

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-13

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	translations	8
4.3	Options	9
4.3.1	Auxiliary	9
4.3.2	countertype option	9
4.3.3	counterresetters option	10
4.3.4	counterresetby option	11
4.3.5	ref option	11
4.3.6	typeset option	13
4.3.7	sort option	13
4.3.8	typesort option	13
4.3.9	comp option	14
4.3.10	range option	14
4.3.11	hyperref option	14
4.3.12	nameinlink option	15
4.3.13	cap and capfirst options	15
4.3.14	abbrev and noabbrevfirst options	16
4.3.15	lang option	16
4.3.16	font option	19
4.3.17	note option	19
4.3.18	check option	19
4.3.19	Reference options	20
4.4	\zcsetup	21
4.5	Package options	21

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

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

5	Reference format	21
5.1	<code>\zcRefTypeSetup</code>	22
5.2	<code>\zcDeclareTranslations</code>	24
6	User interface	26
6.1	<code>\zcref</code>	26
6.2	<code>\zcpageref</code>	28
7	Sorting	28
8	Typesetting	35
9	Special handling	57
9.1	<code>\appendix</code>	57
9.2	<code>\newtheorem</code>	57
9.3	<code>enumitem</code> package	57
10	Translations	57
10.1	Fallback	57
10.2	English	58
10.3	German	61
10.4	French	64
10.5	Portuguese	66
10.6	Spanish	69
	Index	72

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 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 translations (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 (`ltxcmds`), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

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

```

11     }%
12     \endinput
13 }%
    Identify the package.
14 \ProvidesExplPackage {zref-clever} {2021-09-13} {0.1.0-alpha}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

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

```

16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \RequirePackage { zref-counter }
19 \RequirePackage { zref-abspage }
20 \RequirePackage { translations }

```

3 zref setup

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

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

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

```

21 \zref@newprop { zc@thecnt } { \use:c { the \@currentcounter } }
22 \zref@addprop \ZREF@mainlist { zc@thecnt }

```

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

```

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

```

```

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

```

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

```

33 \zref@newprop { zc@cntval } [0] { \int_use:c { c@ \@currentcounter } }
34 \zref@addprop \ZREF@mainlist { zc@cntval }
35 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
36 \zref@addprop \ZREF@mainlist { zc@pgval }

```

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

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

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

The procedure below examines a set of counters, those included in `\l__zrefclever_counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\cl@⟨counter⟩`, looking for the counter for which we are trying to set a label (`\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it

is easy to add more counters to this list if needed, with the option `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\c1@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resetters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`__zrefclever_get_enclosing_counters:n`
`__zrefclever_get_enclosing_counters_value:n`

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

```

    \__zrefclever_get_enclosing_counters:n {<counter>}
    \__zrefclever_get_enclosing_counters_value:n {<counter>}

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

```

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

```

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

```

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

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

```

\__zrefclever_counter_reset_by:n {<counter>}

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

```

(End definition for `_zrefclever_counter_reset_by:n`.)

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

```

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

```

Another piece of information we need is the page numbering format being used by `\thepage`, so that we know when we can (or not) group a set of page references in a range. Unfortunately, `page` is not a typical counter in ways which complicates things. First, it does commonly get reset along the document, not necessarily by the usual counter reset chains, but rather with `\pagenumbering` or variations thereof. Second, the format of the page number commonly changes in the document (roman, arabic, etc.), not necessarily, though usually, together with a reset. Trying to “parse” `\thepage` to retrieve such information is bound to go wrong: we don’t know, and can’t know, what is within that macro, and that’s the business of the user, or of the documentclass, or of the loaded packages. The technique used by `cleveref`, which we borrow here, is simple

and smart: store with the label what `\thepage` would return, if the counter `\c@page` was “1”. That does not allow us to *sort* the references, luckily however, we have `abspage` which solves this problem. But we can decide whether two labels can be compressed into a range or not based on this format: if they are identical, we can compress them, otherwise, we can’t. To do so, we locally redefine `\c@page` to return “1”, thus avoiding any global spillovers of this trick. Since this operation is not expandable we cannot run it directly from the property definition. Hence, we use a shipout hook, and set `\g__zrefclever_page_format_tl`, which can then be retrieved by the starred definition of `\zref@newprop*{zc@pgfmt}`.

