

# 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>44</b>
<b>9</b>	<b>Compatibility</b>	<b>69</b>
9.1	\appendix . . . . .	69
9.2	appendix package . . . . .	70
9.3	listings package . . . . .	71
9.4	enumitem package . . . . .	71

---

\*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>72</b>
10.1	English	72
10.2	German	76
10.3	French	79
10.4	Portuguese	83
10.5	Spanish	86
	<b>Index</b>	<b>90</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

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

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

Identify the package.

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

## 2 Dependencies

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

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

### 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”, “super-counter”, “parent counter” etc.). This information is a little trickier to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\c1@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see section ‘ltcounts.dtx’ in ‘source2e’). Besides, there may be a chain of resetting counters, which must be taken into account: if ‘counterC’ gets reset by ‘counterB’, and ‘counterB’ gets reset by ‘counterA’, stepping the latter affects all three of them.

The procedure below examines a set of counters, those included in `\l__zrefclever-counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\c1@<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 `\c1@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually

tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

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`.)

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 { c1@ #2 }
76     {
77       \tl_map_tokens:cn { c1@ #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\zceref'~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:.. }
179   \msg_new:nnn { zref-clever } { compat-package }
180     { Loaded~support~for~'#1'~package. }
181   \msg_new:nnn { zref-clever } { compat-class }
182     { Loaded~support~for~'#1'~documentclass. }

```

## 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`.

```
183 \tl_new:N \l__zrefclever_setup_type_tl
184 \tl_new:N \l__zrefclever_dict_language_tl
```

(End definition for `\l__zrefclever_setup_type_tl` and `\l__zrefclever_dict_language_tl`.)

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

```
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

185 \seq_const_from_clist:Nn
186   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
187   {
188     tpairsep ,
189     tlistsep ,
190     tlastsep ,
191     notesep ,
192   }
193 \seq_const_from_clist:Nn
194   \c__zrefclever_ref_options_possibly_type_specific_seq
195   {
196     namesep ,
197     pairsep ,
198     listsep ,
199     lastsep ,
200     rangesep ,
201     refpre ,
202     refpos ,
203     refpre-in ,
204     refpos-in ,
205   }
```

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:`.

```
206 \seq_const_from_clist:Nn
207   \c__zrefclever_ref_options_necessarily_type_specific_seq
208   {
209     Name-sg ,
210     name-sg ,
211     Name-pl ,
212     name-pl ,
213     Name-sg-ab ,
214     name-sg-ab ,
215     Name-pl-ab ,
216     name-pl-ab ,
217   }
```

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

```

218 \seq_const_from_clist:Nn
219 \c__zrefclever_ref_options_font_seq
220 {
221     namefont ,
222     reffont ,
223     reffont-in ,
224 }
225 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
226 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
227 \c__zrefclever_ref_options_possibly_type_specific_seq
228 \c__zrefclever_ref_options_necessarily_type_specific_seq
229 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
230 \c__zrefclever_ref_options_typesetup_seq
231 \c__zrefclever_ref_options_font_seq
232 \seq_new:N \c__zrefclever_ref_options_reference_seq
233 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
234 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
235 \c__zrefclever_ref_options_possibly_type_specific_seq
236 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
237 \c__zrefclever_ref_options_reference_seq
238 \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 of not a language or alias is known to `zref-clever` is decided by its presence in this property list. A “base language” (loose concept here, meaning just “the name we gave for the dictionary in that particular language”) is just like any other one, the only difference is that the “language name” happens to be the same as the “dictionary name”, in other words, it is an “alias to itself”.

```

239 \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 {\language}}

240 \NewDocumentCommand \zcDeclareLanguage { m }
241 {
242     \tl_if_empty:nF {#1}
243     {
244         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
245         { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
246         { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
247     }
248 }
249 \@onlypreamble \zcDeclareLanguage

```

(End definition for `\zcDeclareLanguage`.)

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

```

\zcDeclareLanguageAlias {\langle language alias \rangle} {\langle aliased language \rangle}

250 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
251 {
252   \tl_if_empty:nF {#1}
253   {
254     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
255     {
256       \exp_args:NnNx
257       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
258       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
259     }
260     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
261   }
262 }
263 \@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 `.lbf` 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.

```
264 \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.

```
265 \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⟩}

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

```

```

284         \l__zrefclever_dict_language_tl _prop
285     }
286     {
287         \prop_new:c
288         {
289             g__zrefclever_dict_
290             \l__zrefclever_dict_language_tl _prop
291         }
292     }
293     \tl_clear:N \l__zrefclever_setup_type_tl
294     \exp_args:NnV
295         \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
296     \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
297         \l__zrefclever_dict_language_tl
298     \msg_note:nnx { zref-clever } { dict-loaded }
299         { \l__zrefclever_dict_language_tl }
300 }
301 {
302     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
303     {
304         \msg_warning:nnx { zref-clever } { dict-not-available }
305         { \l__zrefclever_dict_language_tl }
306     }

```

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.

```

307         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
308         \l__zrefclever_dict_language_tl
309     }
310 }
311 }
312 {
313     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
314     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
315 }
316 \group_end:
317 }
318 \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>}
319 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
320 {
321     \group_begin:
322     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool

```

```

323     \__zrefclever_provide_dictionary:n {#1}
324     \group_end:
325   }
326   \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbos:n { x }

```

(End definition for \\_\_zrefclever\_provide\_dictionary\_verbos:n.)

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

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

```

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

327 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
328 {
329   \exp_args:Nnx \prop_gput_if_new:cnn
330   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
331   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
332 }
333 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
334 {
335   \prop_gput_if_new:cnn
336   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
337   { default- #1 } {#2}
338 }

```

(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.

```

339 \keys_define:nn { zref-clever / dictionary }
340 {
341   type .code:n =
342   {
343     \tl_if_empty:nTF {#1}
344     { \tl_clear:N \l__zrefclever_setup_type_tl }
345     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
346   } ,
347 }
348 \seq_map_inline:Nn
349 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
350 {
351   \keys_define:nn { zref-clever / dictionary }
352   {
353     #1 .value_required:n = true ,
354     #1 .code:n =
355     {

```

```

356         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
357         { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
358         {
359             \msg_info:nnn { zref-clever }
360             { option-not-type-specific } {#1}
361         }
362     } ,
363 }
364 }
365 \seq_map_inline:Nn
366 \c__zrefclever_ref_options_possibly_type_specific_seq
367 {
368     \keys_define:nn { zref-clever / dictionary }
369     {
370         #1 .value_required:n = true ,
371         #1 .code:n =
372         {
373             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
374             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
375             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
376         } ,
377     }
378 }
379 \seq_map_inline:Nn
380 \c__zrefclever_ref_options_necessarily_type_specific_seq
381 {
382     \keys_define:nn { zref-clever / dictionary }
383     {
384         #1 .value_required:n = true ,
385         #1 .code:n =
386         {
387             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
388             {
389                 \msg_info:nnn { zref-clever }
390                 { option-only-type-specific } {#1}
391             }
392             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
393         } ,
394     }
395 }

```

## 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.

```

396 \prop_new:N \g__zrefclever_fallback_dict_prop
397 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
398 {
399     tpairsep = {,~} ,
400     tlistsep = {,~} ,
401     tlastsep = {,~} ,
402     notesep  = {~} ,
403     namesep  = {\nobreakspace} ,
404     pairsep  = {,~} ,
405     listsep  = {,~} ,
406     lastsep  = {,~} ,
407     rangesep = {\textendash} ,
408     refpre   = {} ,
409     refpos   = {} ,
410     refpre-in = {} ,
411     refpos-in = {} ,
412 }

```

## 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>}

413 \prg_new_protected_conditional:Npnn
414 \_zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
415 {
416     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
417     \l__zrefclever_dict_language_tl
418     {
419         \prop_get:cnNTF
420         { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
421         { type- #2 - #3 } #4
422         { \prg_return_true: }
423         { \prg_return_false: }
424     }
425     { \prg_return_false: }
426 }
427 \prg_generate_conditional_variant:Nnn
428 \_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>}

```



```

429 \prg_new_protected_conditional:Npnn
430 \__zrefclever_get_default_transl:nnN #1#2#3 { F }
431 {
432   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
433   \l__zrefclever_dict_language_tl
434   {
435     \prop_get:cnNTF
436     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
437     { default- #2 } #3
438     { \prg_return_true: }
439     { \prg_return_false: }
440   }
441   { \prg_return_false: }
442 }
443 \prg_generate_conditional_variant:Nnn
444 \__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>}

445 % {<key>}<tl var to set>
446 \prg_new_protected_conditional:Npnn
447 \__zrefclever_get_fallback_transl:nN #1#2 { F }
448 {
449   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
450   { #1 } #2
451   { \prg_return_true: }
452   { \prg_return_false: }
453 }

```

(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>}

454 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
455 {
456   \tl_if_empty:nTF {#3}
457   { \prop_remove:Nn #1 {#2} }
458   { \prop_put:Nnn #1 {#2} {#3} }
459 }

```

(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.

```

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

```

```

500             \msg_warning:nn { zref-clever } { missing-zref-titleref }
501             \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
502         }
503     }
504 }
505 }

```

### typeset option

```

506 \bool_new:N \l__zrefclever_typeset_ref_bool
507 \bool_new:N \l__zrefclever_typeset_name_bool
508 \keys_define:nn { zref-clever / reference }
509 {
510     typeset .choice: ,
511     typeset / both .code:n =
512     {
513         \bool_set_true:N \l__zrefclever_typeset_ref_bool
514         \bool_set_true:N \l__zrefclever_typeset_name_bool
515     } ,
516     typeset / ref .code:n =
517     {
518         \bool_set_true:N \l__zrefclever_typeset_ref_bool
519         \bool_set_false:N \l__zrefclever_typeset_name_bool
520     } ,
521     typeset / name .code:n =
522     {
523         \bool_set_false:N \l__zrefclever_typeset_ref_bool
524         \bool_set_true:N \l__zrefclever_typeset_name_bool
525     } ,
526     typeset .initial:n = both ,
527     typeset .value_required:n = true ,
528
529     noname .meta:n = { typeset = ref },
530     noname .value_forbidden:n = true ,
531 }

```

### sort option

```

532 \bool_new:N \l__zrefclever_typeset_sort_bool
533 \keys_define:nn { zref-clever / reference }
534 {
535     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
536     sort .initial:n = true ,
537     sort .default:n = true ,
538     nosort .meta:n = { sort = false },
539     nosort .value_forbidden:n = true ,
540 }

```

### 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.

```

541 \seq_new:N \l__zrefclever_typesort_seq

```

```

542 \keys_define:nn { zref-clever / reference }
543 {
544   typesort .code:n =
545   {
546     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
547     \seq_reverse:N \l__zrefclever_typesort_seq
548   } ,
549   typesort .initial:n =
550   { part , chapter , section , paragraph } ,
551   typesort .value_required:n = true ,
552   notypesort .code:n =
553   { \seq_clear:N \l__zrefclever_typesort_seq } ,
554   notypesort .value_forbidden:n = true ,
555 }

```

#### comp option

```

556 \bool_new:N \l__zrefclever_typeset_compress_bool
557 \keys_define:nn { zref-clever / reference }
558 {
559   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
560   comp .initial:n = true ,
561   comp .default:n = true ,
562   nocomp .meta:n = { comp = false } ,
563   nocomp .value_forbidden:n = true ,
564 }

```

#### range option

```

565 \bool_new:N \l__zrefclever_typeset_range_bool
566 \keys_define:nn { zref-clever / reference }
567 {
568   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
569   range .initial:n = false ,
570   range .default:n = true ,
571 }

```

#### cap and capfirst options

```

572 \bool_new:N \l__zrefclever_capitalize_bool
573 \bool_new:N \l__zrefclever_capitalize_first_bool
574 \keys_define:nn { zref-clever / reference }
575 {
576   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
577   cap .initial:n = false ,
578   cap .default:n = true ,
579   nocap .meta:n = { cap = false } ,
580   nocap .value_forbidden:n = true ,
581
582   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
583   capfirst .initial:n = false ,
584   capfirst .default:n = true ,
585 }

```

#### abbrev and noabbrevfirst options

```

586 \bool_new:N \l__zrefclever_abbrev_bool

```

```

587 \bool_new:N \l__zrefclever_noabbrev_first_bool
588 \keys_define:nn { zref-clever / reference }
589 {
590   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
591   abbrev .initial:n = false ,
592   abbrev .default:n = true ,
593   noabbrev .meta:n = { abbrev = false },
594   noabbrev .value_forbidden:n = true ,
595
596   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
597   noabbrevfirst .initial:n = false ,
598   noabbrevfirst .default:n = true ,
599 }

```

### S option

```

600 \keys_define:nn { zref-clever / reference }
601 {
602   S .meta:n =
603     { capfirst = true , noabbrevfirst = true },
604   S .value_forbidden:n = true ,
605 }

```

### hyperref option

```

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

```

```

637     \bool_if:NT \l__zrefclever_warn_hyperref_bool
638     { \msg_warning:nn { zref-clever } { missing-hyperref } }
639     \bool_set_false:N \l__zrefclever_use_hyperref_bool
640   }
641   \keys_define:nn { zref-clever / reference }
642   {
643     hyperref .code:n =
644     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
645   }
646 }

```

#### nameinlink option

```

647 \str_new:N \l__zrefclever_nameinlink_str
648 \keys_define:nn { zref-clever / reference }
649 {
650   nameinlink .choice: ,
651   nameinlink / true .code:n =
652   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
653   nameinlink / false .code:n =
654   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
655   nameinlink / single .code:n =
656   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
657   nameinlink / tsingle .code:n =
658   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
659   nameinlink .initial:n = tsingle ,
660   nameinlink .default:n = true ,
661 }

```

#### 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`.

```

662 \tl_new:N \l__zrefclever_ref_language_tl
663 \tl_new:N \l__zrefclever_main_language_tl
664 \tl_new:N \l__zrefclever_current_language_tl
665 \AddToHook { begindocument }
666 {
667   \@ifpackageloaded { babel }
668   {
669     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
670     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
671   }
672   {
673     \@ifpackageloaded { polyglossia }
674     {
675       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
676       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
677     }
678     {
679       \tl_set:Nn \l__zrefclever_current_language_tl { english }
680       \tl_set:Nn \l__zrefclever_main_language_tl { english }
681     }
682   }

```

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.

```

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

```

```

707         { \l__zrefclever_ref_language_tl }
708     }
709 }
710 {
711     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
712     {
713         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
714     }
715     {
716         \msg_warning:nnn { zref-clever }
717         { unknown-language-opt } {#1}
718         \tl_set:Nn \l__zrefclever_ref_language_tl
719         { \l__zrefclever_main_language_tl }
720     }
721     \__zrefclever_provide_dictionary_verbose:x
722     { \l__zrefclever_ref_language_tl }
723 }
724 }
725 } ,
726 lang .value_required:n = true ,
727 }
728 \AddToHook { begindocument / before }
729 {
730     \AddToHook { begindocument }
731     {

```

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.

```

732     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

733     \keys_define:nn { zref-clever / reference }
734     {
735         lang .code:n =
736         {
737             \str_case:nnF {#1}
738             {
739                 { main }
740                 {
741                     \tl_set:Nn \l__zrefclever_ref_language_tl
742                     { \l__zrefclever_main_language_tl }
743                     \__zrefclever_provide_dictionary:x
744                     { \l__zrefclever_ref_language_tl }
745                 }
746
747                 { current }
748                 {
749                     \tl_set:Nn \l__zrefclever_ref_language_tl
750                     { \l__zrefclever_current_language_tl }
751                     \__zrefclever_provide_dictionary:x
752                     { \l__zrefclever_ref_language_tl }

```



```

753         }
754     }
755     {
756         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
757         {
758             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
759         }
760         {
761             \msg_warning:nnn { zref-clever }
762             { unknown-language-opt } {#1}
763             \tl_set:Nn \l__zrefclever_ref_language_tl
764             { \l__zrefclever_main_language_tl }
765         }
766         \__zrefclever_provide_dictionary:x
767         { \l__zrefclever_ref_language_tl }
768     }
769     } ,
770     lang .value_required:n = true ,
771 }
772 }
773 }

```

### 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`.

```

774 \tl_new:N \l__zrefclever_ref_typeset_font_tl
775 \keys_define:nn { zref-clever / reference }
776 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

### note option

```

777 \tl_new:N \l__zrefclever_zcref_note_tl
778 \keys_define:nn { zref-clever / reference }
779 {
780     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
781     note .value_required:n = true ,
782 }

```

### check option

Integration with `zref-check`.

```

783 \bool_new:N \l__zrefclever_zrefcheck_available_bool
784 \bool_new:N \l__zrefclever_zcref_with_check_bool
785 \keys_define:nn { zref-clever / reference }
786 {
787     check .code:n =
788     { \msg_warning:nn { zref-clever } { check-document-only } } ,
789 }
790 \AddToHook { begindocument }
791 {
792     \@ifpackageloaded { zref-check }
793     {

```

```

794 \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
795 \keys_define:nn { zref-clever / reference }
796 {
797     check .code:n =
798     {
799         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
800         \keys_set:nn { zref-check / zcheck } {#1}
801     }
802 }
803 }
804 {
805     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
806     \keys_define:nn { zref-clever / reference }
807     {
808         check .code:n =
809         { \msg_warning:nn { zref-clever } { missing-zref-check } }
810     }
811 }
812 }

```

### 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`.

```

813 \prop_new:N \l__zrefclever_counter_type_prop
814 \keys_define:nn { zref-clever / label }
815 {
816     countertype .code:n =
817     {
818         \keyval_parse:nnn
819         {
820             \msg_warning:nnnn { zref-clever }
821             { key-requires-value } { countertype }
822         }
823         {
824             \__zrefclever_prop_put_non_empty:Nnn
825             \l__zrefclever_counter_type_prop
826         }
827         {#1}
828     } ,
829     countertype .value_required:n = true ,
830     countertype .initial:n =
831     {
832         subsection = section ,
833         subsubsection = section ,
834         subparagraph = paragraph ,
835         enumi = item ,
836         enumii = item ,
837         enumiii = item ,
838         enumiv = item ,
839     } ,

```

840 }

### counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `\__zrefclever_counter_reset_by:n` to populate the `zc@enclcnt` and `zc@enclval` properties, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resetters_seq` with the `counterresetby` option.

