](#), [250](#),  
[253](#), [254](#), [265](#), [412](#), [428](#), [707](#), [752](#), [970](#)  
`\l_zrefclever_last_of_type_bool`  
. . . . . [44](#), [1469](#), [1565](#), [1570](#), [1571](#),  
[1575](#), [1584](#), [1599](#), [1603](#), [1609](#), [1659](#)  
`\l_zrefclever_lastsep_tl` . . [1489](#),  
[1641](#), [1704](#), [1721](#), [1744](#), [1762](#), [1774](#)  
`\l_zrefclever_link_star_bool` . . .  
. . . . . [1074](#), [1104](#), [2101](#), [2236](#), [2399](#)  
`\l_zrefclever_listsep_tl` . . . . .  
. . . . . [1489](#), [1639](#), [1716](#), [1758](#), [2003](#),  
[2016](#), [2023](#), [2047](#), [2059](#), [2063](#), [2073](#)  
`\l_zrefclever_load_dict_`  
`verbose_bool` . . . . . [261](#), [298](#), [309](#), [318](#)  
`\g_zrefclever_loaded_dictionaries_`  
`seq` . . . . . [260](#), [269](#), [292](#), [303](#)

\l_zrefclever_main_language_tl .	1484, 1515, 1697, 1731, 1930, 1966,
..... 22, 659,	1977, 1982, 1988, 1996, 2037, 2083
666, 672, 676, 680, 693, 715, 738, 760	
\_zrefclever_name_default: ....	\l_zrefclever_range_same_count_-
..... 2090, 2219	int ..... 44,
\l_zrefclever_name_format_-	1484, 1516, 1684, 1719, 1732, 1931,
fallback_tl ..... 2090, 2219	1968, 1984, 1990, 2021, 2038, 2084
.. 1480, 2326, 2330, 2332, 2368, 2380	\l_zrefclever_rangesep_tl .....
\l_zrefclever_name_format_tl ...	..... 1489, 1635, 1778, 1813, 2077
... 1480, 2312, 2313, 2316, 2317,	\_zrefclever_ref_default: .....
2327, 2328, 2339, 2345, 2360, 2374	..... 2090, 2141, 2147, 2213, 2292
\l_zrefclever_name_in_link_bool	\l_zrefclever_ref_language_tl ..
..... 59,	..... 22, 23, 658, 679,
62, 1480, 1842, 2149, 2403, 2419, 2420	692, 695, 700, 703, 709, 714, 718,
\l_zrefclever_namefont_tl 1489,	728, 737, 740, 745, 748, 754, 759,
1651, 1845, 1873, 2178, 2209, 2224	763, 1075, 2343, 2372, 2378, 2512, 2518
\l_zrefclever_nameinlink_str ...	\c_zrefclever_ref_options_font_-
..... 643, 648,	seq ..... 9, 15, 181
650, 652, 654, 2401, 2407, 2409, 2413	\c_zrefclever_ref_options_-
\l_zrefclever_namesep_tl .....	necessarily_not_type_specific_-
.. 1489, 1633, 2181, 2212, 2220, 2227	seq ..... 15, 181, 345, 928, 1002
\l_zrefclever_next_is_same_bool	\c_zrefclever_ref_options_-
..... 44, 65, 1484,	necessarily_type_specific_seq
1939, 1967, 1983, 1989, 2443, 2469	..... 181, 376, 1046
\l_zrefclever_next_maybe_range_-	\c_zrefclever_ref_options_-
bool ..... 44, 65, 1484, 1801, 1811, 1938,	possibly_type_specific_seq ..
1963, 1973, 2435, 2442, 2461, 2468	..... 15, 181, 362, 1023
\l_zrefclever_noabbrev_first_-	\l_zrefclever_ref_options_prop ..
bool ..... 583, 592, 2323	.... 28, 30, 894, 904, 905, 2479, 2536
\_zrefclever_page_format_aux: ..	\c_zrefclever_ref_options_-
..... 94, 98	reference_seq ..... 181, 896
\g_zrefclever_page_format_tl ...	\c_zrefclever_ref_options_-
..... 6, 93, 99, 102	typesetup_seq ..... 181, 940
\l_zrefclever_pairsep_tl .....	\l_zrefclever_ref_property_tl ..