```

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

```

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

4 Plumbing

4.1 Messages

```

100 \msg_new:nnn { zref-clever } { option-not-type-specific }
101 {
102   Option~'#1'~is-not-type-specific~\msg_line_context:~
103   Set-it-in~'\exp_not:N \zcDeclareTranslations'~before-first~'type'~switch-
104   or-as~package~option.
105 }
106 \msg_new:nnn { zref-clever } { option-only-type-specific }
107 {
108   No~type~specified~for~option~'#1'~\msg_line_context:~
109   Set-it-after~'type'~switch-or-in~'\exp_not:N \zcRefTypeSetup'.
110 }
111 \msg_new:nnn { zref-clever } { key-requires-value }
112 { The~'#1'~key~'#2'~requires~a~value. }
113 \msg_new:nnn { zref-clever } { missing-zref-titleref }
114 {
115   Option~'ref=title'~requested~\msg_line_context:~
116   But~package~'zref-titleref'~is~not~loaded,~falling-back~to~default~'ref'.
117 }
118 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
119 {
120   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
121   Use~the~starred~version~of~'\noexpand\zcheck'~instead.

```

```

122 }
123 \msg_new:nnn { zref-clever } { missing-hyperref }
124 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
125 \msg_new:nnn { zref-check } { check-document-only }
126 { Option~'check'~only~available~in~the~document. }
127 \msg_new:nnn { zref-clever } { missing-zref-check }
128 {
129   Option~'check'~requested~\msg_line_context:~
130   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
131 }
132 \msg_new:nnn { zref-clever } { counters-not-nested }
133 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:. }
134 \msg_new:nnn { zref-clever } { missing-type }
135 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
136 \msg_new:nnn { zref-clever } { missing-name }
137 { Name~undefined~for~type~'#1'~\msg_line_context:. }
138 \msg_new:nnn { zref-clever } { single-element-range }
139 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }

```

4.2 translations

Some wrappers around translations functions, so that we can generate variants with expansion control for arguments, or for convenience.

`__zrefclever_if_transl:nnTF` Conditional to check if a translation of $\langle key \rangle$ exists for language $\langle lang \rangle$.

```

\__zrefclever_if_transl:nnTF {<lang>} {<key>} {<true>} {<false>}
140 \prg_new_conditional:Npnn \__zrefclever_if_transl:nn #1#2 { TF }
141 {
142   \IfTranslation {#1} {#2}
143   { \prg_return_true: }
144   { \prg_return_false: }
145 }
146 \prg_generate_conditional_variant:Nnn \__zrefclever_if_transl:nn { xx } { TF }

```

(End definition for `__zrefclever_if_transl:nnTF`.)

`__zrefclever_get_transl:nnn` Retrieves the translation of $\langle key \rangle$ for the language $\langle lang \rangle$ and saves it in $\langle macro \rangle$.

```

\__zrefclever_get_transl:nnn {<macro>} {<lang>} {<key>}
147 \cs_new_protected:Npn \__zrefclever_get_transl:nnn #1#2#3
148 { \SaveTranslationFor{#1}{#2}{#3} }
149 \cs_generate_variant:Nn \__zrefclever_get_transl:nnn { nxx }

```

(End definition for `__zrefclever_get_transl:nnn`.)

`_zrefclever_declare_transl:nnn` Defines the translation of $\langle key \rangle$ for the language $\langle lang \rangle$. The $\langle key \rangle$ here is the full key, including package prefix, type, and internal key name (i.e. the “key” from the perspective of translations).

```

\__zrefclever_declare_transl:nnn {<lang>} {<key>} {<translation>}
150 \cs_new_protected:Npn \_zrefclever_declare_transl:nnn #1#2#3
151 { \declaretranslation {#1} {#2} {#3} }
152 \cs_generate_variant:Nn \_zrefclever_declare_transl:nnn { xxn }

```


(End definition for `_zrefclever_declare_transl:nnn`.)

`_zrefclever_declare_fallback_transl:nn` Defines the default fallback translation of $\langle key \rangle$ for the language $\langle lang \rangle$. The $\langle key \rangle$ here is the internal key name (i.e. the name of the option).