```

841 \seq_new:N \l__zrefclever_counter_resetters_seq
842 \keys_define:nn { zref-clever / label }
843 {
844   counterresetters .code:n =
845   {
846     \clist_map_inline:nn {#1}
847     {
848       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
849       {
850         \seq_put_right:Nn
851         \l__zrefclever_counter_resetters_seq {##1}
852       }
853     }
854   },
855   counterresetters .initial:n =
856   {
857     part ,
858     chapter ,
859     section ,
860     subsection ,
861     subsubsection ,
862     paragraph ,
863     subparagraph ,
864   },
865   counterresetters .value_required:n = true ,
866 }
```

### counterresetby option

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

```

867 \prop_new:N \l__zrefclever_counter_resetby_prop
868 \keys_define:nn { zref-clever / label }
869 {
870   counterresetby .code:n =
871   {
872     \keyval_parse:nnn
```

```

873         {
874             \msg_warning:nnn { zref-clever }
875             { key-requires-value } { counterresetby }
876         }
877         {
878             \__zrefclever_prop_put_non_empty:Nnn
879             \l__zrefclever_counter_resetby_prop
880         }
881         {#1}
882     } ,
883     counterresetby .value_required:n = true ,
884     counterresetby .initial:n =
885     {

```

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.

```

886         enumii = enumi ,
887         enumiii = enumii ,
888         enumiv = enumiii ,
889     } ,
890 }

```

### 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.

```

891 \tl_new:N \l__zrefclever_current_counter_tl
892 \keys_define:nn { zref-clever / label }
893 {
894     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
895     currentcounter .value_required:n = true ,
896     currentcounter .initial:n = \@currentcounter ,
897 }

```

### Reference options

This is a set of options related to reference typesetting which receive equal treatment and, hence, are handled in batch. Since we are dealing with options to be passed to `\zcref` or to `\zcsetup` or at load time, only “not necessarily type-specific” options are pertinent here. However, they *may* either be type-specific or language-specific, and thus must be stored in a property list, `\l__zrefclever_ref_options_prop`, in order to be retrieved from the option *name* by `\__zrefclever_get_ref_string:nN` and `\__zrefclever_get_ref_font:nN` according to context and precedence rules.

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

```

898 \prop_new:N \l__zrefclever_ref_options_prop
899 \seq_map_inline:Nn

```

```

900 \c__zrefclever_ref_options_reference_seq
901 {
902   \keys_define:nn { zref-clever / reference }
903   {
904     #1 .default:V = \c_novalue_tl ,
905     #1 .code:n =
906     {
907       \tl_if_novalue:nTF {##1}
908       { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
909       { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
910     } ,
911   }
912 }

```

## 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.

```

913 \keys_define:nn { }
914 {
915   zref-clever / zcsetup .inherit:n =
916   {
917     zref-clever / label ,
918     zref-clever / reference ,
919   }
920 }

```

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

```

921 \ProcessKeysOptions { zref-clever / zcsetup }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

922 \NewDocumentCommand \zcsetup { m }
923 { \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>}
924 \NewDocumentCommand \zcRefTypeSetup { m m }
925 {
926   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
927   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
928   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
929   \keys_set:nn { zref-clever / typesetup } {#2}
930 }
```

(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>.

```
931 \seq_map_inline:Nn
932   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
933   {
934     \keys_define:nn { zref-clever / typesetup }
935     {
936       #1 .code:n =
937       {
938         \msg_warning:nnn { zref-clever }
939         { option-not-type-specific } {#1}
940       } ,
941     }
942   }
943 \seq_map_inline:Nn
944   \c__zrefclever_ref_options_typesetup_seq
945   {
946     \keys_define:nn { zref-clever / typesetup }
947     {
948       #1 .default:V = \c_novaluel_tl ,
949       #1 .code:n =
950       {
951         \tl_if_novaluel:NTF {##1}
952         {
953           \prop_remove:cn
954           {
955             l__zrefclever_type_
```

```

956         \l__zrefclever_setup_type_tl _options_prop
957     }
958     {#1}
959 }
960 {
961     \prop_put:cnn
962     {
963         l__zrefclever_type_
964         \l__zrefclever_setup_type_tl _options_prop
965     }
966     {#1} {##1}
967 }
968 } ,
969 }
970 }

```

### 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>}
971 \NewDocumentCommand \zcLanguageSetup { m m }
972 {
973     \group_begin:
974     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
975     \l__zrefclever_dict_language_tl
976     {
977         \tl_clear:N \l__zrefclever_setup_type_tl
978         \keys_set:nn { zref-clever / langsetup } {#2}
979     }
980     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
981     \group_end:
982 }
983 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

```

\__zrefclever_declare_type_transl:nnnn
\__zrefclever_declare_default_transl:nnn

```

A couple of auxiliary functions for the of `zref-clever/translation` keys set in \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>}

```

```

984 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
985 {
986   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
987   { type- #2 - #3 } {#4}
988 }
989 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
990 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
991 {
992   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
993   { default- #2 } {#3}
994 }
995 \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.

```

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

```



```

1033     {
1034         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1035         {
1036             \__zrefclever_declare_default_transl:Vnn
1037             \l__zrefclever_dict_language_tl
1038             {#1} {##1}
1039         }
1040         {
1041             \__zrefclever_declare_type_transl:Vnn
1042             \l__zrefclever_dict_language_tl
1043             \l__zrefclever_setup_type_tl
1044             {#1} {##1}
1045         }
1046     } ,
1047 }
1048 }
1049 \seq_map_inline:Nn
1050 \c__zrefclever_ref_options_necessarily_type_specific_seq
1051 {
1052     \keys_define:nn { zref-clever / langsetup }
1053     {
1054         #1 .value_required:n = true ,
1055         #1 .code:n =
1056         {
1057             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1058             {
1059                 \msg_warning:nnn { zref-clever }
1060                 { option-only-type-specific } {#1}
1061             }
1062             {
1063                 \__zrefclever_declare_type_transl:Vnn
1064                 \l__zrefclever_dict_language_tl
1065                 \l__zrefclever_setup_type_tl
1066                 {#1} {##1}
1067             }
1068         } ,
1069     }
1070 }

```

## 6 User interface

### 6.1 \zcref

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

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

```

1071 \NewDocumentCommand \zcref { s O { } m }
1072 { \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>}}
1073 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1074 {
1075     \group_begin:

```

Set options.

```

1076     \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1077     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1078     \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.

```

1079     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with `zref-check`.

```

1080     \bool_lazy_and:nnT
1081     { \l__zrefclever_zrefcheck_available_bool }
1082     { \l__zrefclever_zcref_with_check_bool }
1083     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1084     \bool_lazy_or:nnT
1085     { \l__zrefclever_typeset_sort_bool }
1086     { \l__zrefclever_typeset_range_bool }
1087     { \__zrefclever_sort_labels: }

```

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

```

1088     \group_begin:
1089     \l__zrefclever_ref_typeset_font_tl
1090     \__zrefclever_typeset_refs:
1091     \group_end:

```

Typeset note.

```

1092     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1093     {
1094         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1095         \l_tmpa_tl
1096         \l__zrefclever_zcref_note_tl
1097     }

```

Integration with `zref-check`.

```

1098     \bool_lazy_and:nnT
1099     { \l__zrefclever_zrefcheck_available_bool }
1100     { \l__zrefclever_zcref_with_check_bool }
1101     {
1102         \zrefcheck_zcref_end_label_maybe:
1103         \zrefcheck_zcref_run_checks_on_labels:n
1104         { \l__zrefclever_zcref_labels_seq }
1105     }
1106     \group_end:
1107 }

```

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

```
\l_zrefclever_zcref_labels_seq
\l_zrefclever_link_star_bool
```

```
1108 \seq_new:N \l__zrefclever_zcref_labels_seq
1109 \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>}
```

```
1110 \NewDocumentCommand \zcpageref { s O { } m }
1111 {
1112   \IfBooleanTF {#1}
1113     { \zcref*[#2, ref = page] {#3} }
1114     { \zcref [ #2, ref = page] {#3} }
1115 }
```

(End definition for `\zcpageref`.)

## 7 Sorting

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

Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.

```
\l_zrefclever_label_type_a_tl
\l_zrefclever_label_type_b_tl
\l_zrefclever_label_enclcnt_a_tl
\l_zrefclever_label_enclcnt_b_tl
\l_zrefclever_label_enclval_a_tl
\l_zrefclever_label_enclval_b_tl
\l_zrefclever_label_extdoc_a_tl
\l_zrefclever_label_extdoc_b_tl
```

```
1116 \tl_new:N \l__zrefclever_label_type_a_tl
1117 \tl_new:N \l__zrefclever_label_type_b_tl
1118 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
1119 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1120 \tl_new:N \l__zrefclever_label_enclval_a_tl
1121 \tl_new:N \l__zrefclever_label_enclval_b_tl
1122 \tl_new:N \l__zrefclever_label_extdoc_a_tl
1123 \tl_new:N \l__zrefclever_label_extdoc_b_tl
```

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

```
\l_zrefclever_sort_decided_bool
```

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

```
1124 \bool_new:N \l__zrefclever_sort_decided_bool
```

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

\l\_zrefclever\_sort\_prior\_a\_int Auxiliary variables for \\_\_zrefclever\_sort\_default\_different\_types:nn. Store the  
\l\_zrefclever\_sort\_prior\_b\_int sort priority of the “current” and “next” labels.

```
1125 \int_new:N \l__zrefclever_sort_prior_a_int
1126 \int_new:N \l__zrefclever_sort_prior_b_int
```

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

\l\_zrefclever\_label\_types\_seq Stores the order in which reference types appear in the label list supplied by the user in  
\zcref. This variable is populated by \\_\_zrefclever\_label\_type\_put\_new\_right:n  
at the start of \\_\_zrefclever\_sort\_labels:. This order is required as a “last resort”  
sort criterion between the reference types, for use in \\_\_zrefclever\_sort\_default\_  
different\_types:nn.

```
1127 \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.

```
1128 \cs_new_protected:Npn \__zrefclever_sort_labels:
1129 {
```

Store label types sequence.

```
1130   \seq_clear:N \l__zrefclever_label_types_seq
1131   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1132   {
1133     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1134     \__zrefclever_label_type_put_new_right:n
1135   }
```

Sort.

```
1136   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1137   {
1138     \zref@ifrefundefined {##1}
1139     {
1140       \zref@ifrefundefined {##2}
1141       {
1142         % Neither label is defined.
1143         \sort_return_same:
1144       }
1145       {
1146         % The second label is defined, but the first isn't, leave the
1147         % undefined first (to be more visible).
1148         \sort_return_same:
1149       }
1150     }
1151     {
1152       \zref@ifrefundefined {##2}
1153       {
1154         % The first label is defined, but the second isn't, bring the
```

```

1155         % second forward.
1156         \sort_return_swapped:
1157     }
1158     {
1159         % The interesting case: both labels are defined.  References
1160         % to the "default" property or to the "page" are quite
1161         % different with regard to sorting, so we branch them here to
1162         % specialized functions.
1163         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1164             { \__zrefclever_sort_page:nn {##1} {##2} }
1165             { \__zrefclever_sort_default:nn {##1} {##2} }
1166     }
1167 }
1168 }
1169 }

```

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

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

```

\__zrefclever_label_type_put_new_right:n {\label}

1170 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1171 {
1172     \tl_set:Nx \l__zrefclever_label_type_a_tl
1173     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1174     \seq_if_in:NVF \l__zrefclever_label_types_seq
1175     \l__zrefclever_label_type_a_tl
1176     {
1177         \seq_put_right:NV \l__zrefclever_label_types_seq
1178         \l__zrefclever_label_type_a_tl
1179     }
1180 }

```

(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}

1181 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1182 {
1183     \tl_set:Nx \l__zrefclever_label_type_a_tl
1184     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }

```

```

1185 \tl_set:Nx \l__zrefclever_label_type_b_tl
1186 { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1187
1188 \bool_if:nTF
1189 {
1190   % The second label has a type, but the first doesn't, leave the
1191   % undefined first (to be more visible).
1192   \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1193   ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1194 }
1195 { \sort_return_same: }
1196 {
1197   \bool_if:nTF
1198   {
1199     % The first label has a type, but the second doesn't, bring the
1200     % second forward.
1201     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1202     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1203   }
1204   { \sort_return_swapped: }
1205   {
1206     \bool_if:nTF
1207     {
1208       % The interesting case: both labels have a type...
1209       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1210       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1211     }
1212     {
1213       \tl_if_eq:NNTF
1214       \l__zrefclever_label_type_a_tl
1215       \l__zrefclever_label_type_b_tl
1216       % ...and it's the same type.
1217       { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1218       % ...and they are different types.
1219       { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1220     }
1221     {
1222       % Neither label has a type. We can't do much of meaningful
1223       % here, but if it's the same counter, compare it.
1224       \exp_args:Nxx \tl_if_eq:nnTF
1225       { \zref@extractdefault {#1} {zc@counter} { } }
1226       { \zref@extractdefault {#2} {zc@counter} { } }
1227       {
1228         \int_compare:nNnTF
1229         { \zref@extractdefault {#1} {zc@cntval} { -1 } }
1230         >
1231         { \zref@extractdefault {#2} {zc@cntval} { -1 } }
1232         { \sort_return_swapped: }
1233         { \sort_return_same: }
1234       }
1235       { \sort_return_same: }
1236     }
1237   }
1238 }

```

1239 }

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

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

1240 \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}\}

1241 \cs\_new\_protected:Npn \\_zrefclever\_sort\_default\_same\_type:nn #1#2

1242 {

1243 \tl\_set:Nx \l\_\_zrefclever\_label\_enclcnt\_a\_tl

1244 { \zref@extractdefault {#1} { zc@enclcnt } { \c\_empty\_tl } }

1245 \tl\_set:Nx \l\_\_zrefclever\_label\_enclcnt\_a\_tl

1246 { \tl\_reverse\_items:V \l\_\_zrefclever\_label\_enclcnt\_a\_tl }

1247 \tl\_set:Nx \l\_\_zrefclever\_label\_enclcnt\_b\_tl

1248 { \zref@extractdefault {#2} { zc@enclcnt } { \c\_empty\_tl } }

1249 \tl\_set:Nx \l\_\_zrefclever\_label\_enclcnt\_b\_tl

1250 { \tl\_reverse\_items:V \l\_\_zrefclever\_label\_enclcnt\_b\_tl }

1251 \tl\_set:Nx \l\_\_zrefclever\_label\_enclval\_a\_tl

1252 { \zref@extractdefault {#1} { zc@enclval } { \c\_empty\_tl } }

1253 \tl\_set:Nx \l\_\_zrefclever\_label\_enclval\_a\_tl

1254 { \tl\_reverse\_items:V \l\_\_zrefclever\_label\_enclval\_a\_tl }

1255 \tl\_set:Nx \l\_\_zrefclever\_label\_enclval\_b\_tl

1256 { \zref@extractdefault {#2} { zc@enclval } { \c\_empty\_tl } }

1257 \tl\_set:Nx \l\_\_zrefclever\_label\_enclval\_b\_tl

1258 { \tl\_reverse\_items:V \l\_\_zrefclever\_label\_enclval\_b\_tl }

1259 \tl\_set:Nx \l\_\_zrefclever\_label\_extdoc\_a\_tl

1260 { \zref@extractdefault {#1} { externaldocument } { \c\_empty\_tl } }

1261 \tl\_set:Nx \l\_\_zrefclever\_label\_extdoc\_b\_tl

1262 { \zref@extractdefault {#2} { externaldocument } { \c\_empty\_tl } }

1263

1264 \bool\_set\_false:N \l\_\_zrefclever\_sort\_decided\_bool

1265

1266 % First we check if there's any "external document" difference (coming  
1267 % from 'zref-xr') and, if so, sort based on that.

1268 \tl\_if\_eq:NNTF

1269 \l\_\_zrefclever\_label\_extdoc\_a\_tl

1270 \l\_\_zrefclever\_label\_extdoc\_b\_tl

1271 {

1272 \bool\_if:nTF

1273 {

1274 \tl\_if\_empty\_p:V \l\_\_zrefclever\_label\_extdoc\_a\_tl &&

1275 ! \tl\_if\_empty\_p:V \l\_\_zrefclever\_label\_extdoc\_b\_tl

1276 }

1277 {

1278 \bool\_set\_true:N \l\_\_zrefclever\_sort\_decided\_bool

1279 \sort\_return\_same:

1280 }

1281 {

1282 \bool\_if:nTF

1283 {

1284 ! \tl\_if\_empty\_p:V \l\_\_zrefclever\_label\_extdoc\_a\_tl &&

1285 \tl\_if\_empty\_p:V \l\_\_zrefclever\_label\_extdoc\_b\_tl

1286 }

}

```

1287     {
1288         \bool_set_true:N \l__zrefclever_sort_decided_bool
1289         \sort_return_swapped:
1290     }
1291     {
1292         \bool_set_true:N \l__zrefclever_sort_decided_bool
1293         % Two different "external documents": last resort, sort by the
1294         % document name itself.
1295         \str_compare:eNeTF
1296         { \l__zrefclever_label_extdoc_b_tl } <
1297         { \l__zrefclever_label_extdoc_a_tl }
1298         { \sort_return_swapped: }
1299         { \sort_return_same:      }
1300     }
1301 }
1302 }
1303
1304 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1305 {
1306     \bool_if:nTF
1307     {
1308         % Both are empty: neither label has any (further) "enclosing
1309         % counters" (left).
1310         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1311         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1312     }
1313     {
1314         \exp_args:Nxx \tl_if_eq:nnTF
1315         { \zref@extractdefault {#1} {zc@counter} { } }
1316         { \zref@extractdefault {#2} {zc@counter} { } }
1317         {
1318             \bool_set_true:N \l__zrefclever_sort_decided_bool
1319             \int_compare:nNnTF
1320             { \zref@extractdefault {#1} {zc@cntval} { -1 } }
1321             >
1322             { \zref@extractdefault {#2} {zc@cntval} { -1 } }
1323             { \sort_return_swapped: }
1324             { \sort_return_same:      }
1325         }
1326         {
1327             \msg_warning:nnnn {zref-clever}
1328             {counters-not-nested} {#1} {#2}
1329             \bool_set_true:N \l__zrefclever_sort_decided_bool
1330             \sort_return_same:
1331         }
1332     }
1333 }
1334 \bool_if:nTF
1335 {
1336     % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1337     \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1338 }
1339 {
1340     \int_zero:N \l_tmpb_int

```



```

1341 \tl_map_inline:Nn \l__zrefclever_label_enclcnt_b_tl
1342 {
1343   \int_incr:N \l_tmpb_int
1344   \exp_args:Nnx \tl_if_eq:nnT {##1}
1345   { \zref@extractdefault {#1} { zc@counter } { } }
1346   {
1347     \tl_map_break:n
1348     {
1349       \int_compare:nNnTF
1350       { \zref@extractdefault {#1} { zc@cntval } { } }
1351       >
1352       {
1353         \tl_item:Nn \l__zrefclever_label_enclval_b_tl
1354         { \l_tmpb_int }
1355       }
1356       { \sort_return_swapped: }
1357       { \sort_return_same: }
1358       \bool_set_true:N \l__zrefclever_sort_decided_bool
1359     }
1360   }
1361 }
1362 \bool_if:NF \l__zrefclever_sort_decided_bool
1363 {
1364   \msg_warning:nnnn { zref-clever }
1365   { counters-not-nested } {#1} {#2}
1366   \bool_set_true:N \l__zrefclever_sort_decided_bool
1367   \sort_return_same:
1368 }
1369 }
1370 {
1371   \bool_if:nTF
1372   {
1373     % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1374     \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1375   }
1376   {
1377     \int_zero:N \l_tmpa_int
1378     \tl_map_inline:Nn \l__zrefclever_label_enclcnt_a_tl
1379     {
1380       \int_incr:N \l_tmpa_int
1381       \exp_args:Nnx \tl_if_eq:nnT {##1}
1382       { \zref@extractdefault {#2} { zc@counter } { } }
1383       {
1384         \tl_map_break:n
1385         {
1386           \int_compare:nNnTF
1387           {
1388             \tl_item:Nn
1389             \l__zrefclever_label_enclval_a_tl
1390             { \l_tmpa_int }
1391           }
1392           <
1393           {
1394             \zref@extractdefault {#2}

```