..... 1489, 1637, 1688, 1812	..... 17, 18,
\_zrefclever_prop_put_non_-	456, 461, 463, 469, 472, 488, 497,
empty:Nnn ..... 17, 450, 820, 874	1125, 1157, 1542, 2096, 2120, 2134,
\_zrefclever_provide_dict_-	2153, 2190, 2231, 2269, 2285, 2425
default_transl:nn 14, 323, 353, 370	\l_zrefclever_ref_typeset_font_-
\_zrefclever_provide_dict_type_-	tl ..... 770, 772, 1085
transl:nn ..... 14, 323, 371, 388	\l_zrefclever_reffont_in_tl 1489,
\_zrefclever_provide_dictionary:n	1655, 2108, 2132, 2186, 2243, 2281
..... 8, 12-14,	\l_zrefclever_reffont_out_tl ...
34, 262, 319, 728, 739, 747, 762, 1075	..... 1489, 1653,
\_zrefclever_provide_dictionary_-	2105, 2129, 2183, 2203, 2240, 2278
verbose:n ... 13, 315, 694, 702, 717	\l_zrefclever_refpos_in_tl 1489,
\l_zrefclever_range_beg_label_-	1649, 2121, 2135, 2191, 2270, 2286
tl ..... 44, 1484, 1512,	\l_zrefclever_refpos_out_tl 1489,
1717, 1740, 1746, 1756, 1760, 1772,	1645, 2124, 2137, 2204, 2273, 2288
1776, 1927, 1965, 1980, 2014, 2018,	\l_zrefclever_refpre_in_tl 1489,
2045, 2049, 2057, 2061, 2071, 2075	1647, 2118, 2133, 2187, 2266, 2282
\l_zrefclever_range_count_int ..	\l_zrefclever_refpre_out_tl 1489,
..... 44,	1643, 2106, 2130, 2184, 2241, 2279
	\l_zrefclever_setup_type_tl ...
	..... 14, 179, 289, 327, 340, 341,

352, 369, 383, 924, 952, 960, 973,  
 997, 998, 1009, 1030, 1039, 1053, 1061  
 \l\_zrefclever\_sort\_decided\_bool  
 ..... 1118,  
 1254, 1255, 1269, 1280, 1310, 1314,  
 1318, 1352, 1356, 1360, 1388, 1400  
 \\_zrefclever\_sort\_default:nn ...  
 ..... 37, 1159, 1175  
 \\_zrefclever\_sort\_default\_-  
 different\_types:nn .....  
 ..... 19, 35, 36, 42, 1213, 1408  
 \\_zrefclever\_sort\_default\_same\_-  
 type:nn ..... 35, 38, 1211, 1235  
 \\_zrefclever\_sort\_labels: .....  
 ..... 36, 37, 43, 1083, 1122  
 \\_zrefclever\_sort\_page:nn .....  
 ..... 43, 1158, 1460  
 \l\_zrefclever\_sort\_prior\_a\_int .  
 ..... 1119,  
 1410, 1416, 1417, 1423, 1433, 1441  
 \l\_zrefclever\_sort\_prior\_b\_int .  
 ..... 1119,  
 1411, 1418, 1419, 1426, 1434, 1442  
 \l\_zrefclever\_tlastsep\_tl .....  
 ..... 1489, 1524, 1916  
 \l\_zrefclever\_tlistsep\_tl .....  
 ..... 1489, 1522, 1894  
 \l\_zrefclever\_tpairsep\_tl .....  
 ..... 1489, 1520, 1910  
 \l\_zrefclever\_type\_<type>-  
 options\_prop ..... 30  
 \l\_zrefclever\_type\_count\_int ...  