```

\__zrefclever_declare_fallback_transl:nn {\langle key \rangle} {\langle translation \rangle}

153 \cs_new_protected:Npn \__zrefclever_declare_fallback_transl:nn #1#2
154 { \declaretranslationfallback { zrefclever-default- #1 } {#2} }

```

(End definition for `_zrefclever_declare_fallback_transl:nn`.)

`\zcDicDefaultTransl` `\zcDicTypeTransl` Functions for providing translations in dictionary files. We refrain from using `expl3` names and “atletter”, so that we don’t have to control catcodes in those files (as far as I can tell, `translations` itself doesn’t cater for this), even if these commands are only really meant for internal use. The $\langle key \rangle$ here is always the internal key name (i.e. the name of the option). The language does not need to be specified, it is automatically retrieved from the dictionary’s declaration done by `\ProvideDictionaryFor`. Since `\ProvideDictTranslation` is restricted by translations to the preamble, we inherit this restriction here.

```

\zcDicDefaultTransl {\langle key \rangle} {\langle translation \rangle}
\zcDicTypeTransl {\langle type \rangle} {\langle key \rangle} {\langle translation \rangle}

155 \NewDocumentCommand \zcDicDefaultTransl { m m }
156 { \ProvideDictTranslation { zrefclever-default- #1 } {#2} }
157 \NewDocumentCommand \zcDicTypeTransl { m m m }
158 { \ProvideDictTranslation { zrefclever-type- #1 - #2 } {#3} }
159 \@onlypreamble \zcDicDefaultTransl
160 \@onlypreamble \zcDicTypeTransl

```

(End definition for `\zcDicDefaultTransl` and `\zcDicTypeTransl`.)

4.3 Options

4.3.1 Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn \langle property list \rangle {\langle key \rangle} {\langle value \rangle}

161 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
162 {
163   \tl_if_empty:nTF {#3}
164     { \prop_remove:Nn #1 {#2} }
165     { \prop_put:Nnn #1 {#2} {#3} }
166 }

```

(End definition for `_zrefclever_prop_put_non_empty:Nnn`.)

4.3.2 countertype option

`\l__zrefclever_counter_type_prop` is used by `zc@type` property, and stores a mapping from “counter” to “reference type”. Only those counters whose type name is different from that of the counter need to be specified, since `zc@type` presumes the counter as the type if the counter is not found in `\l__zrefclever_counter_type_prop`.

```

167 \prop_new:N \l__zrefclever_counter_type_prop
168 \keys_define:nn { zref-clever / label }
169 {
170   countertype .code:n =
171   {
172     \keyval_parse:nnn
173     {
174       \msg_warning:nnnn { zref-clever }
175       { key-requires-value } { countertype }
176     }
177     {
178       \__zrefclever_prop_put_non_empty:Nnn
179       \l__zrefclever_counter_type_prop
180     }
181     {#1}
182   } ,
183   countertype .value_required:n = true ,
184   countertype .initial:n =
185   {
186     subsection      = section ,
187     subsubsection    = section ,
188     subparagraph     = paragraph ,
189     enumi             = item ,
190     enumii            = item ,
191     enumiii           = item ,
192     enumiv            = item ,
193   } ,
194 }
```

4.3.3 counterresetters option

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

```

195 \seq_new:N \l__zrefclever_counter_resetters_seq
196 \keys_define:nn { zref-clever / label }
197 {
198   counterresetters .code:n =
199   {
200     \clist_map_inline:nn {#1}
201     {
202       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
```

```

203         {
204             \seq_put_right:Nn
205             \l__zrefclever_counter_resettters_seq {##1}
206         }
207     }
208 },
209 counterresettters .initial:n =
210 {
211     part ,
212     chapter ,
213     section ,
214     subsection ,
215     subsubsection ,
216     paragraph ,
217     subparagraph ,
218 },
219 typesort .value_required:n = true ,
220 }

```

4.3.4 counterresetby option

`\l__zrefclever_counter_resetby_prop` is used by `__zrefclever_counter_resetby: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_resetby:n` over the search through `\l__zrefclever_counter_resettters_seq`.

```

221 \prop_new:N \l__zrefclever_counter_resetby_prop
222 \keys_define:nn { zref-clever / label }
223 {
224     counterresetby .code:n =
225     {
226         \keyval_parse:nnn
227         {
228             \msg_warning:nnn { zref-clever }
229             { key-requires-value } { counterresetby }
230         }
231         {
232             \__zrefclever_prop_put_non_empty:Nnn
233             \l__zrefclever_counter_resetby_prop
234         }
235         {##1}
236     } ,
237     counterresetby .value_required:n = true ,
238     counterresetby .initial:n =
239     {