```

1395         { zc@cntval } { }
1396     }
1397     { \sort_return_same:    }
1398     { \sort_return_swapped: }
1399     \bool_set_true:N
1400     \l__zrefclever_sort_decided_bool
1401 }
1402 }
1403 }
1404 \bool_if:NF \l__zrefclever_sort_decided_bool
1405 {
1406     \msg_warning:nnnn { zref-clever }
1407     { counters-not-nested } {#1} {#2}
1408     \bool_set_true:N \l__zrefclever_sort_decided_bool
1409     \sort_return_same:
1410 }
1411 }
1412 {
1413     % Neither is empty: we can (possibly) compare the values
1414     % of the current enclosing counter in the loop, if they
1415     % are equal, we are still in the loop, if they are not, a
1416     % sorting decision can be made directly.
1417     \exp_args:Nxx \tl_if_eq:nnTF
1418     { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1419     { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1420     {
1421         \int_compare:nNnTF
1422         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1423         =
1424         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1425         {
1426             \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1427             { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1428             \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1429             { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1430             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1431             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1432             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1433             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1434         }
1435         {
1436             \bool_set_true:N \l__zrefclever_sort_decided_bool
1437             \int_compare:nNnTF
1438             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1439             >
1440             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1441             { \sort_return_swapped: }
1442             { \sort_return_same:    }
1443         }
1444     }
1445 }
1446 \msg_warning:nnnn { zref-clever }
1447 { counters-not-nested } {#1} {#2}
1448 \bool_set_true:N \l__zrefclever_sort_decided_bool

```

```

1449         \sort_return_same:
1450     }
1451 }
1452 }
1453 }
1454 }
1455 }

```

(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}
1456 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1457 {

```

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”.

```

1458     \int_zero:N \l__zrefclever_sort_prior_a_int
1459     \int_zero:N \l__zrefclever_sort_prior_b_int
1460     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1461     {
1462         \tl_if_eq:nnTF {##2} {{othertypes}}
1463         {
1464             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1465             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1466             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1467             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1468         }
1469         {
1470             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1471             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1472             {
1473                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1474                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1475             }
1476         }
1477     }

```

Then do the actual sorting.

```

1478     \bool_if:nTF
1479     {
1480         \int_compare_p:nNn
1481         { \l__zrefclever_sort_prior_a_int } <
1482         { \l__zrefclever_sort_prior_b_int }
1483     }
1484     { \sort_return_same: }
1485     {
1486         \bool_if:nTF
1487         {
1488             \int_compare_p:nNn
1489             { \l__zrefclever_sort_prior_a_int } >
1490             { \l__zrefclever_sort_prior_b_int }
1491         }
1492         { \sort_return_swapped: }

```

```

1493     {
1494         % Sort priorities are equal: the type that occurs first in
1495         % 'labels', as given by the user, is kept (or brought) forward.
1496         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1497         {
1498             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1499             { \seq_map_break:n { \sort_return_same: } }
1500             {
1501                 \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##1}
1502                 { \seq_map_break:n { \sort_return_swapped: } }
1503             }
1504         }
1505     }
1506 }
1507 }

```

(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}

1508 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1509 {
1510     \int_compare:nNnTF
1511     { \zref@extractdefault {#1} { abspage } {-1} }
1512     >
1513     { \zref@extractdefault {#2} { abspage } {-1} }
1514     { \sort_return_swapped: }
1515     { \sort_return_same: }
1516 }

```

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

## 8 Typesetting

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

While processing the label stack (kept in `\l__zrefclever_typeset_labels_seq`), `\__zrefclever_typeset_refs:` “sees” two labels, and two labels only, the “current” one (kept in `\l__zrefclever_label_a_tl`), and the “next” one (kept in `\l__zrefclever_label_b_tl`). However, the typesetting needs (a lot) more information than just these two immediate labels to make a number of critical decisions. Some examples: i) We

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

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

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

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

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this (and good ones at that) see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be an option receiving the label(s) not to be compressed, this would be a repetition, but would keep the syntax clean. All in all, probably the best is simply not to allow individual inhibition of compression. We can already control compression of each `\zcref` call with existing options, this should be enough. I don’t think the small extra flexibility individual label control for this would grant is worth the syntax disruption it

would entail. Anyway, it would be easy to deal with this in case the need arose, by just adding another condition (coming from whatever the chosen syntax was) when we check for `\__zrefclever_labels_in_sequence:nn` in `\__zrefclever_typeset_refs_not_last_of_type:.` But I remain unconvinced of the pertinence of doing so.

## Variables

<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>	1517 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l_zrefclever_last_of_type_bool</code>	1518 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	1519 <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>	1520 <code>\int_new:N \l__zrefclever_type_count_int</code>
	1521 <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>	1522 <code>\tl_new:N \l__zrefclever_label_a_tl</code>
<code>\l_zrefclever_typeset_queue_curr_tl</code>	1523 <code>\tl_new:N \l__zrefclever_label_b_tl</code>
<code>\l_zrefclever_type_first_label_tl</code>	1524 <code>\tl_new:N \l__zrefclever_typeset_queue_prev_tl</code>
<code>\l_zrefclever_type_first_label_type_tl</code>	1525 <code>\tl_new:N \l__zrefclever_typeset_queue_curr_tl</code>
	1526 <code>\tl_new:N \l__zrefclever_type_first_label_tl</code>
	1527 <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>	1528 <code>\tl_new:N \l__zrefclever_type_name_tl</code>
<code>\l_zrefclever_name_format_tl</code>	1529 <code>\bool_new:N \l__zrefclever_name_in_link_bool</code>
<code>\l_zrefclever_name_format_fallback_tl</code>	1530 <code>\tl_new:N \l__zrefclever_name_format_tl</code>
	1531 <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>	1532 <code>\int_new:N \l__zrefclever_range_count_int</code>
<code>\l_zrefclever_range_beg_label_tl</code>	1533 <code>\int_new:N \l__zrefclever_range_same_count_int</code>
<code>\l_zrefclever_next_maybe_range_bool</code>	1534 <code>\tl_new:N \l__zrefclever_range_beg_label_tl</code>
<code>\l_zrefclever_next_is_same_bool</code>	1535 <code>\bool_new:N \l__zrefclever_next_maybe_range_bool</code>
	1536 <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 1537 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl 1538 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl 1539 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl 1540 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl 1541 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl 1542 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl 1543 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_rangeseq_tl 1544 \tl_new:N \l__zrefclever_rangeseq_tl
\l__zrefclever_refpre_out_tl 1545 \tl_new:N \l__zrefclever_refpre_out_tl
\l__zrefclever_refpos_out_tl 1546 \tl_new:N \l__zrefclever_refpos_out_tl
\l__zrefclever_refpre_in_tl 1547 \tl_new:N \l__zrefclever_refpre_in_tl
\l__zrefclever_refpos_in_tl 1548 \tl_new:N \l__zrefclever_refpos_in_tl
\l__zrefclever_namefont_tl 1549 \tl_new:N \l__zrefclever_namefont_tl
\l__zrefclever_reffont_out_tl 1550 \tl_new:N \l__zrefclever_reffont_out_tl
\l__zrefclever_reffont_in_tl 1551 \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`.

```

1552 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1553 {
1554   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1555   \l__zrefclever_zcref_labels_seq
1556   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1557   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1558   \tl_clear:N \l__zrefclever_type_first_label_tl
1559   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1560   \tl_clear:N \l__zrefclever_range_beg_label_tl
1561   \int_zero:N \l__zrefclever_label_count_int
1562   \int_zero:N \l__zrefclever_type_count_int
1563   \int_zero:N \l__zrefclever_range_count_int
1564   \int_zero:N \l__zrefclever_range_same_count_int
1565
1566   % Get type block options (not type-specific).
1567   \__zrefclever_get_ref_string:nN { tpairsep }
1568   \l__zrefclever_tpairsep_tl
1569   \__zrefclever_get_ref_string:nN { tlistsep }
1570   \l__zrefclever_tlistsep_tl
1571   \__zrefclever_get_ref_string:nN { tlastsep }
1572   \l__zrefclever_tlastsep_tl
1573
1574   % Process label stack.
1575   \bool_set_false:N \l__zrefclever_typeset_last_bool
1576   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1577   {
1578     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1579     \l__zrefclever_label_a_tl
1580     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1581     {
1582       \tl_clear:N \l__zrefclever_label_b_tl

```

```

1583         \bool_set_true:N \l__zrefclever_typeset_last_bool
1584     }
1585     {
1586         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1587         \l__zrefclever_label_b_tl
1588     }
1589
1590 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1591 {
1592     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1593     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1594 }
1595 {
1596     \tl_set:Nx \l__zrefclever_label_type_a_tl
1597     {
1598         \zref@extractdefault
1599         { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1600     }
1601     \tl_set:Nx \l__zrefclever_label_type_b_tl
1602     {
1603         \zref@extractdefault
1604         { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1605     }
1606 }
1607
1608 % First, we establish whether the "current label" (i.e. 'a') is the
1609 % last one of its type. This can happen because the "next label"
1610 % (i.e. 'b') is of a different type (or different definition status),
1611 % or because we are at the end of the list.
1612 \bool_if:NTF \l__zrefclever_typeset_last_bool
1613 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1614 {
1615     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1616     {
1617         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1618         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1619         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1620     }
1621     {
1622         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1623         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1624         {
1625             % Neither is undefined, we must check the types.
1626             \bool_if:nTF
1627             {
1628                 % Both empty: same "type".
1629                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1630                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1631             }
1632             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1633             {
1634                 \bool_if:nTF
1635                 {
1636                     % Neither empty: compare types.

```



```

1637         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1638         &&
1639         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1640     }
1641     {
1642         \tl_if_eq:NNTF
1643             \l__zrefclever_label_type_a_tl
1644             \l__zrefclever_label_type_b_tl
1645         {
1646             \bool_set_false:N
1647                 \l__zrefclever_last_of_type_bool
1648         }
1649         {
1650             \bool_set_true:N
1651                 \l__zrefclever_last_of_type_bool
1652         }
1653     }
1654     % One empty, the other not: different "types".
1655     {
1656         \bool_set_true:N
1657             \l__zrefclever_last_of_type_bool
1658     }
1659 }
1660 }
1661 }
1662 }
1663
1664 % Handle warnings in case of reference or type undefined.
1665 \zref@refused { \l__zrefclever_label_a_tl }
1666 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1667 {}
1668 {
1669     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1670     {
1671         \msg_warning:nxx { zref-clever } { missing-type }
1672         { \l__zrefclever_label_a_tl }
1673     }
1674 }
1675
1676 % Get type-specific separators, refpre/pos and font options, once per
1677 % type.
1678 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1679 {
1680     \__zrefclever_get_ref_string:nN { namesep      }
1681     \l__zrefclever_namesep_tl
1682     \__zrefclever_get_ref_string:nN { rangesep     }
1683     \l__zrefclever_rangesep_tl
1684     \__zrefclever_get_ref_string:nN { pairsep      }
1685     \l__zrefclever_pairsep_tl
1686     \__zrefclever_get_ref_string:nN { listsep      }
1687     \l__zrefclever_listsep_tl
1688     \__zrefclever_get_ref_string:nN { lastsep      }
1689     \l__zrefclever_lastsep_tl
1690     \__zrefclever_get_ref_string:nN { refpre       }

```

```

1691         \l__zrefclever_refpre_out_tl
1692         \__zrefclever_get_ref_string:nN { refpos      }
1693         \l__zrefclever_refpos_out_tl
1694         \__zrefclever_get_ref_string:nN { refpre-in   }
1695         \l__zrefclever_refpre_in_tl
1696         \__zrefclever_get_ref_string:nN { refpos-in   }
1697         \l__zrefclever_refpos_in_tl
1698         \__zrefclever_get_ref_font:nN   { namefont    }
1699         \l__zrefclever_namefont_tl
1700         \__zrefclever_get_ref_font:nN   { reffont     }
1701         \l__zrefclever_reffont_out_tl
1702         \__zrefclever_get_ref_font:nN   { reffont-in  }
1703         \l__zrefclever_reffont_in_tl
1704     }
1705
1706     % Here we send this to a couple of auxiliary functions.
1707     \bool_if:NTF \l__zrefclever_last_of_type_bool
1708     % There exists no next label of the same type as the current.
1709     { \__zrefclever_typeset_refs_last_of_type: }
1710     % There exists a next label of the same type as the current.
1711     { \__zrefclever_typeset_refs_not_last_of_type: }
1712 }
1713 }

```

(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.

```

1714 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1715 {
1716     % Process the current label to the current queue.
1717     \int_case:nnF { \l__zrefclever_label_count_int }
1718     {
1719         % It is the last label of its type, but also the first one, and that's
1720         % what matters here: just store it.
1721         { 0 }
1722         {
1723             \tl_set:NV \l__zrefclever_type_first_label_tl
1724             \l__zrefclever_label_a_tl
1725             \tl_set:NV \l__zrefclever_type_first_label_type_tl
1726             \l__zrefclever_label_type_a_tl
1727         }
1728
1729         % The last is the second: we have a pair (if not repeated).
1730         { 1 }
1731         {

```

```

1732 \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1733 {
1734   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1735   {
1736     \exp_not:V \l__zrefclever_pairsep_tl
1737     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1738   }
1739 }
1740 }
1741 }
1742 % Last is third or more of its type: without repetition, we'd have the
1743 % last element on a list, but control for possible repetition.
1744 {
1745   \int_case:nNnF { \l__zrefclever_range_count_int }
1746   {
1747     % There was no range going on.
1748     { 0 }
1749     {
1750       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1751       {
1752         \exp_not:V \l__zrefclever_lastsep_tl
1753         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1754       }
1755     }
1756     % Last in the range is also the second in it.
1757     { 1 }
1758     {
1759       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1760       {
1761         % We know 'range_beg_label' is not empty, since this is the
1762         % second element in the range, but the third or more in the
1763         % type list.
1764         \exp_not:V \l__zrefclever_listsep_tl
1765         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1766         \int_compare:nNnF
1767         { \l__zrefclever_range_same_count_int } = { 1 }
1768         {
1769           \exp_not:V \l__zrefclever_lastsep_tl
1770           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1771         }
1772       }
1773     }
1774   }
1775   % Last in the range is third or more in it.
1776   {
1777     \int_case:nNnF
1778     {
1779       \l__zrefclever_range_count_int -
1780       \l__zrefclever_range_same_count_int
1781     }
1782     {
1783       % Repetition, not a range.
1784       { 0 }
1785       {

```

```

1786         % If 'range_beg_label' is empty, it means it was also the
1787         % first of the type, and hence was already handled.
1788         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1789         {
1790             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1791             {
1792                 \exp_not:V \l__zrefclever_lastsep_tl
1793                 \__zrefclever_get_ref:V
1794                 \l__zrefclever_range_beg_label_tl
1795             }
1796         }
1797     }
1798     % A 'range', but with no skipped value, treat as list.
1799     { 1 }
1800     {
1801         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1802         {
1803             % Ditto.
1804             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1805             {
1806                 \exp_not:V \l__zrefclever_listsep_tl
1807                 \__zrefclever_get_ref:V
1808                 \l__zrefclever_range_beg_label_tl
1809             }
1810             \exp_not:V \l__zrefclever_lastsep_tl
1811             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1812         }
1813     }
1814 }
1815 {
1816     % An actual range.
1817     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1818     {
1819         % Ditto.
1820         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1821         {
1822             \exp_not:V \l__zrefclever_lastsep_tl
1823             \__zrefclever_get_ref:V
1824             \l__zrefclever_range_beg_label_tl
1825         }
1826         \exp_not:V \l__zrefclever_rangesep_tl
1827         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1828     }
1829 }
1830 }
1831 }
1832
1833 % Handle "range" option. The idea is simple: if the queue is not empty,
1834 % we replace it with the end of the range (or pair). We can still
1835 % retrieve the end of the range from 'label_a' since we know to be
1836 % processing the last label of its type at this point.
1837 \bool_if:NT \l__zrefclever_typeset_range_bool
1838 {
1839     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl

```

```

1840 {
1841   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1842   { }
1843   {
1844     \msg_warning:nxx { zref-clever } { single-element-range }
1845     { \l__zrefclever_type_first_label_type_tl }
1846   }
1847 }
1848 {
1849   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1850   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1851   { }
1852   {
1853     \__zrefclever_labels_in_sequence:nn
1854     { \l__zrefclever_type_first_label_tl }
1855     { \l__zrefclever_label_a_tl }
1856   }
1857   \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1858   {
1859     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1860     { \exp_not:V \l__zrefclever_pairsep_tl }
1861     { \exp_not:V \l__zrefclever_rangesep_tl }
1862     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1863   }
1864 }
1865 }
1866
1867 % Now that the type block is finished, we can add the name and the first
1868 % ref to the queue. Also, if "typeset" option is not "both", handle it
1869 % here as well.
1870 \__zrefclever_type_name_setup:
1871 \bool_if:nTF
1872 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1873 {
1874   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1875   { \__zrefclever_get_ref_first: }
1876 }
1877 {
1878   \bool_if:nTF
1879   { \l__zrefclever_typeset_ref_bool }
1880   {
1881     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1882     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1883   }
1884   {
1885     \bool_if:nTF
1886     { \l__zrefclever_typeset_name_bool }
1887     {
1888       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1889       {
1890         \bool_if:NTF \l__zrefclever_name_in_link_bool
1891         {
1892           \exp_not:N \group_begin:
1893           \exp_not:V \l__zrefclever_namefont_tl

```

```

1894 % It's two '@s', but escaped for DocStrip.
1895 \exp_not:N \hyper@@link
1896 {
1897   \__zrefclever_extract_url:V
1898   \l__zrefclever_type_first_label_tl
1899 }
1900 {
1901   \zref@extractdefault
1902   { \l__zrefclever_type_first_label_tl }
1903   { anchor } {}
1904 }
1905 { \exp_not:V \l__zrefclever_type_name_tl }
1906 \exp_not:N \group_end:
1907 }
1908 {
1909   \exp_not:N \group_begin:
1910   \exp_not:V \l__zrefclever_namefont_tl
1911   \exp_not:V \l__zrefclever_type_name_tl
1912   \exp_not:N \group_end:
1913 }
1914 }
1915 }
1916 {
1917   % Logically, this case would correspond to "typeset=none", but
1918   % it should not occur, given that the options are set up to
1919   % typeset either "ref" or "name". Still, leave here a
1920   % sensible fallback, equal to the behavior of "both".
1921   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1922     { \__zrefclever_get_ref_first: }
1923 }
1924 }
1925 }
1926
1927 % Typeset the previous type, if there is one.
1928 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1929 {
1930   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1931     { \l__zrefclever_tlistsep_tl }
1932   \l__zrefclever_typeset_queue_prev_tl
1933 }
1934
1935 % Wrap up loop, or prepare for next iteration.
1936 \bool_if:NTF \l__zrefclever_typeset_last_bool
1937 {
1938   % We are finishing, typeset the current queue.
1939   \int_case:nnF { \l__zrefclever_type_count_int }
1940   {
1941     % Single type.
1942     { 0 }
1943     { \l__zrefclever_typeset_queue_curr_tl }
1944     % Pair of types.
1945     { 1 }
1946     {
1947       \l__zrefclever_tpairsep_tl

```

```

1948         \l__zrefclever_typeset_queue_curr_tl
1949     }
1950 }
1951 {
1952     % Last in list of types.
1953     \l__zrefclever_tlastsep_tl
1954     \l__zrefclever_typeset_queue_curr_tl
1955 }
1956 }
1957 {
1958     % There are further labels, set variables for next iteration.
1959     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1960         \l__zrefclever_typeset_queue_curr_tl
1961     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1962     \tl_clear:N \l__zrefclever_type_first_label_tl
1963     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1964     \tl_clear:N \l__zrefclever_range_beg_label_tl
1965     \int_zero:N \l__zrefclever_label_count_int
1966     \int_incr:N \l__zrefclever_type_count_int
1967     \int_zero:N \l__zrefclever_range_count_int
1968     \int_zero:N \l__zrefclever_range_same_count_int
1969 }
1970 }

```

(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.