 ..... 44, 62, 1472, 1514, 1891,  
 1893, 1902, 1929, 2310, 2322, 2416  
 \l\_zrefclever\_type\_first\_label\_-  
 tl 44, 59, 1474, 1510, 1675, 1793,  
 1802, 1806, 1834, 1850, 1854, 1859,  
 1865, 1925, 1955, 2146, 2152, 2159,  
 2162, 2167, 2173, 2189, 2230, 2248,  
 2251, 2256, 2262, 2268, 2284, 2298  
 \l\_zrefclever\_type\_first\_label\_-  
 type\_tl 44, 62, 1474, 1511, 1677,  
 1797, 1926, 1957, 2301, 2337, 2344,  
 2350, 2358, 2366, 2373, 2379, 2386  
 \\_zrefclever\_type\_name\_setup: ..  
 ..... 8, 9, 59, 1822, 2296  
 \l\_zrefclever\_type\_name\_tl ....  
 ..... 59, 62,  
 1480, 1868, 1874, 2179, 2210, 2217,  
 2225, 2299, 2302, 2340, 2346, 2348,  
 2361, 2369, 2375, 2381, 2383, 2400  
 \l\_zrefclever\_typeset\_compress\_-  
 bool ..... 552, 555, 1940  
 \l\_zrefclever\_typeset\_labels\_-  
 seq 43, 1469, 1506, 1530, 1532, 1538  
 \l\_zrefclever\_typeset\_last\_bool  
 ..... 44, 1469,  
 1527, 1528, 1535, 1564, 1899, 2415  
 \l\_zrefclever\_typeset\_name\_bool  
 ..... 503, 510, 515, 520, 1824, 1838  
 \l\_zrefclever\_typeset\_queue\_-  
 curr\_tl ..... 44,  
 57, 62, 1474, 1509, 1686, 1702,  
 1711, 1742, 1753, 1769, 1791,  
 1809, 1826, 1833, 1840, 1884, 1906,  
 1911, 1917, 1923, 1924, 2001, 2012,  
 2043, 2055, 2069, 2315, 2410, 2414  
 \l\_zrefclever\_typeset\_queue\_-  
 prev\_tl . 44, 1474, 1508, 1895, 1922  
 \l\_zrefclever\_typeset\_range\_-  
 bool ..... 561, 564, 1082, 1789  
 \l\_zrefclever\_typeset\_ref\_bool .  
 ..... 502, 509, 514, 519, 1824, 1831  
 \\_zrefclever\_typeset\_refs: .....  
 ..... 43, 45, 46, 1086, 1504  
 \\_zrefclever\_typeset\_refs\_last\_-  
 of\_type: . 49, 57, 59, 62, 1661, 1666  
 \\_zrefclever\_typeset\_refs\_not\_-  
 last\_of\_type: .....  
 ..... 45, 49, 57, 65, 1663, 1934  
 \l\_zrefclever\_typeset\_sort\_bool  
 ..... 528, 531, 1081  
 \l\_zrefclever\_typeset\_sort\_seq ....  
 ..... 19, 42, 537, 542, 543, 549, 1412  
 \l\_zrefclever\_use\_hyperref\_bool  
 ..... 602, 609,  
 614, 619, 629, 635, 2100, 2235, 2398  
 \l\_zrefclever\_warn\_hyperref\_-  
 bool ..... 603, 610, 615, 620, 633  
 \\_zrefclever\_zcref:nnn .. 1068, 1069  
 \\_zrefclever\_zcref:nnnn 33, 36, 1069  
 \l\_zrefclever\_zcref\_labels\_seq .  
 ..... 36,  
 37, 1073, 1100, 1104, 1127, 1130, 1507  
 \l\_zrefclever\_zcref\_note\_tl ...  
 ..... 773, 776, 1088, 1092  
 \l\_zrefclever\_zcref\_with\_check\_-  
 bool ..... 780, 795, 1078, 1096  
 \l\_zrefclever\_zrefcheck\_-  
 available\_bool .....  
 ..... 779, 790, 801, 1077, 1095