```

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.

```

240         enumii = enumi ,
241         enumiii = enumii ,
242         enumiv = enumiii ,
243     } ,
244 }

```

4.3.5 ref option

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

```

245 \tl_new:N \l__zrefclever_ref_property_tl
246 \keys_define:nn { zref-clever / reference }
247 {
248   ref .choice: ,
249   ref / zc@thecnt .code:n =
250     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
251   ref / page .code:n =
252     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
253   ref / title .code:n =
254     {
255       \AddToHook { begindocument }
256       {
257         \@ifpackageloaded { zref-titleref }
258         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
259         {
260           \msg_warning:nn { zref-clever } { missing-zref-titleref }
261           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
262         }
263       }
264     } ,
265   ref .initial:n = zc@thecnt ,
266   ref .value_required:n = true ,
267   page .meta:n = { ref = page },
268   page .value_forbidden:n = true ,
269 }
270 \AddToHook { begindocument }
271 {
272   \@ifpackageloaded { zref-titleref }
273   {
274     \keys_define:nn { zref-clever / reference }
275     {
276       ref / title .code:n =
277         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
278     }
279   }
280   {
281     \keys_define:nn { zref-clever / reference }
282     {
283       ref / title .code:n =
284       {

```

```

285         \msg_warning:nn { zref-clever } { missing-zref-titleref }
286         \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
287     }
288 }
289 }
290 }

```

4.3.6 typeset option

```

291 \bool_new:N \l__zrefclever_typeset_ref_bool
292 \bool_new:N \l__zrefclever_typeset_name_bool
293 \keys_define:nn { zref-clever / reference }
294 {
295     typeset .choice: ,
296     typeset / both .code:n =
297     {
298         \bool_set_true:N \l__zrefclever_typeset_ref_bool
299         \bool_set_true:N \l__zrefclever_typeset_name_bool
300     } ,
301     typeset / ref .code:n =
302     {
303         \bool_set_true:N \l__zrefclever_typeset_ref_bool
304         \bool_set_false:N \l__zrefclever_typeset_name_bool
305     } ,
306     typeset / name .code:n =
307     {
308         \bool_set_false:N \l__zrefclever_typeset_ref_bool
309         \bool_set_true:N \l__zrefclever_typeset_name_bool
310     } ,
311     typeset .initial:n = both ,
312     typeset .value_required:n = true ,
313
314     noname .meta:n = { typeset = ref },
315     noname .value_forbidden:n = true ,
316 }

```

4.3.7 sort option

```

317 \bool_new:N \l__zrefclever_typeset_sort_bool
318 \keys_define:nn { zref-clever / reference }
319 {
320     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
321     sort .initial:n = true ,
322     sort .default:n = true ,
323     nosort .meta:n = { sort = false },
324     nosort .value_forbidden:n = true ,
325 }

```

4.3.8 typesort option

`\l__zrefclever_typesort_seq` is stored reversed, since the sort priorities are computed in the negative range in `__zrefclever_sort_default_different_types:nn`, so that we can implicitly rely on ‘0’ being the “last value”, and spare creating an integer variable using `\seq_map_indexed_inline:Nn`.

```

326 \seq_new:N \l__zrefclever_typesort_seq

```

```

327 \keys_define:nn { zref-clever / reference }
328 {
329   typesort .code:n =
330   {
331     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
332     \seq_reverse:N \l__zrefclever_typesort_seq
333   } ,
334   typesort .initial:n =
335   { part , chapter , section , paragraph } ,
336   typesort .value_required:n = true ,
337   notypesort .code:n =
338   { \seq_clear:N \l__zrefclever_typesort_seq } ,
339   notypesort .value_forbidden:n = true ,
340 }

```

4.3.9 comp option

```

341 \bool_new:N \l__zrefclever_typeset_compress_bool
342 \keys_define:nn { zref-clever / reference }
343 {
344   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
345   comp .initial:n = true ,
346   comp .default:n = true ,
347   nocomp .meta:n = { comp = false } ,
348   nocomp .value_forbidden:n = true ,
349 }