```

1971 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1972 {
1973     % Signal if next label may form a range with the current one (only
1974     % considered if compression is enabled in the first place).
1975     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1976     \bool_set_false:N \l__zrefclever_next_is_same_bool
1977     \bool_if:NT \l__zrefclever_typeset_compress_bool
1978     {
1979         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1980         { }
1981         {
1982             \__zrefclever_labels_in_sequence:nn
1983             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1984         }
1985     }
1986
1987     % Process the current label to the current queue.
1988     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1989     {
1990         % Current label is the first of its type (also not the last, but it
1991         % doesn't matter here): just store the label.
1992         \tl_set:NV \l__zrefclever_type_first_label_tl
1993             \l__zrefclever_label_a_tl
1994         \tl_set:NV \l__zrefclever_type_first_label_type_tl
1995             \l__zrefclever_label_type_a_tl
1996
1997         % If the next label may be part of a range, we set 'range_beg_label'

```

```

1998 % to "empty" (we deal with it as the "first", and must do it there, to
1999 % handle hyperlinking), but also step the range counters.
2000 \bool_if:NT \l__zrefclever_next_maybe_range_bool
2001 {
2002   \tl_clear:N \l__zrefclever_range_beg_label_tl
2003   \int_incr:N \l__zrefclever_range_count_int
2004   \bool_if:NT \l__zrefclever_next_is_same_bool
2005     { \int_incr:N \l__zrefclever_range_same_count_int }
2006 }
2007 }
2008 {
2009 % Current label is neither the first (nor the last) of its type.
2010 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2011 {
2012   % Starting, or continuing a range.
2013   \int_compare:nNnTF
2014     { \l__zrefclever_range_count_int } = { 0 }
2015     {
2016       % There was no range going, we are starting one.
2017       \tl_set:NV \l__zrefclever_range_beg_label_tl
2018         \l__zrefclever_label_a_tl
2019       \int_incr:N \l__zrefclever_range_count_int
2020       \bool_if:NT \l__zrefclever_next_is_same_bool
2021         { \int_incr:N \l__zrefclever_range_same_count_int }
2022     }
2023     {
2024       % Second or more in the range, but not the last.
2025       \int_incr:N \l__zrefclever_range_count_int
2026       \bool_if:NT \l__zrefclever_next_is_same_bool
2027         { \int_incr:N \l__zrefclever_range_same_count_int }
2028     }
2029 }
2030 {
2031 % Next element is not in sequence: there was no range, or we are
2032 % closing one.
2033 \int_case:nnF { \l__zrefclever_range_count_int }
2034 {
2035   % There was no range going on.
2036   { 0 }
2037   {
2038     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2039       {
2040         \exp_not:V \l__zrefclever_listsep_tl
2041         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2042       }
2043   }
2044   % Last is second in the range: if 'range_same_count' is also
2045   % '1', it's a repetition (drop it), otherwise, it's a "pair
2046   % within a list", treat as list.
2047   { 1 }
2048   {
2049     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2050       {
2051         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl

```



```

2052         {
2053             \exp_not:V \l__zrefclever_listsep_tl
2054             \__zrefclever_get_ref:V
2055             \l__zrefclever_range_beg_label_tl
2056         }
2057     \int_compare:nNnF
2058     { \l__zrefclever_range_same_count_int } = { 1 }
2059     {
2060         \exp_not:V \l__zrefclever_listsep_tl
2061         \__zrefclever_get_ref:V
2062         \l__zrefclever_label_a_tl
2063     }
2064 }
2065 }
2066 }
2067 {
2068     % Last is third or more in the range: if 'range_count' and
2069     % 'range_same_count' are the same, its a repetition (drop it),
2070     % if they differ by '1', its a list, if they differ by more,
2071     % it is a real range.
2072     \int_case:nnF
2073     {
2074         \l__zrefclever_range_count_int -
2075         \l__zrefclever_range_same_count_int
2076     }
2077     {
2078         { 0 }
2079         {
2080             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2081             {
2082                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2083                 {
2084                     \exp_not:V \l__zrefclever_listsep_tl
2085                     \__zrefclever_get_ref:V
2086                     \l__zrefclever_range_beg_label_tl
2087                 }
2088             }
2089         }
2090         { 1 }
2091         {
2092             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2093             {
2094                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2095                 {
2096                     \exp_not:V \l__zrefclever_listsep_tl
2097                     \__zrefclever_get_ref:V
2098                     \l__zrefclever_range_beg_label_tl
2099                 }
2100             }
2101             \exp_not:V \l__zrefclever_listsep_tl
2102             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2103         }
2104     }
2105 }

```

```

2106         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2107         {
2108             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2109             {
2110                 \exp_not:V \l__zrefclever_listsep_tl
2111                 \__zrefclever_get_ref:V
2112                 \l__zrefclever_range_beg_label_tl
2113             }
2114             \exp_not:V \l__zrefclever_rangesep_tl
2115             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2116         }
2117     }
2118 }
2119 % Reset counters.
2120 \int_zero:N \l__zrefclever_range_count_int
2121 \int_zero:N \l__zrefclever_range_same_count_int
2122 }
2123 }
2124 % Step label counter for next iteration.
2125 \int_incr:N \l__zrefclever_label_count_int
2126 }

```

(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.

```

2127 \cs_new_protected:Npn \__zrefclever_ref_default:

```

```

2128 { \zref@default }
2129 \cs_new_protected:Npn \__zrefclever_name_default:
2130 { \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{}}

2131 \cs_new:Npn \__zrefclever_get_ref:n #1
2132 {
2133   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2134   {
2135     \bool_if:nTF
2136     {
2137       \l__zrefclever_use_hyperref_bool &&
2138       ! \l__zrefclever_link_star_bool
2139     }
2140     {
2141       \exp_not:N \group_begin:
2142       \exp_not:V \l__zrefclever_reffont_out_tl
2143       \exp_not:V \l__zrefclever_refpre_out_tl
2144       \exp_not:N \group_begin:
2145       \exp_not:V \l__zrefclever_reffont_in_tl
2146       % It's two '@s', but escaped for DocStrip.
2147       \exp_not:N \hyper@@link
2148       { \__zrefclever_extract_url:n {#1} }
2149       { \zref@extractdefault {#1} { anchor } { } }
2150       {
2151         \exp_not:V \l__zrefclever_refpre_in_tl
2152         \zref@extractdefault {#1}
2153         { \l__zrefclever_ref_property_tl } { }
2154         \exp_not:V \l__zrefclever_refpos_in_tl
2155       }
2156       \exp_not:N \group_end:
2157       \exp_not:V \l__zrefclever_refpos_out_tl
2158       \exp_not:N \group_end:
2159     }
2160     {
2161       \exp_not:N \group_begin:
2162       \exp_not:V \l__zrefclever_reffont_out_tl
2163       \exp_not:V \l__zrefclever_refpre_out_tl
2164       \exp_not:N \group_begin:
2165       \exp_not:V \l__zrefclever_reffont_in_tl
2166       \exp_not:V \l__zrefclever_refpre_in_tl
2167       \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2168       \exp_not:V \l__zrefclever_refpos_in_tl
2169       \exp_not:N \group_end:
2170       \exp_not:V \l__zrefclever_refpos_out_tl
2171       \exp_not:N \group_end:
2172     }
2173   }

```

```

2174     { \__zrefclever_ref_default: }
2175   }
2176 \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.

```

2177 \cs_new:Npn \__zrefclever_get_ref_first:
2178 {
2179   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2180   { \__zrefclever_ref_default: }
2181   {
2182     \bool_if:NTF \l__zrefclever_name_in_link_bool
2183     {
2184       \zref@ifrefcontainsprop
2185       { \l__zrefclever_type_first_label_tl }
2186       { \l__zrefclever_ref_property_tl }
2187       {
2188         % It's two '@s', but escaped for DocStrip.
2189         \exp_not:N \hyper@@link
2190         {
2191           \__zrefclever_extract_url:V
2192           \l__zrefclever_type_first_label_tl
2193         }
2194         {
2195           \zref@extractdefault
2196           { \l__zrefclever_type_first_label_tl }
2197           { anchor } { }
2198         }
2199       }
2200       \exp_not:N \group_begin:
2201       \exp_not:V \l__zrefclever_namefont_tl
2202       \exp_not:V \l__zrefclever_type_name_tl
2203       \exp_not:N \group_end:
2204       \exp_not:V \l__zrefclever_namesep_tl
2205       \exp_not:N \group_begin:
2206       \exp_not:V \l__zrefclever_reffont_out_tl
2207       \exp_not:V \l__zrefclever_refpre_out_tl
2208       \exp_not:N \group_begin:
2209       \exp_not:V \l__zrefclever_reffont_in_tl
2210       \exp_not:V \l__zrefclever_refpre_in_tl
2211       \zref@extractdefault
2212       { \l__zrefclever_type_first_label_tl }
2213       { \l__zrefclever_ref_property_tl } { }
2214       \exp_not:V \l__zrefclever_refpos_in_tl
2215       \exp_not:N \group_end:
2216       % hyperlink makes it's own group, we'd like to close the

```

```

2217         % 'refpre-out' group after 'refpos-out', but... we close
2218         % it here, and give the trailing 'refpos-out' its own
2219         % group. This will result that formatting given to
2220         % 'refpre-out' will not reach 'refpos-out', but I see no
2221         % alternative, and this has to be handled specially.
2222         \exp_not:N \group_end:
2223     }
2224     \exp_not:N \group_begin:
2225     % Ditto: special treatment.
2226     \exp_not:V \l__zrefclever_reffont_out_tl
2227     \exp_not:V \l__zrefclever_refpos_out_tl
2228     \exp_not:N \group_end:
2229 }
2230 {
2231     \exp_not:N \group_begin:
2232     \exp_not:V \l__zrefclever_namefont_tl
2233     \exp_not:V \l__zrefclever_type_name_tl
2234     \exp_not:N \group_end:
2235     \exp_not:V \l__zrefclever_namesep_tl
2236     \__zrefclever_ref_default:
2237 }
2238 }
2239 {
2240     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2241     {
2242         \__zrefclever_name_default:
2243         \exp_not:V \l__zrefclever_namesep_tl
2244     }
2245     {
2246         \exp_not:N \group_begin:
2247         \exp_not:V \l__zrefclever_namefont_tl
2248         \exp_not:V \l__zrefclever_type_name_tl
2249         \exp_not:N \group_end:
2250         \exp_not:V \l__zrefclever_namesep_tl
2251     }
2252     \zref@ifrefcontainsprop
2253     { \l__zrefclever_type_first_label_tl }
2254     { \l__zrefclever_ref_property_tl }
2255     {
2256         \bool_if:nTF
2257         {
2258             \l__zrefclever_use_hyperref_bool &&
2259             ! \l__zrefclever_link_star_bool
2260         }
2261         {
2262             \exp_not:N \group_begin:
2263             \exp_not:V \l__zrefclever_reffont_out_tl
2264             \exp_not:V \l__zrefclever_refpre_out_tl
2265             \exp_not:N \group_begin:
2266             \exp_not:V \l__zrefclever_reffont_in_tl
2267             % It's two '@s', but escaped for DocStrip.
2268             \exp_not:N \hyper@@link
2269             {
2270                 \__zrefclever_extract_url:V

```

```

2271         \l__zrefclever_type_first_label_tl
2272     }
2273     {
2274         \zref@extractdefault
2275         { \l__zrefclever_type_first_label_tl }
2276         { anchor } { }
2277     }
2278     {
2279         \exp_not:V \l__zrefclever_refpre_in_tl
2280         \zref@extractdefault
2281         { \l__zrefclever_type_first_label_tl }
2282         { \l__zrefclever_ref_property_tl } { }
2283         \exp_not:V \l__zrefclever_refpos_in_tl
2284     }
2285     \exp_not:N \group_end:
2286     \exp_not:V \l__zrefclever_refpos_out_tl
2287     \exp_not:N \group_end:
2288 }
2289 {
2290     \exp_not:N \group_begin:
2291     \exp_not:V \l__zrefclever_reffont_out_tl
2292     \exp_not:V \l__zrefclever_refpre_out_tl
2293     \exp_not:N \group_begin:
2294     \exp_not:V \l__zrefclever_reffont_in_tl
2295     \exp_not:V \l__zrefclever_refpre_in_tl
2296     \zref@extractdefault
2297     { \l__zrefclever_type_first_label_tl }
2298     { \l__zrefclever_ref_property_tl } { }
2299     \exp_not:V \l__zrefclever_refpos_in_tl
2300     \exp_not:N \group_end:
2301     \exp_not:V \l__zrefclever_refpos_out_tl
2302     \exp_not:N \group_end:
2303 }
2304 }
2305 { \__zrefclever_ref_default: }
2306 }
2307 }
2308 }

```

(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.

2309 \cs\_new\_protected:Npn \\_\_zrefclever\_type\_name\_setup:

```

2310 {
2311   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2312   { \tl_clear:N \l__zrefclever_type_name_tl }
2313   {
2314     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2315     { \tl_clear:N \l__zrefclever_type_name_tl }
2316     {
2317       % Determine whether we should use capitalization, abbreviation,
2318       % and plural.
2319       \bool_lazy_or:nnTF
2320         { \l__zrefclever_capitalize_bool }
2321         {
2322           \l__zrefclever_capitalize_first_bool &&
2323           \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2324         }
2325         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2326         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2327       % If the queue is empty, we have a singular, otherwise, plural.
2328       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2329         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2330         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2331       \bool_lazy_and:nnTF
2332         { \l__zrefclever_abbrev_bool }
2333         {
2334           ! \int_compare_p:nNn
2335             { \l__zrefclever_type_count_int } = { 0 } ||
2336           ! \l__zrefclever_noabbrev_first_bool
2337         }
2338         {
2339           \tl_set:NV \l__zrefclever_name_format_fallback_tl
2340             \l__zrefclever_name_format_tl
2341           \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2342         }
2343         { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2344
2345       \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2346       {
2347         \prop_get:cVNF
2348         {
2349           l__zrefclever_type_
2350           \l__zrefclever_type_first_label_type_tl _options_prop
2351         }
2352         \l__zrefclever_name_format_tl
2353         \l__zrefclever_type_name_tl
2354         {
2355           \__zrefclever_get_type_transl:xxxNF
2356           { \l__zrefclever_ref_language_tl }
2357           { \l__zrefclever_type_first_label_type_tl }
2358           { \l__zrefclever_name_format_tl }
2359           \l__zrefclever_type_name_tl
2360           {
2361             \tl_clear:N \l__zrefclever_type_name_tl
2362             \msg_warning:nnx { zref-clever } { missing-name }
2363             { \l__zrefclever_type_first_label_type_tl }

```

```

2364     }
2365   }
2366 }
2367 {
2368   \prop_get:cVNF
2369   {
2370     l__zrefclever_type_
2371     \l__zrefclever_type_first_label_type_tl _options_prop
2372   }
2373   \l__zrefclever_name_format_tl
2374   \l__zrefclever_type_name_tl
2375   {
2376     \prop_get:cVNF
2377     {
2378       l__zrefclever_type_
2379       \l__zrefclever_type_first_label_type_tl _options_prop
2380     }
2381     \l__zrefclever_name_format_fallback_tl
2382     \l__zrefclever_type_name_tl
2383     {
2384       \__zrefclever_get_type_transl:xxxNF
2385       { \l__zrefclever_ref_language_tl }
2386       { \l__zrefclever_type_first_label_type_tl }
2387       { \l__zrefclever_name_format_tl }
2388       \l__zrefclever_type_name_tl
2389       {
2390         \__zrefclever_get_type_transl:xxxNF
2391         { \l__zrefclever_ref_language_tl }
2392         { \l__zrefclever_type_first_label_type_tl }
2393         { \l__zrefclever_name_format_fallback_tl }
2394         \l__zrefclever_type_name_tl
2395         {
2396           \tl_clear:N \l__zrefclever_type_name_tl
2397           \msg_warning:nnx { zref-clever }
2398             { missing-name }
2399             { \l__zrefclever_type_first_label_type_tl }
2400         }
2401       }
2402     }
2403   }
2404 }
2405 }
2406 }
2407
2408 % Signal whether the type name is to be included in the hyperlink or not.
2409 \bool_lazy_any:nTF
2410 {
2411   { ! \l__zrefclever_use_hyperref_bool }
2412   { \l__zrefclever_link_star_bool }
2413   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2414   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2415 }
2416 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2417 {

```



```

2418 \bool_lazy_any:nTF
2419 {
2420   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2421   {
2422     \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2423     \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2424   }
2425   {
2426     \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2427     \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2428     \l__zrefclever_typeset_last_bool &&
2429     \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2430   }
2431 }
2432 { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2433 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2434 }
2435 }

```

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

\\_\_zrefclever\_extract\_url:n A convenience auxiliary function for extraction of the url / urluse property, provided by the zref-xr module.

```

2436 \cs_new:Npn \__zrefclever_extract_url:n #1
2437 {
2438   \zref@ifpropundefined { urluse }
2439   { \zref@extractdefault {#1} { url } { \c_empty_tl } }
2440   {
2441     \zref@ifrefcontainsprop {#1} { urluse }
2442     { \zref@extractdefault {#1} { urluse } { \c_empty_tl } }
2443     { \zref@extractdefault {#1} { url } { \c_empty_tl } }
2444   }
2445 }
2446 \cs_generate_variant:Nn \__zrefclever_extract_url:n { V }

```

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

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

```

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

2447 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2448 {
2449   \tl_set:Nx \l__zrefclever_label_extdoc_a_tl
2450   { \zref@extractdefault {#1} { externaldocument } { \c_empty_tl } }
2451   \tl_set:Nx \l__zrefclever_label_extdoc_b_tl
2452   { \zref@extractdefault {#2} { externaldocument } { \c_empty_tl } }
2453
2454   \tl_if_eq:NNT

```

```

2455 \l__zrefclever_label_extdoc_a_tl
2456 \l__zrefclever_label_extdoc_b_tl
2457 {
2458   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2459   {
2460     \exp_args:Nxx \tl_if_eq:nnT
2461     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2462     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2463     {
2464       \int_compare:nNnTF
2465       { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2466       =
2467       { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2468       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2469       {
2470         \int_compare:nNnT
2471         { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2472         =
2473         { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2474         {
2475           \bool_set_true:N
2476           \l__zrefclever_next_maybe_range_bool
2477           \bool_set_true:N
2478           \l__zrefclever_next_is_same_bool
2479         }
2480       }
2481     }
2482   }
2483   {
2484     \exp_args:Nxx \tl_if_eq:nnT
2485     { \zref@extractdefault {#1} { zc@counter } { } }
2486     { \zref@extractdefault {#2} { zc@counter } { } }
2487     {
2488       \exp_args:Nxx \tl_if_eq:nnT
2489       { \zref@extractdefault {#1} { zc@enclval } { } }
2490       { \zref@extractdefault {#2} { zc@enclval } { } }
2491       {
2492         \int_compare:nNnTF
2493         { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2494         =
2495         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2496         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2497         {
2498           \int_compare:nNnT
2499           { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2500           =
2501           { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2502           {
2503             \bool_set_true:N
2504             \l__zrefclever_next_maybe_range_bool
2505             \bool_set_true:N
2506             \l__zrefclever_next_is_same_bool
2507           }
2508         }
2509       }
2510     }
2511   }

```

```

2509         }
2510     }
2511 }
2512 }
2513 }

```

(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  $\langle option \rangle$  as argument, and store the retrieved value in  $\langle tl\ variable \rangle$ . 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 {\langle option \rangle} {\langle tl variable \rangle}
2514 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2515 {
2516   % First attempt: general options.
2517   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2518   {
2519     % If not found, try type specific options.
2520     \bool_lazy_all:nTF
2521     {
2522       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2523       {
2524         \prop_if_exist_p:c
2525         {
2526           l__zrefclever_type_
2527           \l__zrefclever_label_type_a_tl _options_prop
2528         }
2529       }
2530       {
2531         \prop_if_in_p:cn
2532         {
2533           l__zrefclever_type_
2534           \l__zrefclever_label_type_a_tl _options_prop
2535         }
2536         {#1}
2537       }
2538     }
2539     {
2540       \prop_get:cnN
2541       {
2542         l__zrefclever_type_
2543         \l__zrefclever_label_type_a_tl _options_prop
2544       }
2545       {#1} #2

```

```

2546     }
2547   {
2548     % If not found, try type specific translations.
2549     \__zrefclever_get_type_transl:xnNF
2550     { \l__zrefclever_ref_language_tl }
2551     { \l__zrefclever_label_type_a_tl }
2552     {#1} #2
2553     {
2554       % If not found, try default translations.
2555       \__zrefclever_get_default_transl:xnNF
2556       { \l__zrefclever_ref_language_tl }
2557       {#1} #2
2558       {
2559         % If not found, try fallback.
2560         \__zrefclever_get_fallback_transl:nNF {#1} #2
2561         {
2562           \tl_clear:N #2
2563           \msg_warning:nnn { zref-clever }
2564             { missing-string } {#1}
2565         }
2566       }
2567     }
2568   }
2569 }
2570 }

```

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

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2571 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2572 {
2573   % First attempt: general options.
2574   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2575   {
2576     % If not found, try type specific options.
2577     \bool_lazy_and:nnTF
2578       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2579       {
2580         \prop_if_exist_p:c
2581         {
2582           l__zrefclever_type_
2583           \l__zrefclever_label_type_a_tl _options_prop
2584         }
2585       }
2586     {
2587       \prop_get:cnNF
2588       {
2589         l__zrefclever_type_
2590         \l__zrefclever_label_type_a_tl _options_prop
2591       }
2592       {#1} #2
2593       { \tl_clear:N #2 }
2594     }
2595   }

```

```

2596     }
2597 }

```

(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 `memoir` class and the `appendix` package define the `appendices` and `subappendices` environments, which provide for a way for the appendix to “end”, but in this case, of course, we can hook into the environment instead.

```

2598 \AddToHook { cmd / appendix / before }
2599 {
2600   \zcsetup
2601   {
2602     countertype =
2603     {
2604       chapter      = appendix ,
2605       section      = appendix ,
2606       subsection   = appendix ,
2607       subsubsection = appendix ,
2608     }
2609   }
2610 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmds` (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (`##`) the patch to add the hook, if it needs to be done with the `\scantokens`

method, may fail noisily (see <https://tex.stackexchange.com/q/617905>, thanks Phelype Oleinik). The 2021-11-15 kernel release should already handle this gracefully. In the meantime, given we cannot really expect to know what `\appendix` may contain in general, since it potentially gets redefined in quite a number of classes and packages, a user facing workaround may be needed in case of trouble. Phelype Oleinik recommends activating/providing the generic hook in question, so that `ltxcmdhooks` considers the patch as already done, and do the patch ourselves with `etoolbox` (<https://tex.stackexchange.com/a/617998>). Like so:

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

## 9.2 appendix package

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

```
2611 \AddToHook { begindocument }
2612 {
2613   \@ifpackageloaded { appendix }
2614   {
2615     \AddToHook { env / appendices / begin }
2616     {
2617       \zcsetup
2618       {
2619         countertype =
2620         {
2621           chapter      = appendix ,
2622           section      = appendix ,
2623           subsection   = appendix ,
2624           subsubsection = appendix ,
2625         }
2626       }
2627     }
2628     \AddToHook { env / subappendices / begin }
2629     {
2630       \zcsetup
2631       {
2632         countertype =
2633         {
2634           chapter      = subappendix ,
2635           section      = subappendix ,
2636           subsection   = subappendix ,
2637           subsubsection = subappendix ,
2638         }
2639       }
2640     }
```

```

2641     \msg_info:nnn { zref-clever } { compat-package } { appendix }
2642   }
2643   {}
2644 }

```

### 9.3 listings package

```

2645 \AddToHook { begindocument }
2646 {
2647   \@ifpackageloaded { listings }
2648   {
2649     \zcsetup
2650     {
2651       countertype =
2652       {
2653         lstlisting = listing ,
2654         lstnumber = line ,
2655       } ,
2656       counterresetby = { lstnumber = lstlisting } ,
2657     }
2658     \lst@AddToHook { Init }
2659     {

```

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

```

2660       \tl_if_empty:NF \lst@label
2661       { \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.

```

2662       \zcsetup { currentcounter = lstnumber }
2663     }
2664     \msg_info:nnn { zref-clever } { compat-package } { listings }
2665   }
2666   {}
2667 }

```

### 9.4 enumitem package

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

The only real reason to “renew” `enumerate` itself is to change `{\max-depth}`. `\renewlist` *hard-codes* `max-depth` in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But `\renewlist`

also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from `zref-clever`'s perspective. Since the first four are defined by the kernel and already setup for `zref-clever` by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```

2668 \AddToHook { begindocument }
2669 {
2670   \@ifpackageloaded { enumitem }
2671   {
2672     \int_set:Nn \l_tmpa_int { 5 }
2673     \bool_while_do:nn
2674     {
2675       \cs_if_exist_p:c
2676       { c@ enum \int_to_roman:n { \l_tmpa_int } }
2677     }
2678     {
2679       \exp_args:Nx \zcsetup
2680       {
2681         counterresetby =
2682         {
2683           enum \int_to_roman:n { \l_tmpa_int } =
2684           enum \int_to_roman:n { \l_tmpa_int - 1 }
2685         } ,
2686         countertype =
2687         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
2688       }
2689       \int_incr:N \l_tmpa_int
2690     }
2691     \int_compare:nNnT { \l_tmpa_int } > { 5 }
2692     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
2693   }
2694   {}
2695 }
2696 </package>

```

## 10 Dictionaries

### 10.1 English

```

2697 <package>\zcDeclareLanguage { english }
2698 <package>\zcDeclareLanguageAlias { american } { english }
2699 <package>\zcDeclareLanguageAlias { australian } { english }
2700 <package>\zcDeclareLanguageAlias { british } { english }
2701 <package>\zcDeclareLanguageAlias { canadian } { english }
2702 <package>\zcDeclareLanguageAlias { newzealand } { english }
2703 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2704 <package>\zcDeclareLanguageAlias { USenglish } { english }
2705 <*dict-english>

2706 namesep = {\nobreakspace} ,
2707 pairsep = {\~and\nobreakspace} ,
2708 listsep = {,~} ,

```



```

2709 lastsep = {\~and\nobreakspace} ,
2710 tpairsep = {\~and\nobreakspace} ,
2711 tlistsep = {\~} ,
2712 tlastsep = {\~and\nobreakspace} ,
2713 notesep = {\~} ,
2714 rangesep = {\~to\nobreakspace} ,
2715
2716 type = part ,
2717   Name-sg = Part ,
2718   name-sg = part ,
2719   Name-pl = Parts ,
2720   name-pl = parts ,
2721
2722 type = chapter ,
2723   Name-sg = Chapter ,
2724   name-sg = chapter ,
2725   Name-pl = Chapters ,
2726   name-pl = chapters ,
2727
2728 type = section ,
2729   Name-sg = Section ,
2730   name-sg = section ,
2731   Name-pl = Sections ,
2732   name-pl = sections ,
2733
2734 type = paragraph ,
2735   Name-sg = Paragraph ,
2736   name-sg = paragraph ,
2737   Name-pl = Paragraphs ,
2738   name-pl = paragraphs ,
2739   Name-sg-ab = Par. ,
2740   name-sg-ab = par. ,
2741   Name-pl-ab = Par. ,
2742   name-pl-ab = par. ,
2743
2744 type = appendix ,
2745   Name-sg = Appendix ,
2746   name-sg = appendix ,
2747   Name-pl = Appendices ,
2748   name-pl = appendices ,
2749
2750 type = subappendix ,
2751   Name-sg = Appendix ,
2752   name-sg = appendix ,
2753   Name-pl = Appendices ,
2754   name-pl = appendices ,
2755
2756 type = page ,
2757   Name-sg = Page ,
2758   name-sg = page ,
2759   Name-pl = Pages ,
2760   name-pl = pages ,
2761   name-sg-ab = p. ,
2762   name-pl-ab = pp. ,

```

```

2763
2764 type = line ,
2765     Name-sg = Line ,
2766     name-sg = line ,
2767     Name-pl = Lines ,
2768     name-pl = lines ,
2769
2770 type = figure ,
2771     Name-sg = Figure ,
2772     name-sg = figure ,
2773     Name-pl = Figures ,
2774     name-pl = figures ,
2775     Name-sg-ab = Fig. ,
2776     name-sg-ab = fig. ,
2777     Name-pl-ab = Figs. ,
2778     name-pl-ab = figs. ,
2779
2780 type = table ,
2781     Name-sg = Table ,
2782     name-sg = table ,
2783     Name-pl = Tables ,
2784     name-pl = tables ,
2785
2786 type = item ,
2787     Name-sg = Item ,
2788     name-sg = item ,
2789     Name-pl = Items ,
2790     name-pl = items ,
2791
2792 type = footnote ,
2793     Name-sg = Footnote ,
2794     name-sg = footnote ,
2795     Name-pl = Footnotes ,
2796     name-pl = footnotes ,
2797
2798 type = note ,
2799     Name-sg = Note ,
2800     name-sg = note ,
2801     Name-pl = Notes ,
2802     name-pl = notes ,
2803
2804 type = equation ,
2805     Name-sg = Equation ,
2806     name-sg = equation ,
2807     Name-pl = Equations ,
2808     name-pl = equations ,
2809     Name-sg-ab = Eq. ,
2810     name-sg-ab = eq. ,
2811     Name-pl-ab = Eqs. ,
2812     name-pl-ab = eqs. ,
2813     refpre-in = {} ,
2814     refpos-in = {} ,
2815
2816 type = theorem ,

```

```

2817 Name-sg = Theorem ,
2818 name-sg = theorem ,
2819 Name-pl = Theorems ,
2820 name-pl = theorems ,
2821
2822 type = lemma ,
2823 Name-sg = Lemma ,
2824 name-sg = lemma ,
2825 Name-pl = Lemmas ,
2826 name-pl = lemmas ,
2827
2828 type = corollary ,
2829 Name-sg = Corollary ,
2830 name-sg = corollary ,
2831 Name-pl = Corollaries ,
2832 name-pl = corollaries ,
2833
2834 type = proposition ,
2835 Name-sg = Proposition ,
2836 name-sg = proposition ,
2837 Name-pl = Propositions ,
2838 name-pl = propositions ,
2839
2840 type = definition ,
2841 Name-sg = Definition ,
2842 name-sg = definition ,
2843 Name-pl = Definitions ,
2844 name-pl = definitions ,
2845
2846 type = proof ,
2847 Name-sg = Proof ,
2848 name-sg = proof ,
2849 Name-pl = Proofs ,
2850 name-pl = proofs ,
2851
2852 type = result ,
2853 Name-sg = Result ,
2854 name-sg = result ,
2855 Name-pl = Results ,
2856 name-pl = results ,
2857
2858 type = remark ,
2859 Name-sg = Remark ,
2860 name-sg = remark ,
2861 Name-pl = Remarks ,
2862 name-pl = remarks ,
2863
2864 type = example ,
2865 Name-sg = Example ,
2866 name-sg = example ,
2867 Name-pl = Examples ,
2868 name-pl = examples ,
2869
2870 type = algorithm ,

```

```

2871 Name-sg = Algorithm ,
2872 name-sg = algorithm ,
2873 Name-pl = Algorithms ,
2874 name-pl = algorithms ,
2875
2876 type = listing ,
2877 Name-sg = Listing ,
2878 name-sg = listing ,
2879 Name-pl = Listings ,
2880 name-pl = listings ,
2881
2882 type = exercise ,
2883 Name-sg = Exercise ,
2884 name-sg = exercise ,
2885 Name-pl = Exercises ,
2886 name-pl = exercises ,
2887
2888 type = solution ,
2889 Name-sg = Solution ,
2890 name-sg = solution ,
2891 Name-pl = Solutions ,
2892 name-pl = solutions ,
2893 </dict-english>

```

## 10.2 German

```

2894 <package>\zcDeclareLanguage { german }
2895 <package>\zcDeclareLanguageAlias { austrian      } { german }
2896 <package>\zcDeclareLanguageAlias { germanb       } { german }
2897 <package>\zcDeclareLanguageAlias { ngerman        } { german }
2898 <package>\zcDeclareLanguageAlias { naustrian      } { german }
2899 <package>\zcDeclareLanguageAlias { nswissgerman   } { german }
2900 <package>\zcDeclareLanguageAlias { swissgerman    } { german }
2901 <*dict-german>
2902
2902 namesep = {\nobreakspace} ,
2903 pairsep  = {\simund\nobreakspace} ,
2904 listsep  = {,~} ,
2905 lastsep  = {\simund\nobreakspace} ,
2906 tpairsep = {\simund\nobreakspace} ,
2907 tlistsep = {,~} ,
2908 tlastsep = {\simund\nobreakspace} ,
2909 notesep  = {~} ,
2910 rangesep = {\simbis\nobreakspace} ,
2911
2912 type = part ,
2913 Name-sg = Teil ,
2914 name-sg = Teil ,
2915 Name-pl = Teile ,
2916 name-pl = Teile ,
2917
2918 type = chapter ,
2919 Name-sg = Kapitel ,
2920 name-sg = Kapitel ,
2921 Name-pl = Kapitel ,

```

```

2922     name-pl = Kapitel ,
2923
2924 type = section ,
2925     Name-sg = Abschnitt ,
2926     name-sg = Abschnitt ,
2927     Name-pl = Abschnitte ,
2928     name-pl = Abschnitte ,
2929
2930 type = paragraph ,
2931     Name-sg = Absatz ,
2932     name-sg = Absatz ,
2933     Name-pl = Absätze ,
2934     name-pl = Absätze ,
2935
2936 type = appendix ,
2937     Name-sg = Anhang ,
2938     name-sg = Anhang ,
2939     Name-pl = Anhänge ,
2940     name-pl = Anhänge ,
2941
2942 type = subappendix ,
2943     Name-sg = Anhang ,
2944     name-sg = Anhang ,
2945     Name-pl = Anhänge ,
2946     name-pl = Anhänge ,
2947
2948 type = page ,
2949     Name-sg = Seite ,
2950     name-sg = Seite ,
2951     Name-pl = Seiten ,
2952     name-pl = Seiten ,
2953
2954 type = line ,
2955     Name-sg = Zeile ,
2956     name-sg = Zeile ,
2957     Name-pl = Zeilen ,
2958     name-pl = Zeilen ,
2959
2960 type = figure ,
2961     Name-sg = Abbildung ,
2962     name-sg = Abbildung ,
2963     Name-pl = Abbildungen ,
2964     name-pl = Abbildungen ,
2965     Name-sg-ab = Abb. ,
2966     name-sg-ab = Abb. ,
2967     Name-pl-ab = Abb. ,
2968     name-pl-ab = Abb. ,
2969
2970 type = table ,
2971     Name-sg = Tabelle ,
2972     name-sg = Tabelle ,
2973     Name-pl = Tabellen ,
2974     name-pl = Tabellen ,
2975

```