```

4.3.10 range option

```

350 \bool_new:N \l__zrefclever_typeset_range_bool
351 \keys_define:nn { zref-clever / reference }
352 {
353   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
354   range .initial:n = false ,
355   range .default:n = true ,
356 }

```

4.3.11 hyperref option

```

357 \bool_new:N \l__zrefclever_use_hyperref_bool
358 \bool_new:N \l__zrefclever_warn_hyperref_bool
359 \keys_define:nn { zref-clever / reference }
360 {
361   hyperref .choice: ,
362   hyperref / auto .code:n =
363   {
364     \bool_set_true:N \l__zrefclever_use_hyperref_bool
365     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
366   } ,
367   hyperref / true .code:n =
368   {
369     \bool_set_true:N \l__zrefclever_use_hyperref_bool
370     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
371   } ,
372   hyperref / false .code:n =
373   {

```

```

374         \bool_set_false:N \l__zrefclever_use_hyperref_bool
375         \bool_set_false:N \l__zrefclever_warn_hyperref_bool
376     } ,
377     hyperref .initial:n = auto ,
378     hyperref .default:n = auto
379 }
380 \AddToHook { begindocument }
381 {
382     \@ifpackageloaded { hyperref }
383     {
384         \bool_if:NT \l__zrefclever_use_hyperref_bool
385         { \RequirePackage { zref-hyperref } }
386     }
387     {
388         \bool_if:NT \l__zrefclever_warn_hyperref_bool
389         { \msg_warning:nn { zref-clever } { missing-hyperref } }
390         \bool_set_false:N \l__zrefclever_use_hyperref_bool
391     }
392     \keys_define:nn { zref-clever / reference }
393     {
394         hyperref .code:n =
395         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
396     }
397 }

```

4.3.12 nameinlink option

```

398 \str_new:N \l__zrefclever_nameinlink_str
399 \keys_define:nn { zref-clever / reference }
400 {
401     nameinlink .choice: ,
402     nameinlink / true .code:n =
403     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
404     nameinlink / false .code:n =
405     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
406     nameinlink / single .code:n =
407     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
408     nameinlink / tsingle .code:n =
409     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
410     nameinlink .initial:n = tsingle ,
411     nameinlink .default:n = true ,
412 }

```

4.3.13 cap and capfirst options

```

413 \bool_new:N \l__zrefclever_capitalize_bool
414 \bool_new:N \l__zrefclever_capitalize_first_bool
415 \keys_define:nn { zref-clever / reference }
416 {
417     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
418     cap .initial:n = false ,
419     cap .default:n = true ,
420     nocap .meta:n = { cap = false } ,
421     nocap .value_forbidden:n = true ,
422
423     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,

```

```

424     capfirst .initial:n = false ,
425     capfirst .default:n = true ,
426
427     C .meta:n =
428       { capfirst = true , noabbrevfirst = true },
429     C .value_forbidden:n = true ,
430   }

```

4.3.14 abbrev and noabbrevfirst options

```

431 \bool_new:N \l__zrefclever_abbrev_bool
432 \bool_new:N \l__zrefclever_noabbrev_first_bool
433 \keys_define:nn { zref-clever / reference }
434 {
435   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
436   abbrev .initial:n = false ,
437   abbrev .default:n = true ,
438   noabbrev .meta:n = { abbrev = false },
439   noabbrev .value_forbidden:n = true ,
440
441   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
442   noabbrevfirst .initial:n = false ,
443   noabbrevfirst .default:n = true ,
444 }

```

4.3.15 lang option

`\l__zrefclever_current_language_tl` is an internal alias for translations’s internal macro `\@trnslt@current@language` which, in turn, is an alias for `\language` used by both `babel` and `polyglossia`, but translations ensures it always exists, even if no language package is loaded. `\l__zrefclever_main_language_tl` is an internal alias for `babel`’s `\bbl@main@language` or for `polyglossia`’s `\xpg@main@language`, as the case may be. `\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/before` hook. And it must be `before`, since `\LoadDictionaryFor` is preamble only. The `begindocument/before` hook is responsible to get values for `\l__zrefclever_main_language_tl` and `\l__zrefclever_current_language_tl` and load `zref-clever` dictionaries for all languages loaded by `babel` or `polyglossia`, or directly specified by the user. After this information is retrieved, the preamble options are executed, and this is handled by the internal `zref-clever/reflanguage` hook, which is called at this point. This hook handles two things: it executes the preamble options and, in sequence, it redefines the `lang` option key, since in the document body, we can handle “main” and “current” language options immediately. This redefinition is added to the `zref-clever/reflanguage` hook, but `\AtEndOfPackage` so that it comes after `\ProcessKeysOptions`. In other words, this is how we ensure the preamble options are executed before the `lang` key is redefined.