```

2976 type = item ,
2977     Name-sg = Punkt ,
2978     name-sg = Punkt ,
2979     Name-pl = Punkte ,
2980     name-pl = Punkte ,
2981
2982 type = footnote ,
2983     Name-sg = Fußnote ,
2984     name-sg = Fußnote ,
2985     Name-pl = Fußnoten ,
2986     name-pl = Fußnoten ,
2987
2988 type = note ,
2989     Name-sg = Anmerkung ,
2990     name-sg = Anmerkung ,
2991     Name-pl = Anmerkungen ,
2992     name-pl = Anmerkungen ,
2993
2994 type = equation ,
2995     Name-sg = Gleichung ,
2996     name-sg = Gleichung ,
2997     Name-pl = Gleichungen ,
2998     name-pl = Gleichungen ,
2999     refpre-in = {()} ,
3000     refpos-in = {} ,
3001
3002 type = theorem ,
3003     Name-sg = Theorem ,
3004     name-sg = Theorem ,
3005     Name-pl = Theoreme ,
3006     name-pl = Theoreme ,
3007
3008 type = lemma ,
3009     Name-sg = Lemma ,
3010     name-sg = Lemma ,
3011     Name-pl = Lemmata ,
3012     name-pl = Lemmata ,
3013
3014 type = corollary ,
3015     Name-sg = Korollar ,
3016     name-sg = Korollar ,
3017     Name-pl = Korollare ,
3018     name-pl = Korollare ,
3019
3020 type = proposition ,
3021     Name-sg = Satz ,
3022     name-sg = Satz ,
3023     Name-pl = Sätze ,
3024     name-pl = Sätze ,
3025
3026 type = definition ,
3027     Name-sg = Definition ,
3028     name-sg = Definition ,
3029     Name-pl = Definitionen ,

```

```

3030   name-pl = Definitionen ,
3031
3032   type = proof ,
3033     Name-sg = Beweis ,
3034     name-sg = Beweis ,
3035     Name-pl = Beweise ,
3036     name-pl = Beweise ,
3037
3038   type = result ,
3039     Name-sg = Ergebnis ,
3040     name-sg = Ergebnis ,
3041     Name-pl = Ergebnisse ,
3042     name-pl = Ergebnisse ,
3043
3044   type = remark ,
3045     Name-sg = Bemerkung ,
3046     name-sg = Bemerkung ,
3047     Name-pl = Bemerkungen ,
3048     name-pl = Bemerkungen ,
3049
3050   type = example ,
3051     Name-sg = Beispiel ,
3052     name-sg = Beispiel ,
3053     Name-pl = Beispiele ,
3054     name-pl = Beispiele ,
3055
3056   type = algorithm ,
3057     Name-sg = Algorithmus ,
3058     name-sg = Algorithmus ,
3059     Name-pl = Algorithmen ,
3060     name-pl = Algorithmen ,
3061
3062   type = listing ,
3063     Name-sg = Listing ,
3064     name-sg = Listing ,
3065     Name-pl = Listings ,
3066     name-pl = Listings ,
3067
3068   type = exercise ,
3069     Name-sg = Übungsaufgabe ,
3070     name-sg = Übungsaufgabe ,
3071     Name-pl = Übungsaufgaben ,
3072     name-pl = Übungsaufgaben ,
3073
3074   type = solution ,
3075     Name-sg = Lösung ,
3076     name-sg = Lösung ,
3077     Name-pl = Lösungen ,
3078     name-pl = Lösungen ,
3079 </dict-german>

```

## 10.3 French

```

3080 <package>\zcDeclareLanguage { french }
3081 <package>\zcDeclareLanguageAlias { acadian } { french }

```

```

3082 <package>\zcDeclareLanguageAlias { canadien } { french }
3083 <package>\zcDeclareLanguageAlias { francais } { french }
3084 <package>\zcDeclareLanguageAlias { frenchb } { french }
3085 <*dict-french>

3086 namesep = {\nobreakspace} ,
3087 pairsep = {\simet\nobreakspace} ,
3088 listsep = {,~} ,
3089 lastsep = {\simet\nobreakspace} ,
3090 tpairsep = {\simet\nobreakspace} ,
3091 tlistsep = {,~} ,
3092 tlastsep = {\simet\nobreakspace} ,
3093 notesep = {~} ,
3094 rangesep = {\~à\nobreakspace} ,
3095
3096 type = part ,
3097   Name-sg = Partie ,
3098   name-sg = partie ,
3099   Name-pl = Parties ,
3100   name-pl = parties ,
3101
3102 type = chapter ,
3103   Name-sg = Chapitre ,
3104   name-sg = chapitre ,
3105   Name-pl = Chapitres ,
3106   name-pl = chapitres ,
3107
3108 type = section ,
3109   Name-sg = Section ,
3110   name-sg = section ,
3111   Name-pl = Sections ,
3112   name-pl = sections ,
3113
3114 type = paragraph ,
3115   Name-sg = Paragraphe ,
3116   name-sg = paragraphe ,
3117   Name-pl = Paragraphes ,
3118   name-pl = paragraphes ,
3119
3120 type = appendix ,
3121   Name-sg = Annexe ,
3122   name-sg = annexe ,
3123   Name-pl = Annexes ,
3124   name-pl = annexes ,
3125
3126 type = subappendix ,
3127   Name-sg = Annexe ,
3128   name-sg = annexe ,
3129   Name-pl = Annexes ,
3130   name-pl = annexes ,
3131
3132 type = page ,
3133   Name-sg = Page ,
3134   name-sg = page ,

```



```

3135     Name-pl = Pages ,
3136     name-pl = pages ,
3137
3138     type = line ,
3139     Name-sg = Ligne ,
3140     name-sg = ligne ,
3141     Name-pl = Lignes ,
3142     name-pl = lignes ,
3143
3144     type = figure ,
3145     Name-sg = Figure ,
3146     name-sg = figure ,
3147     Name-pl = Figures ,
3148     name-pl = figures ,
3149
3150     type = table ,
3151     Name-sg = Table ,
3152     name-sg = table ,
3153     Name-pl = Tables ,
3154     name-pl = tables ,
3155
3156     type = item ,
3157     Name-sg = Point ,
3158     name-sg = point ,
3159     Name-pl = Points ,
3160     name-pl = points ,
3161
3162     type = footnote ,
3163     Name-sg = Note ,
3164     name-sg = note ,
3165     Name-pl = Notes ,
3166     name-pl = notes ,
3167
3168     type = note ,
3169     Name-sg = Note ,
3170     name-sg = note ,
3171     Name-pl = Notes ,
3172     name-pl = notes ,
3173
3174     type = equation ,
3175     Name-sg = Équation ,
3176     name-sg = équation ,
3177     Name-pl = Équations ,
3178     name-pl = équations ,
3179     refpre-in = {()} ,
3180     refpos-in = {} } ,
3181
3182     type = theorem ,
3183     Name-sg = Théorème ,
3184     name-sg = théorème ,
3185     Name-pl = Théorèmes ,
3186     name-pl = théorèmes ,
3187
3188     type = lemma ,

```

```

3189     Name-sg = Lemme ,
3190     name-sg = lemme ,
3191     Name-pl = Lemmes ,
3192     name-pl = lemmes ,
3193
3194     type = corollary ,
3195     Name-sg = Corollaire ,
3196     name-sg = corollaire ,
3197     Name-pl = Corollaires ,
3198     name-pl = corollaires ,
3199
3200     type = proposition ,
3201     Name-sg = Proposition ,
3202     name-sg = proposition ,
3203     Name-pl = Propositions ,
3204     name-pl = propositions ,
3205
3206     type = definition ,
3207     Name-sg = Définition ,
3208     name-sg = définition ,
3209     Name-pl = Définitions ,
3210     name-pl = définitions ,
3211
3212     type = proof ,
3213     Name-sg = Démonstration ,
3214     name-sg = démonstration ,
3215     Name-pl = Démonstrations ,
3216     name-pl = démonstrations ,
3217
3218     type = result ,
3219     Name-sg = Résultat ,
3220     name-sg = résultat ,
3221     Name-pl = Résultats ,
3222     name-pl = résultats ,
3223
3224     type = remark ,
3225     Name-sg = Remarque ,
3226     name-sg = remarque ,
3227     Name-pl = Remarques ,
3228     name-pl = remarques ,
3229
3230     type = example ,
3231     Name-sg = Exemple ,
3232     name-sg = exemple ,
3233     Name-pl = Exemples ,
3234     name-pl = exemples ,
3235
3236     type = algorithm ,
3237     Name-sg = Algorithme ,
3238     name-sg = algorithme ,
3239     Name-pl = Algorithmes ,
3240     name-pl = algorithmes ,
3241
3242     type = listing ,

```

```

3243 Name-sg = Liste ,
3244 name-sg = liste ,
3245 Name-pl = Listes ,
3246 name-pl = listes ,
3247
3248 type = exercise ,
3249 Name-sg = Exercice ,
3250 name-sg = exercice ,
3251 Name-pl = Exercices ,
3252 name-pl = exercices ,
3253
3254 type = solution ,
3255 Name-sg = Solution ,
3256 name-sg = solution ,
3257 Name-pl = Solutions ,
3258 name-pl = solutions ,
3259 </dict-french>

```

## 10.4 Portuguese

```

3260 <package>\zcDeclareLanguage { portuguese }
3261 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3262 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
3263 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3264 <*dict-portuguese>
3265 namesep = {\nobreakspace} ,
3266 pairsep = {\~e\nobreakspace} ,
3267 listsep = {\~,~} ,
3268 lastsep = {\~e\nobreakspace} ,
3269 tpairsep = {\~e\nobreakspace} ,
3270 tlistsep = {\~,~} ,
3271 tlastsep = {\~e\nobreakspace} ,
3272 notesep = {\~} ,
3273 rangesep = {\~a\nobreakspace} ,
3274
3275 type = part ,
3276 Name-sg = Parte ,
3277 name-sg = parte ,
3278 Name-pl = Partes ,
3279 name-pl = partes ,
3280
3281 type = chapter ,
3282 Name-sg = Capítulo ,
3283 name-sg = capítulo ,
3284 Name-pl = Capítulos ,
3285 name-pl = capítulos ,
3286
3287 type = section ,
3288 Name-sg = Seção ,
3289 name-sg = seção ,
3290 Name-pl = Seções ,
3291 name-pl = seções ,
3292
3293 type = paragraph ,

```

```

3294     Name-sg = Parágrafo ,
3295     name-sg = parágrafo ,
3296     Name-pl = Parágrafos ,
3297     name-pl = parágrafos ,
3298     Name-sg-ab = Par. ,
3299     name-sg-ab = par. ,
3300     Name-pl-ab = Par. ,
3301     name-pl-ab = par. ,
3302
3303     type = appendix ,
3304     Name-sg = Apêndice ,
3305     name-sg = apêndice ,
3306     Name-pl = Apêndices ,
3307     name-pl = apêndices ,
3308
3309     type = subappendix ,
3310     Name-sg = Apêndice ,
3311     name-sg = apêndice ,
3312     Name-pl = Apêndices ,
3313     name-pl = apêndices ,
3314
3315     type = page ,
3316     Name-sg = Página ,
3317     name-sg = página ,
3318     Name-pl = Páginas ,
3319     name-pl = páginas ,
3320     name-sg-ab = p. ,
3321     name-pl-ab = pp. ,
3322
3323     type = line ,
3324     Name-sg = Linha ,
3325     name-sg = linha ,
3326     Name-pl = Linhas ,
3327     name-pl = linhas ,
3328
3329     type = figure ,
3330     Name-sg = Figura ,
3331     name-sg = figura ,
3332     Name-pl = Figuras ,
3333     name-pl = figuras ,
3334     Name-sg-ab = Fig. ,
3335     name-sg-ab = fig. ,
3336     Name-pl-ab = Figs. ,
3337     name-pl-ab = figs. ,
3338
3339     type = table ,
3340     Name-sg = Tabela ,
3341     name-sg = tabela ,
3342     Name-pl = Tabelas ,
3343     name-pl = tabelas ,
3344
3345     type = item ,
3346     Name-sg = Item ,
3347     name-sg = item ,

```

```

3348     Name-pl = Itens ,
3349     name-pl = itens ,
3350
3351 type = footnote ,
3352     Name-sg = Nota ,
3353     name-sg = nota ,
3354     Name-pl = Notas ,
3355     name-pl = notas ,
3356
3357 type = note ,
3358     Name-sg = Nota ,
3359     name-sg = nota ,
3360     Name-pl = Notas ,
3361     name-pl = notas ,
3362
3363 type = equation ,
3364     Name-sg = Equação ,
3365     name-sg = equação ,
3366     Name-pl = Equações ,
3367     name-pl = equações ,
3368     Name-sg-ab = Eq. ,
3369     name-sg-ab = eq. ,
3370     Name-pl-ab = Eqs. ,
3371     name-pl-ab = eqs. ,
3372     refpre-in = {()},
3373     refpos-in = {}},
3374
3375 type = theorem ,
3376     Name-sg = Teorema ,
3377     name-sg = teorema ,
3378     Name-pl = Teoremas ,
3379     name-pl = teoremas ,
3380
3381 type = lemma ,
3382     Name-sg = Lema ,
3383     name-sg = lema ,
3384     Name-pl = Lemas ,
3385     name-pl = lemas ,
3386
3387 type = corollary ,
3388     Name-sg = Corolário ,
3389     name-sg = corolário ,
3390     Name-pl = Corolários ,
3391     name-pl = corolários ,
3392
3393 type = proposition ,
3394     Name-sg = Proposição ,
3395     name-sg = proposição ,
3396     Name-pl = Proposições ,
3397     name-pl = proposições ,
3398
3399 type = definition ,
3400     Name-sg = Definição ,
3401     name-sg = definição ,

```

```

3402     Name-pl = Definições ,
3403     name-pl = definições ,
3404
3405     type = proof ,
3406     Name-sg = Demonstração ,
3407     name-sg = demonstração ,
3408     Name-pl = Demonstrações ,
3409     name-pl = demonstrações ,
3410
3411     type = result ,
3412     Name-sg = Resultado ,
3413     name-sg = resultado ,
3414     Name-pl = Resultados ,
3415     name-pl = resultados ,
3416
3417     type = remark ,
3418     Name-sg = Observação ,
3419     name-sg = observação ,
3420     Name-pl = Observações ,
3421     name-pl = observações ,
3422
3423     type = example ,
3424     Name-sg = Exemplo ,
3425     name-sg = exemplo ,
3426     Name-pl = Exemplos ,
3427     name-pl = exemplos ,
3428
3429     type = algorithm ,
3430     Name-sg = Algoritmo ,
3431     name-sg = algoritmo ,
3432     Name-pl = Algoritmos ,
3433     name-pl = algoritmos ,
3434
3435     type = listing ,
3436     Name-sg = Listagem ,
3437     name-sg = listagem ,
3438     Name-pl = Listagens ,
3439     name-pl = listagens ,
3440
3441     type = exercise ,
3442     Name-sg = Exercício ,
3443     name-sg = exercício ,
3444     Name-pl = Exercícios ,
3445     name-pl = exercícios ,
3446
3447     type = solution ,
3448     Name-sg = Solução ,
3449     name-sg = solução ,
3450     Name-pl = Soluções ,
3451     name-pl = soluções ,
3452 </dict-portuguese>

```

## 10.5 Spanish

```

3453 <package>\zcDeclareLanguage { spanish }
3454 <*dict-spanish>

3455 namesep = {\nobreakspace} ,
3456 pairsep = {\~y\nobreakspace} ,
3457 listsep = {,~} ,
3458 lastsep = {\~y\nobreakspace} ,
3459 tpairsep = {\~y\nobreakspace} ,
3460 tlistsep = {,~} ,
3461 tlastsep = {\~y\nobreakspace} ,
3462 notesep = {\~} ,
3463 rangesep = {\~a\nobreakspace} ,
3464
3465 type = part ,
3466   Name-sg = Parte ,
3467   name-sg = parte ,
3468   Name-pl = Partes ,
3469   name-pl = partes ,
3470
3471 type = chapter ,
3472   Name-sg = Capítulo ,
3473   name-sg = capítulo ,
3474   Name-pl = Capítulos ,
3475   name-pl = capítulos ,
3476
3477 type = section ,
3478   Name-sg = Sección ,
3479   name-sg = sección ,
3480   Name-pl = Secciones ,
3481   name-pl = secciones ,
3482
3483 type = paragraph ,
3484   Name-sg = Párrafo ,
3485   name-sg = párrafo ,
3486   Name-pl = Párrafos ,
3487   name-pl = párrafos ,
3488
3489 type = appendix ,
3490   Name-sg = Apéndice ,
3491   name-sg = apéndice ,
3492   Name-pl = Apéndices ,
3493   name-pl = apéndices ,
3494
3495 type = subappendix ,
3496   Name-sg = Apéndice ,
3497   name-sg = apéndice ,
3498   Name-pl = Apéndices ,
3499   name-pl = apéndices ,
3500
3501 type = page ,
3502   Name-sg = Página ,
3503   name-sg = página ,
3504   Name-pl = Páginas ,
3505   name-pl = páginas ,

```

```

3506
3507 type = line ,
3508     Name-sg = Línea ,
3509     name-sg = línea ,
3510     Name-pl = Líneas ,
3511     name-pl = líneas ,
3512
3513 type = figure ,
3514     Name-sg = Figura ,
3515     name-sg = figura ,
3516     Name-pl = Figuras ,
3517     name-pl = figuras ,
3518
3519 type = table ,
3520     Name-sg = Cuadro ,
3521     name-sg = cuadro ,
3522     Name-pl = Cuadros ,
3523     name-pl = cuadros ,
3524
3525 type = item ,
3526     Name-sg = Punto ,
3527     name-sg = punto ,
3528     Name-pl = Puntos ,
3529     name-pl = puntos ,
3530
3531 type = footnote ,
3532     Name-sg = Nota ,
3533     name-sg = nota ,
3534     Name-pl = Notas ,
3535     name-pl = notas ,
3536
3537 type = note ,
3538     Name-sg = Nota ,
3539     name-sg = nota ,
3540     Name-pl = Notas ,
3541     name-pl = notas ,
3542
3543 type = equation ,
3544     Name-sg = Ecuación ,
3545     name-sg = ecuación ,
3546     Name-pl = Ecuaciones ,
3547     name-pl = ecuaciones ,
3548     refpre-in = {()} ,
3549     refpos-in = {} ,
3550
3551 type = theorem ,
3552     Name-sg = Teorema ,
3553     name-sg = teorema ,
3554     Name-pl = Teoremas ,
3555     name-pl = teoremas ,
3556
3557 type = lemma ,
3558     Name-sg = Lema ,
3559     name-sg = lema ,

```