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.


```

445 \tl_new:N \l__zrefclever_ref_language_tl
446 \tl_new:N \l__zrefclever_main_language_tl
447 \tl_new:N \l__zrefclever_current_language_tl
448 \NewHook { zref-clever / reflanguage }
449 \keys_define:nn { zref-clever / reference }
450 {
451   lang .code:n =
452   {
453     \AddToHook { zref-clever / reflanguage }
454     {
455       \str_case:nnF {#1}
456       {
457         { main }
458         {
459           \tl_set_eq:NN \l__zrefclever_ref_language_tl
460           \l__zrefclever_main_language_tl
461         }
462
463         { current }
464         {
465           \tl_set_eq:NN \l__zrefclever_ref_language_tl
466           \l__zrefclever_current_language_tl
467         }
468       }
469     }
470     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}

```

If the user specified a language in the preamble, make sure it is loaded. There's no need to worry with redundancy with babel and polyglossia loaded languages, since `\LoadDictionaryFor` does not reload a dictionary if it's already been loaded.

```

471     \exp_args:Nx \file_if_exist:nTF
472     { zref-clever- \@trnslt@language {#1} .trsl }
473     { \LoadDictionaryFor {#1} { zref-clever } }
474     {
475       \exp_args:Nx \file_if_exist:nTF
476       { zref-clever- \baselanguage {#1} .trsl }
477       { \LoadDictionaryFor {#1} { zref-clever } }
478     }
479   }
480 } ,
481 lang .initial:n = main ,
482 lang .value_required:n = true ,
483 }

```

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

```

485 \AtEndOfPackage
486 {
487   \AddToHook { zref-clever / reflanguage }
488   {
489     \keys_define:nn { zref-clever / reference }
490     {
491       lang .code:n =
492       {
493         \str_case:nnF {#1}

```

```

494         {
495             { main }
496             {
497                 \tl_set_eq:NN \l__zrefclever_ref_language_tl
498                 \l__zrefclever_main_language_tl
499             }
500
501             { current }
502             {
503                 \tl_set_eq:NN \l__zrefclever_ref_language_tl
504                 \l__zrefclever_current_language_tl
505             }
506         }
507         { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
508     } ,
509     lang .value_required:n = true ,
510 }
511 }
512 }
513 \AddToHook { begindocument / before }
514 {
515     \tl_set_eq:NN \l__zrefclever_current_language_tl
516     \@trnslt@current@language
517     \@ifpackageloaded{babel}
518     {
519         \tl_set_eq:NN \l__zrefclever_main_language_tl
520         \bbl@main@language
521         \clist_map_inline:Nn \bbl@loaded
522         {

```

Funny enough, translations also loads its basic dictionaries for all languages loaded by babel or polyglossia. First, there is no way to disable this, even if we don't need them at all here. Second, translations sends messages of its own missing dictionaries to **info** and everyone else's to **warning**... So we have to control ourselves for missing dictionaries and load them only if available.

```

523         \exp_args:Nx \file_if_exist:nTF
524         { zref-clever- \@trnslt@language {#1} .trsl }
525         { \LoadDictionaryFor {#1} { zref-clever } }
526         {
527             \exp_args:Nx \file_if_exist:nT
528             { zref-clever- \baselanguage {#1} .trsl }
529             { \LoadDictionaryFor {#1} { zref-clever } }
530         }
531     }
532 }
533 {
534     \@ifpackageloaded{polyglossia}
535     {
536         \tl_set_eq:NN \l__zrefclever_main_language_tl
537         \xpg@main@language
538         \clist_map_inline:Nn \xpg@loaded
539         {
540             \exp_args:Nx \file_if_exist:nTF
541             { zref-clever- \@trnslt@language {#1} .trsl }