```

3560     Name-pl = Lemas ,
3561     name-pl = lemas ,
3562
3563 type = corollary ,
3564     Name-sg = Corolario ,
3565     name-sg = corolario ,
3566     Name-pl = Corolarios ,
3567     name-pl = corolarios ,
3568
3569 type = proposition ,
3570     Name-sg = Proposición ,
3571     name-sg = proposición ,
3572     Name-pl = Proposiciones ,
3573     name-pl = proposiciones ,
3574
3575 type = definition ,
3576     Name-sg = Definición ,
3577     name-sg = definición ,
3578     Name-pl = Definiciones ,
3579     name-pl = definiciones ,
3580
3581 type = proof ,
3582     Name-sg = Demostración ,
3583     name-sg = demostración ,
3584     Name-pl = Demostraciones ,
3585     name-pl = demostraciones ,
3586
3587 type = result ,
3588     Name-sg = Resultado ,
3589     name-sg = resultado ,
3590     Name-pl = Resultados ,
3591     name-pl = resultados ,
3592
3593 type = remark ,
3594     Name-sg = Observación ,
3595     name-sg = observación ,
3596     Name-pl = Observaciones ,
3597     name-pl = observaciones ,
3598
3599 type = example ,
3600     Name-sg = Ejemplo ,
3601     name-sg = ejemplo ,
3602     Name-pl = Ejemplos ,
3603     name-pl = ejemplos ,
3604
3605 type = algorithm ,
3606     Name-sg = Algoritmo ,
3607     name-sg = algoritmo ,
3608     Name-pl = Algoritmos ,
3609     name-pl = algoritmos ,
3610
3611 type = listing ,
3612     Name-sg = Listado ,
3613     name-sg = listado ,

```

```

3614 Name-pl = Listados ,
3615 name-pl = listados ,
3616
3617 type = exercise ,
3618 Name-sg = Ejercicio ,
3619 name-sg = ejercicio ,
3620 Name-pl = Ejercicios ,
3621 name-pl = ejercicios ,
3622
3623 type = solution ,
3624 Name-sg = Solución ,
3625 name-sg = solución ,
3626 Name-pl = Soluciones ,
3627 name-pl = soluciones ,
3628 </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	
\\	107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155
† internal commands:	
\l_zrefclever_current_counter_tl	4
A	
\AddToHook	95, 470, 485, 629, 665, 690, 728, 730, 790, 2598, 2611, 2615, 2628, 2645, 2668
\appendix	69, 70
\appendixname	69
B	
\babelname	675
\babelprovide	12, 22
bool commands:	
\bool_case_true:	2
\bool_if:NTF	302, 313, 633, 637, 1362, 1404, 1612, 1707, 1837, 1859, 1890, 1936, 1977, 2000, 2004, 2010, 2020, 2026, 2182
\bool_if:nTF	63, 1188, 1197, 1206, 1272, 1282, 1306, 1334, 1371, 1478, 1486, 1626, 1634, 1871, 1878, 1885, 2135, 2256
\bool_lazy_all:nTF	2520
\bool_lazy_and:nnTF	1080, 1098, 2331, 2577
\bool_lazy_any:nTF	2409, 2418
\bool_lazy_or:nnTF	1084, 2319
\bool_new:N	265, 506, 507, 532, 556, 565, 572, 573, 586, 587, 606, 607, 783, 784, 1109, 1124, 1518, 1519, 1529, 1535, 1536
\bool_set:Nn	1078
\bool_set_false:N	519, 523, 614, 623, 624, 639, 805, 1264, 1575, 1618, 1632, 1646, 1849, 1975, 1976, 2416, 2433
\bool_set_true:N	322, 513, 514, 518, 524, 613, 618, 619, 794, 799, 1278, 1288, 1292, 1318, 1329, 1358, 1366, 1399, 1408, 1436, 1448, 1583, 1613, 1619, 1623, 1650, 1656, 2432, 2468, 2475, 2477, 2496, 2503, 2505
\bool_until_do:Nn	1304, 1576
\bool_while_do:nn	2673
C	
clist commands:	
\clist_map_inline:nn	846
\counterwithin	4
cs commands:	
\cs_generate_variant:Nn	59, 60, 318, 326, 989, 995, 1240, 2176, 2446
\cs_if_exist:NTF	43, 52, 73
\cs_if_exist_p:N	2675
\cs_new:Npn	41, 50, 61, 71, 82, 2131, 2177, 2436

\cs_new_protected:Npn	266, 319, 327, 333, 454, 984, 990, 1073, 1128, 1170, 1181, 1241, 1456, 1508, 1552, 1714, 1971, 2127, 2129, 2309, 2447, 2514, 2571
\cs_new_protected:Npx	94
\cs_set_eq:NN	98
<b>E</b>	
\endinput	12
exp commands:	
\exp_args:NNe	27, 30
\exp_args:NNnx	256
\exp_args:NnV	294
\exp_args:NNx	99
\exp_args:Nnx	329, 1344, 1381
\exp_args:Nx	276, 2679
\exp_args:Nxx	1224, 1314, 1417, 2460, 2484, 2488
\exp_not:N	58, 1892, 1895, 1906, 1909, 1912, 2141, 2144, 2147, 2156, 2158, 2161, 2164, 2169, 2171, 2189, 2200, 2203, 2205, 2208, 2215, 2222, 2224, 2228, 2231, 2234, 2246, 2249, 2262, 2265, 2268, 2285, 2287, 2290, 2293, 2300, 2302
\exp_not:n	1736, 1752, 1764, 1769, 1792, 1806, 1810, 1822, 1826, 1860, 1861, 1893, 1905, 1910, 1911, 2040, 2053, 2060, 2084, 2096, 2100, 2110, 2114, 2142, 2143, 2145, 2151, 2154, 2157, 2162, 2163, 2165, 2166, 2168, 2170, 2201, 2202, 2204, 2206, 2207, 2209, 2210, 2214, 2226, 2227, 2232, 2233, 2235, 2243, 2247, 2248, 2250, 2263, 2264, 2266, 2279, 2283, 2286, 2291, 2292, 2294, 2295, 2299, 2301
\ExplSyntaxOn	12, 278
<b>F</b>	
file commands:	
\file_get:nnNTF	276
\fmtversion	3
<b>G</b>	
group commands:	
\group_begin:	97, 268, 321, 973, 1075, 1088, 1892, 1909, 2141, 2144, 2161, 2164, 2200, 2205, 2208, 2224, 2231, 2246, 2262, 2265, 2290, 2293
\group_end:	100, 316, 324, 981, 1091, 1106, 1906, 1912, 2156, 2158, 2169, 2171, 2203, 2215, 2222, 2228, 2234, 2249, 2285, 2287, 2300, 2302
<b>I</b>	
\IfBooleanTF	1112
\IfFormatAtLeastTF	3, 4
\input	11, 12
int commands:	
\int_case:nnTF	1717, 1745, 1777, 1939, 2033, 2072
\int_compare:nNnTF	1228, 1319, 1349, 1386, 1421, 1437, 1464, 1466, 1510, 1678, 1732, 1766, 1928, 1930, 1988, 2013, 2057, 2464, 2470, 2492, 2498, 2691
\int_compare_p:nNn	1480, 1488, 2323, 2334, 2429
\int_eval:n	94
\int_incr:N	1343, 1380, 1966, 2003, 2005, 2019, 2021, 2025, 2027, 2125, 2689
\int_new:N	1125, 1126, 1520, 1521, 1532, 1533
\int_set:Nn	1465, 1467, 1471, 1474, 2672
\int_to_roman:n	2676, 2683, 2684, 2687
\int_use:N	37, 39, 54
\int_zero:N	1340, 1377, 1458, 1459, 1561, 1562, 1563, 1564, 1965, 1967, 1968, 2120, 2121
\l_tmpa_int	1377, 1380, 1390, 2672, 2676, 2683, 2684, 2687, 2689, 2691
\l_tmpb_int	1340, 1343, 1354
iow commands:	
\iow_char:N	107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155
<b>K</b>	
keys commands:	
\keys_define:nn	30, 339, 351, 368, 382, 461, 489, 496, 508, 533, 542, 557, 566, 574, 588, 600, 608, 641, 648, 686, 733, 775, 778, 785, 795, 806, 814, 842, 868, 892, 902, 913, 934, 946, 996, 1008, 1029, 1052
\keys_set:nn	12, 30, 34, 295, 800, 923, 929, 978, 1076
keyval commands:	
\keyval_parse:nnn	818, 872
<b>L</b>	
\labelformat	3
\languagename	22, 669
<b>M</b>	
\mainbabelname	22, 676
\MessageBreak	10
msg commands:	
\msg_info:nnn	359, 389, 2641, 2664, 2692

\msg_line_context: .....	106, 112, 116, 133, 140, 143, 149, 163, 167, 169, 171, 174, 178
\msg_new:nnn .....	104, 110, 115, 117, 119, 125, 131, 137, 139, 141, 147, 152, 157, 159, 161, 166, 168, 170, 172, 177, 179, 181
\msg_note:nnn .....	298
\msg_warning:nn .....	475, 500, 638, 644, 788, 809
\msg_warning:nnn ...	245, 260, 304, 314, 716, 761, 874, 938, 980, 1020, 1059, 1671, 1844, 2362, 2397, 2563
\msg_warning:nnnn .....	820, 1327, 1364, 1406, 1446
<b>N</b>	
\newcounter .....	4
\NewDocumentCommand .....	240, 250, 922, 924, 971, 1071, 1110
\nobreakspace .....	403, 2706, 2707, 2709, 2710, 2712, 2714, 2902, 2903, 2905, 2906, 2908, 2910, 3086, 3087, 3089, 3090, 3092, 3094, 3265, 3266, 3268, 3269, 3271, 3273, 3455, 3456, 3458, 3459, 3461, 3463
<b>P</b>	
\PackageError .....	7
\pagenumbering .....	6
\pageref .....	35
prg commands:	
\prg_generate_conditional_- variant:Nnn .....	427, 443
\prg_new_protected_conditional:Npnn .....	413, 429, 446
\prg_return_false: .....	423, 425, 439, 441, 452
\prg_return_true: .....	422, 438, 451
\ProcessKeysOptions .....	921
prop commands:	
\prop_get:NnN .....	2540
\prop_get:NnNTF .....	269, 416, 419, 432, 435, 449, 974, 2347, 2368, 2376, 2517, 2574, 2587
\prop_gput:Nnn ....	246, 257, 986, 992
\prop_gput_if_new:Nnn .....	329, 335
\prop_gset_from_keyval:Nn .....	397
\prop_if_exist:NTF .....	281, 926
\prop_if_exist_p:N .....	2524, 2580
\prop_if_in:NnTF .....	27, 244, 254, 711, 756
\prop_if_in_p:Nn .....	64, 2531
\prop_item:Nn .....	30, 65, 258
\prop_new:N .....	239, 287, 396, 813, 867, 898, 927
\prop_put:Nnn .....	458, 909, 961
\prop_remove:Nn .....	457, 908, 953
\providecommand .....	3
\ProvidesExplPackage .....	14
\ProvidesFile .....	11
<b>R</b>	
\refstepcounter .....	3, 71
\renewlist .....	71
\RequirePackage .....	16, 17, 18, 19, 634
<b>S</b>	
\scantokens .....	69
seq commands:	
\seq_clear:N .....	553, 1130
\seq_const_from_clist:Nn .....	185, 193, 206, 218
\seq_gconcat:NNN ...	226, 229, 233, 236
\seq_get_left:NN .....	1586
\seq_gput_right:Nn .....	296, 307
\seq_if_empty:NTF .....	1580
\seq_if_in:NnTF .....	272, 848, 1174
\seq_map_break:n .....	85, 1499, 1502
\seq_map_function:NN .....	1133
\seq_map_indexed_inline:Nn .	19, 1460
\seq_map_inline:Nn .	348, 365, 379, 899, 931, 943, 1005, 1026, 1049, 1496
\seq_map_tokens:Nn .....	67
\seq_new:N .....	225, 232, 264, 541, 841, 1108, 1127, 1517
\seq_pop_left:NN .....	1578
\seq_put_right:Nn .....	850, 1177
\seq_reverse:N .....	547
\seq_set_eq:NN .....	1554
\seq_set_from_clist:Nn ....	546, 1077
\seq_sort:Nn .....	37, 1136
sort commands:	
\sort_return_same: .....	37, 44, 1143, 1148, 1195, 1233, 1235, 1279, 1299, 1324, 1330, 1357, 1367, 1397, 1409, 1442, 1449, 1484, 1499, 1515
\sort_return_swapped: ...	37, 44, 1156, 1204, 1232, 1289, 1298, 1323, 1356, 1398, 1441, 1492, 1502, 1514
str commands:	
\str_case:nnTF .....	692, 737
\str_compare:nNnTF .....	1295
\str_if_eq:nnTF .....	84
\str_if_eq_p:nn .....	2414, 2420, 2422, 2426
\str_new:N .....	647
\str_set:Nn .....	652, 654, 656, 658

## T

TeX and L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> commands:

`\@Alph` ..... 69  
`\@addtoreset` ..... 4  
`\@chapapp` ..... 69  
`\@currentcounter` .... 3, 4, 28, 71, 896  
`\@currentlabel` ..... 3, 71  
`\@elt` ..... 4  
`\@ifl@t@r` ..... 3  
`\@ifpackageloaded` ..... 472, 487,  
     631, 667, 673, 792, 2613, 2647, 2670  
`\@onlypreamble` ..... 249, 263, 983  
`\bbl@loaded` ..... 23  
`\bbl@main@language` ..... 22, 670  
`\c@` ..... 3  
`\c@enumN` ..... 72  
`\c@lstnumber` ..... 71  
`\c@page` ..... 6, 98  
`\cl@` ..... 4  
`\hyper@@link` 58, 1895, 2147, 2189, 2268  
`\lst@AddToHook` ..... 2658  
`\lst@label` ..... 2660, 2661  
`\p@...` ..... 3  
`\zref@addprop` .....  
     ..... 21, 24, 35, 38, 40, 91, 92, 103  
`\zref@default` ..... 58, 2128, 2130  
`\zref@extractdefault` .....  
     ... 1173, 1184, 1186, 1225, 1226,  
     1229, 1231, 1244, 1248, 1252, 1256,  
     1260, 1262, 1315, 1316, 1320, 1322,  
     1345, 1350, 1382, 1394, 1511, 1513,  
     1598, 1603, 1901, 2149, 2152, 2167,  
     2195, 2211, 2274, 2280, 2296, 2439,  
     2442, 2443, 2450, 2452, 2461, 2462,  
     2465, 2467, 2471, 2473, 2485, 2486,  
     2489, 2490, 2493, 2495, 2499, 2501  
`\zref@ifpropundefined` ..... 18, 2438  
`\zref@ifrefcontainsprop` .....  
     ..... 18, 2133, 2184, 2252, 2441  
`\zref@ifrefundefined` .....  
     1138, 1140, 1152, 1615, 1617, 1622,  
     1666, 1841, 1850, 1979, 2179, 2311  
`\ZREF@mainlist` .....  
     ..... 21, 24, 35, 38, 40, 91, 92, 103  
`\zref@newprop` .....  
     ... 5, 7, 20, 22, 25, 36, 39, 87, 89, 102  
`\zref@refused` ..... 1665  
`\zref@wrapper@babel` ..... 33, 1072  
`\textendash` ..... 407  
`\the` ..... 3  
`\thechapter` ..... 69  
`\thelstnumber` ..... 71  
`\thepage` ..... 6, 99  
`\thesection` ..... 69

tl commands:

`\c_empty_tl` 1173, 1184, 1186, 1244,  
     1248, 1252, 1256, 1260, 1262, 1599,  
     1604, 2439, 2442, 2443, 2450, 2452  
`\c_novalue_tl` ..... 904, 948  
`\tl_clear:N` 293, 344, 977, 1001, 1556,  
     1557, 1558, 1559, 1560, 1582, 1961,  
     1962, 1963, 1964, 2002, 2312, 2315,  
     2343, 2361, 2396, 2562, 2593, 2595  
`\tl_gset:Nn` ..... 99  
`\tl_head:N` .....  
     ... 1418, 1419, 1422, 1424, 1438, 1440  
`\tl_if_empty:N` ..... 75, 356,  
     373, 387, 1013, 1034, 1057, 1092,  
     1669, 1839, 2240, 2328, 2345, 2660  
`\tl_if_empty:n` ..... 242,  
     252, 343, 456, 1000, 1788, 1804,  
     1820, 2051, 2082, 2094, 2108, 2314  
`\tl_if_empty_p:N` . 1192, 1193, 1201,  
     1202, 1209, 1210, 1629, 1630, 1637,  
     1639, 2413, 2423, 2427, 2522, 2578  
`\tl_if_empty_p:n` ..... 1274, 1275,  
     1284, 1285, 1310, 1311, 1337, 1374  
`\tl_if_eq:NNTF` 1213, 1268, 1642, 2454  
`\tl_if_eq:NnTF` ..... 1131, 1163,  
     1470, 1473, 1498, 1501, 1590, 2458  
`\tl_if_eq:nnTF` . 1224, 1314, 1344,  
     1381, 1417, 1462, 2460, 2484, 2488  
`\tl_if_novalue:nTF` ..... 907, 951  
`\tl_item:Nn` ..... 1353, 1388  
`\tl_map_break:n` ..... 85, 1347, 1384  
`\tl_map_inline:Nn` ..... 1341, 1378  
`\tl_map_tokens:Nn` ..... 77  
`\tl_new:N` .....  
     . 93, 183, 184, 460, 662, 663, 664,  
     774, 777, 891, 1116, 1117, 1118,  
     1119, 1120, 1121, 1122, 1123, 1522,  
     1523, 1524, 1525, 1526, 1527, 1528,  
     1530, 1531, 1534, 1537, 1538, 1539,  
     1540, 1541, 1542, 1543, 1544, 1545,  
     1546, 1547, 1548, 1549, 1550, 1551  
`\tl_put_left:Nn` .... 1874, 1881, 1921  
`\tl_put_right:Nn` ..... 1734, 1750,  
     1759, 1790, 1801, 1817, 2038, 2049,  
     2080, 2092, 2106, 2329, 2330, 2341  
`\tl_reverse_items:n` .....  
     ..... 1240, 1246, 1250, 1254, 1258  
`\tl_set:Nn` .....  
     . 345, 465, 467, 473, 476, 492, 501,  
     669, 670, 675, 676, 679, 680, 683,  
     696, 704, 713, 718, 741, 749, 758,  
     763, 928, 1002, 1172, 1183, 1185,  
     1243, 1245, 1247, 1249, 1251, 1253,  
     1255, 1257, 1259, 1261, 1426, 1428,

1430, 1432, 1592, 1593, 1596, 1601, 1723, 1725, 1857, 1888, 1992, 1994, 2017, 2325, 2326, 2339, 2449, 2451	
\tl_set_eq:NN . . . . .	1959
\tl_tail:N . . . . .	1427, 1429, 1431, 1433
\l_tmpa_tl . . . . .	279, 295, 1094, 1095
<b>U</b>	
use commands:	
\use:N . . . . .	23
<b>Z</b>	
\zcDeclareLanguage . . . . .	10, 240, 2697, 2894, 3080, 3260, 3453
\zcDeclareLanguageAlias . . . . .	11, 250, 2698, 2699, 2700, 2701, 2702, 2703, 2704, 2895, 2896, 2897, 2898, 2899, 2900, 3081, 3082, 3083, 3084, 3261, 3262, 3263
\zcLanguageSetup . . . . .	9, 11–13, 29, 31, 32, 971
\zcpageref . . . . .	35, 1110
\zceref . . . . .	24, 25, 28, 29, 33, 35–37, 45, 47, 1071, 1113, 1114
\zcRefTypeSetup . . . . .	9, 29, 30, 924
\zcsetup . . . . .	22, 25, 28, 29, 922, 2600, 2617, 2630, 2649, 2662, 2679
\zlabel . . . . .	71, 2661
zrefcheck commands:	
\zrefcheck_zceref_beg_label: . .	1083
\zrefcheck_zceref_end_label_- maybe: . . . . .	1102
\zrefcheck_zceref_run_checks_on_- labels:n . . . . .	1103
zrefclever internal commands:	
\l__zrefclever_abbrev_bool . . . . .	586, 590, 2332
\l__zrefclever_capitalize_bool . .	572, 576, 2320
\l__zrefclever_capitalize_first_- bool . . . . .	573, 582, 2322
\__zrefclever_counter_reset_by:n . . . . .	5, 27, 43, 45, 47, 52, 54, 56, 61
\__zrefclever_counter_reset_by_- aux:nn . . . . .	68, 71
\__zrefclever_counter_reset_by_- aux:nnn . . . . .	78, 82
\l__zrefclever_counter_resetby_- prop . . . . .	5, 27, 64, 65, 867, 879
\l__zrefclever_counter_resetters_- seq . . . . .	4, 5, 27, 67, 841, 848, 851
\l__zrefclever_counter_type_prop . . . . .	3, 26, 27, 30, 813, 825
\l__zrefclever_current_counter_- tl . . . . .	3, 28, 20, 23, 28, 31, 33, 37, 88, 90, 891, 894
\l__zrefclever_current_language_- tl . . . . .	22, 664, 669, 675, 679, 705, 750
\__zrefclever_declare_default_- transl:nnn . . . . .	31, 984, 1015, 1036
\__zrefclever_declare_type_- transl:nnnn . . . . .	31, 984, 1041, 1063
\g__zrefclever_dict_⟨language⟩_prop . . . . .	12
\l__zrefclever_dict_language_tl . . . . . .	183, 270, 274, 277, 284, 290, 297, 299, 305, 308, 330, 336, 417, 420, 433, 436, 975, 1016, 1037, 1042, 1064
\__zrefclever_extract_url:n . . . . .	1897, 2148, 2191, 2270, 2436
\g__zrefclever_fallback_dict_- prop . . . . .	9, 396, 397, 449
\__zrefclever_get_default_- transl:nnN . . . . .	9, 430, 444
\__zrefclever_get_default_- transl:nnNTF . . . . .	16, 429, 2555
\__zrefclever_get_enclosing_- counters:n . . . . .	5, 41, 46, 88
\__zrefclever_get_enclosing_- counters_value:n . . . . .	5, 41, 55, 90
\__zrefclever_get_fallback_- transl:nN . . . . .	447
\__zrefclever_get_fallback_- transl:nNTF . . . . .	17, 445, 2560
\__zrefclever_get_ref:n . . . . .	58, 59, 1737, 1753, 1765, 1770, 1793, 1807, 1811, 1823, 1827, 1862, 1882, 2041, 2054, 2061, 2085, 2097, 2101, 2111, 2115, 2131
\__zrefclever_get_ref_first: . . . . . . . .	58, 59, 62, 1875, 1922, 2177
\__zrefclever_get_ref_font:nN . . . . . . . .	9, 15, 28, 67, 68, 1698, 1700, 1702, 2571
\__zrefclever_get_ref_string:nN . . . . . .	9, 15, 28, 67, 1094, 1567, 1569, 1571, 1680, 1682, 1684, 1686, 1688, 1690, 1692, 1694, 1696, 2514
\__zrefclever_get_type_transl:nnnN . . . . .	9, 414, 428
\__zrefclever_get_type_transl:nnnNTF . . . . .	16, 413, 2355, 2384, 2390, 2549
\l__zrefclever_label_a_tl . . . . .	44, 1522, 1579, 1599, 1615, 1665, 1666, 1672, 1724, 1737, 1753, 1770, 1811, 1827, 1855, 1862, 1979, 1983, 1993, 2018, 2041, 2062, 2101, 2115
\l__zrefclever_label_b_tl . . . . .	44, 1522, 1582, 1587, 1604, 1617, 1622, 1983

\l_zrefclever_label_count_int ..	\l_zrefclever_main_language_tl ..
..... 45, 1520,	..... 22, 663,
1561, 1678, 1717, 1965, 1988, 2125	670, 676, 680, 684, 697, 719, 742, 764
\l_zrefclever_label_enclcnt_a_-	\_zrefclever_name_default: ....
tl ..... 1116, 1243, 1245, 1246,	..... 2127, 2242
1310, 1337, 1378, 1418, 1426, 1427	\l_zrefclever_name_format_-
\l_zrefclever_label_enclcnt_b_-	fallback_tl .....
tl ..... 1116, 1247, 1249, 1250,	.. 1528, 2339, 2343, 2345, 2381, 2393
1311, 1341, 1374, 1419, 1428, 1429	\l_zrefclever_name_format_tl ...
\l_zrefclever_label_enclval_a_-	... 1528, 2325, 2326, 2329, 2330,
tl ..... 1116, 1251, 1253,	2340, 2341, 2352, 2358, 2373, 2387
1254, 1389, 1422, 1430, 1431, 1438	\l_zrefclever_name_in_link_bool
\l_zrefclever_label_enclval_b_-	..... 60,
tl ..... 1116, 1255, 1257,	62, 1528, 1890, 2182, 2416, 2432, 2433
1258, 1353, 1424, 1432, 1433, 1440	\l_zrefclever_namefont_tl 1537,
\l_zrefclever_label_extdoc_a_tl	1699, 1893, 1910, 2201, 2232, 2247
..... 1116, 1259,	\l_zrefclever_nameinlink_str ...
1269, 1274, 1284, 1297, 2449, 2455	..... 647, 652,
\l_zrefclever_label_extdoc_b_tl	654, 656, 658, 2414, 2420, 2422, 2426
..... 1116, 1261,	\l_zrefclever_namesep_tl .....
1270, 1275, 1285, 1296, 2451, 2456	.. 1537, 1681, 2204, 2235, 2243, 2250
\l_zrefclever_label_type_a_tl ..	\l_zrefclever_next_is_same_bool
..... 67, 1116, 1172, 1175,	..... 45, 65, 1532,
1178, 1183, 1192, 1201, 1209, 1214,	1976, 2004, 2020, 2026, 2478, 2506
1470, 1498, 1592, 1596, 1629, 1637,	\l_zrefclever_next_maybe_range_-
1643, 1669, 1726, 1995, 2522, 2527,	bool .....
2534, 2543, 2551, 2578, 2583, 2590	.. 45, 65, 1532, 1849, 1859, 1975,
\l_zrefclever_label_type_b_tl ..	2000, 2010, 2468, 2476, 2496, 2504
..... 1116,	\l_zrefclever_noabbrev_first_-
1185, 1193, 1202, 1210, 1215, 1473,	bool ..... 587, 596, 2336
1501, 1593, 1601, 1630, 1639, 1644	\_zrefclever_page_format_aux: ..
\_zrefclever_label_type_put_-	..... 94, 98
new_right:n .... 36, 37, 1134, 1170	\g_zrefclever_page_format_tl ...
\l_zrefclever_label_types_seq ..	..... 7, 93, 99, 102
.... 37, 1127, 1130, 1174, 1177, 1496	\l_zrefclever_pairsep_tl .....
\_zrefclever_labels_in_sequence:nn	..... 1537, 1685, 1736, 1860
..... 46, 65, 1853, 1982, 2447	\_zrefclever_prop_put_non_-
\g_zrefclever_languages_prop ...	empty:Nnn ..... 17, 454, 824, 878
..... 11, 239, 244, 246, 254,	\_zrefclever_provide_dict_-
257, 258, 269, 416, 432, 711, 756, 974	default_transl:nn 14, 327, 357, 374
\l_zrefclever_last_of_type_bool	\_zrefclever_provide_dict_type_-
..... 45, 1517, 1613, 1618, 1619,	transl:nn ..... 14, 327, 375, 392
1623, 1632, 1647, 1651, 1657, 1707	\_zrefclever_provide_dictionary:n
\l_zrefclever_lastsep_tl . 1537,	..... 9, 12-14,
1689, 1752, 1769, 1792, 1810, 1822	34, 266, 323, 732, 743, 751, 766, 1079
\l_zrefclever_link_star_bool ...	\_zrefclever_provide_dictionary_-
..... 1078, 1108, 2138, 2259, 2412	verbose:n ... 13, 319, 698, 706, 721
\l_zrefclever_listsep_tl .....	\l_zrefclever_range_beg_label_-
... 1537, 1687, 1764, 1806, 2040,	tl ..... 45, 1532, 1560,
2053, 2060, 2084, 2096, 2100, 2110	1765, 1788, 1794, 1804, 1808, 1820,
\l_zrefclever_load_dict_-	1824, 1964, 2002, 2017, 2051, 2055,
verbose_bool ... 265, 302, 313, 322	2082, 2086, 2094, 2098, 2108, 2112
\g_zrefclever_loaded_dictionaries_-	\l_zrefclever_range_count_int ..
seq ..... 264, 273, 296, 307	..... 45,

[1532](#), [1563](#), [1745](#), [1779](#), [1967](#), [2003](#),  
[2014](#), [2019](#), [2025](#), [2033](#), [2074](#), [2120](#)  
\l\_zrefclever\_range\_same\_count\_  
int ..... [45](#),  
[1532](#), [1564](#), [1732](#), [1767](#), [1780](#), [1968](#),  
[2005](#), [2021](#), [2027](#), [2058](#), [2075](#), [2121](#)  
\l\_zrefclever\_rangeseq\_tl .....  
..... [1537](#), [1683](#), [1826](#), [1861](#), [2114](#)  
\\_zrefclever\_ref\_default: .....  
..... [2127](#), [2174](#), [2180](#), [2236](#), [2305](#)  
\l\_zrefclever\_ref\_language\_tl ..  
..... [22](#), [23](#), [662](#), [683](#),  
[696](#), [699](#), [704](#), [707](#), [713](#), [718](#), [722](#),  
[732](#), [741](#), [744](#), [749](#), [752](#), [758](#), [763](#),  
[767](#), [1079](#), [2356](#), [2385](#), [2391](#), [2550](#), [2556](#)  
\c\_zrefclever\_ref\_options\_font\_  
seq ..... [10](#), [15](#), [185](#)  
\c\_zrefclever\_ref\_options\_  
necessarily\_not\_type\_specific\_  
seq ..... [15](#), [185](#), [349](#), [932](#), [1006](#)  
\c\_zrefclever\_ref\_options\_  
necessarily\_type\_specific\_seq  
..... [185](#), [380](#), [1050](#)  
\c\_zrefclever\_ref\_options\_  
possibly\_type\_specific\_seq ..  
..... [15](#), [185](#), [366](#), [1027](#)  
\l\_zrefclever\_ref\_options\_prop ..  
.... [28](#), [30](#), [898](#), [908](#), [909](#), [2517](#), [2574](#)  
\c\_zrefclever\_ref\_options\_  
reference\_seq ..... [185](#), [900](#)  
\c\_zrefclever\_ref\_options\_  
typesetup\_seq ..... [185](#), [944](#)  
\l\_zrefclever\_ref\_property\_tl ..  
..... [18](#),  
[460](#), [465](#), [467](#), [473](#), [476](#), [492](#), [501](#),  
[1131](#), [1163](#), [1590](#), [2133](#), [2153](#), [2167](#),  
[2186](#), [2213](#), [2254](#), [2282](#), [2298](#), [2458](#)  
\l\_zrefclever\_ref\_typeset\_font\_  
tl ..... [774](#), [776](#), [1089](#)  
\l\_zrefclever\_reffont\_in\_tl [1537](#),  
[1703](#), [2145](#), [2165](#), [2209](#), [2266](#), [2294](#)  
\l\_zrefclever\_reffont\_out\_tl ...  
..... [1537](#), [1701](#),  
[2142](#), [2162](#), [2206](#), [2226](#), [2263](#), [2291](#)  
\l\_zrefclever\_refpos\_in\_tl [1537](#),  
[1697](#), [2154](#), [2168](#), [2214](#), [2283](#), [2299](#)  
\l\_zrefclever\_refpos\_out\_tl [1537](#),  
[1693](#), [2157](#), [2170](#), [2227](#), [2286](#), [2301](#)  
\l\_zrefclever\_refpre\_in\_tl [1537](#),  
[1695](#), [2151](#), [2166](#), [2210](#), [2279](#), [2295](#)  
\l\_zrefclever\_refpre\_out\_tl [1537](#),  
[1691](#), [2143](#), [2163](#), [2207](#), [2264](#), [2292](#)  
\l\_zrefclever\_setup\_type\_tl ...  
. [14](#), [183](#), [293](#), [331](#), [344](#), [345](#), [356](#),  
[373](#), [387](#), [928](#), [956](#), [964](#), [977](#), [1001](#),  
[1002](#), [1013](#), [1034](#), [1043](#), [1057](#), [1065](#)  
\l\_zrefclever\_sort\_decided\_bool  
..... [1124](#), [1264](#), [1278](#), [1288](#),  
[1292](#), [1304](#), [1318](#), [1329](#), [1358](#), [1362](#),  
[1366](#), [1400](#), [1404](#), [1408](#), [1436](#), [1448](#)  
\\_zrefclever\_sort\_default:nn ...  
..... [37](#), [1165](#), [1181](#)  
\\_zrefclever\_sort\_default\_  
different\_types:nn .....  
..... [19](#), [36](#), [43](#), [1219](#), [1456](#)  
\\_zrefclever\_sort\_default\_same\_  
type:nn ..... [35](#), [39](#), [1217](#), [1241](#)  
\\_zrefclever\_sort\_labels: .....  
..... [36](#), [37](#), [44](#), [1087](#), [1128](#)  
\\_zrefclever\_sort\_page:nn .....  
..... [44](#), [1164](#), [1508](#)  
\l\_zrefclever\_sort\_prior\_a\_int ..  
..... [1125](#),  
[1458](#), [1464](#), [1465](#), [1471](#), [1481](#), [1489](#)  
\l\_zrefclever\_sort\_prior\_b\_int ..  
..... [1125](#),  
[1459](#), [1466](#), [1467](#), [1474](#), [1482](#), [1490](#)  
\l\_zrefclever\_tlastsep\_tl .....  
..... [1537](#), [1572](#), [1953](#)  
\l\_zrefclever\_tlistsep\_tl .....  
..... [1537](#), [1570](#), [1931](#)  
\l\_zrefclever\_tpairsep\_tl .....  
..... [1537](#), [1568](#), [1947](#)  
\l\_zrefclever\_type\_<type>\_  
options\_prop ..... [30](#)  
\l\_zrefclever\_type\_count\_int ...  
..... [45](#), [62](#), [1520](#), [1562](#), [1928](#),  
[1930](#), [1939](#), [1966](#), [2323](#), [2335](#), [2429](#)  
\l\_zrefclever\_type\_first\_label\_  
tl [45](#), [60](#), [1522](#), [1558](#), [1723](#), [1841](#),  
[1850](#), [1854](#), [1882](#), [1898](#), [1902](#), [1962](#),  
[1992](#), [2179](#), [2185](#), [2192](#), [2196](#), [2212](#),  
[2253](#), [2271](#), [2275](#), [2281](#), [2297](#), [2311](#)  
\l\_zrefclever\_type\_first\_label\_  
type\_tl [45](#), [62](#), [1522](#), [1559](#), [1725](#),  
[1845](#), [1963](#), [1994](#), [2314](#), [2350](#), [2357](#),  
[2363](#), [2371](#), [2379](#), [2386](#), [2392](#), [2399](#)  
\\_zrefclever\_type\_name\_setup: ..  
..... [9](#), [60](#), [1870](#), [2309](#)  
\l\_zrefclever\_type\_name\_tl .....  
..... [60](#), [62](#),  
[1528](#), [1905](#), [1911](#), [2202](#), [2233](#), [2240](#),  
[2248](#), [2312](#), [2315](#), [2353](#), [2359](#), [2361](#),  
[2374](#), [2382](#), [2388](#), [2394](#), [2396](#), [2413](#)  
\l\_zrefclever\_typeset\_compress\_  
bool ..... [556](#), [559](#), [1977](#)  
\l\_zrefclever\_typeset\_labels\_  
seq [44](#), [1517](#), [1554](#), [1578](#), [1580](#), [1586](#)



\l__zrefclever_typeset_last_bool	last_of_type: .....
..... 45, <a href="#">1517</a> ,	..... 46, 50, 58, 65, 1711, <a href="#">1971</a>
1575, 1576, 1583, 1612, 1936, 2428	\l__zrefclever_typeset_sort_bool
\l__zrefclever_typeset_name_bool	..... 532, 535, 1085
..... 507, 514, 519, 524, 1872, 1886	\l__zrefclever_typesort_seq .....
\l__zrefclever_typeset_queue_-	..... 19, 43, 541, 546, 547, 553, 1460
curr_tl .....	\l__zrefclever_use_hyperref_bool
45,	..... 606, 613,
58, 62, <a href="#">1522</a> , 1557, 1734, 1750,	618, 623, 633, 639, 2137, 2258, 2411
1759, 1790, 1801, 1817, 1839,	\l__zrefclever_warn_hyperref_-
1857, 1874, 1881, 1888, 1921, 1943,	bool .....
1948, 1954, 1960, 1961, 2038, 2049,	607, 614, 619, 624, 637
2080, 2092, 2106, 2328, 2423, 2427	\__zrefclever_zcref:nnn ..
\l__zrefclever_typeset_queue_-	1072, 1073
prev_tl .	\__zrefclever_zcref:nnnn
45, <a href="#">1522</a> , 1556, 1932, 1959	34, 36, <a href="#">1073</a>
\l__zrefclever_typeset_range_-	\l__zrefclever_zcref_labels_seq .
bool .....	..... 36,
565, 568, 1086, 1837	37, 1077, 1104, <a href="#">1108</a> , 1133, 1136, 1555
\l__zrefclever_typeset_ref_bool .	\l__zrefclever_zcref_note_tl ...
..... 506, 513, 518, 523, 1872, 1879	..... 777, 780, 1092, 1096
\__zrefclever_typeset_refs: .....	\l__zrefclever_zcref_with_check_-
..... 44, 46, 47, 1090, <a href="#">1552</a>	bool .....
\__zrefclever_typeset_refs_last_-	784, 799, 1082, 1100
of_type: .	\l__zrefclever_zrefcheck_-
50, 58, 60, 62, 1709, <a href="#">1714</a>	available_bool .....
\__zrefclever_typeset_refs_not_-	..... 783, 794, 805, 1081, 1099