```

```

542         { \LoadDictionaryFor {#1} { zref-clever } }
543         {
544             \exp_args:Nx \file_if_exist:nT
545             { zref-clever- \baselanguage {#1} .trsl }
546             { \LoadDictionaryFor {#1} { zref-clever } }
547         }
548     }
549 }
550 {
551     \tl_set:Nn \l__zrefclever_main_language_tl { english }
552     \LoadDictionaryFor { english } { zref-clever }
553 }
554 }

```

Then we execute the package options stored in the zref-clever/reflanguage hook.

```

555     \UseHook { zref-clever / refluanguage }
556 }

```

4.3.16 font option

```

557 \tl_new:N \l__zrefclever_ref_typeset_font_tl
558 \keys_define:nn { zref-clever / reference }
559 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

4.3.17 note option

```

560 \tl_new:N \l__zrefclever_zcref_note_tl
561 \keys_define:nn { zref-clever / reference }
562 {
563     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
564     note .value_required:n = true ,
565 }

```

4.3.18 check option

Integration with zref-check.

```

566 \bool_new:N \l__zrefclever_zrefcheck_available_bool
567 \bool_new:N \l__zrefclever_zcref_with_check_bool
568 \keys_define:nn { zref-clever / reference }
569 {
570     check .code:n =
571     { \msg_warning:nn { zref-clever } { check-document-only } } ,
572 }
573 \AddToHook { begindocument }
574 {
575     \@ifpackageloaded { zref-check }
576     {
577         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
578         \keys_define:nn { zref-clever / reference }
579         {
580             check .code:n =
581             {
582                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
583                 \keys_set:nn { zref-check / zcheck } {#1}
584             }
585         }
586     }
587 }

```

```

586     }
587     {
588       \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
589       \keys_define:nn { zref-clever / reference }
590       {
591         check .code:n =
592         { \msg_warning:nn { zref-clever } { missing-zref-check } }
593       }
594     }
595   }

```

4.3.19 Reference options

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

The keys are set so that any value, including an empty one, is added to `\l__zrefclever_ref_options_prop`, while a key with *no value* removes the property from the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.1.

```

596 \prop_new:N \l__zrefclever_ref_options_prop
597 \clist_map_inline:nn
598 {

```

Not type-specific options.

```

599   tpairsep ,
600   tlistsep ,
601   tlastsep ,
602   notesep ,

```

Possibly type-specific options.

```

603   namefont ,
604   namesep ,
605   pairsep ,
606   listsep ,
607   lastsep ,
608   rangesep ,
609   reffont ,
610   refpre ,
611   refpos ,
612   reffont-in ,
613   refpre-in ,
614   refpos-in ,
615 }
616 {
617   \keys_define:nn { zref-clever / reference }
618   {
619     #1 .default:V = \c_novalue_tl ,
620     #1 .code:n =

```

```

621         {
622             \tl_if_novalue:nTF {##1}
623             { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
624             { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
625         } ,
626     }
627 }

```

4.4 \zcsetup

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

```

628 \keys_define:nn { }
629 {
630     zref-clever / zcsetup .inherit:n = zref-clever / label ,
631     zref-clever / zcsetup .inherit:n = zref-clever / reference ,
632 }

```

`\zcsetup` Provide `\zcsetup`.

```

633 \NewDocumentCommand \zcsetup { m }
634 { \keys_set:nn { zref-clever / zcsetup } {#1} }

```

(End definition for `\zcsetup`.)

4.5 Package options

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

```

635 \RequirePackage { l3keys2e }
636 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Reference format

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

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

enforced in `__zrefclever_get_option_with_transl:nN` and `__zrefclever_get_option_plain:nN`, which are the basic functions to retrieve proper values for reference format settings.

General “options” (i) can be given by the user in the optional argument of `\zcRef`, but just as well in `\zcsetup` or as package options at load-time (see Section 4.3.19). “Type-specific options” (ii) are handled by `\zcRefTypeSetup`. “Language-specific translations”, be they “type-specific” (iii) or “default” (iv) have their user interface in `\zcDeclareTranslations`, and have their values populated by the package’s dictionaries.

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

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

```
\l__zrefclever_setup_type_tl Store type and language in use in \zcRefTypeSetup and \zcDeclareTranslations.
\l__zrefclever_setup_language_tl
637 \tl_new:N \l__zrefclever_setup_type_tl
638 \tl_new:N \l__zrefclever_setup_language_tl

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

5.1 \zcRefTypeSetup

`\zcRefTypeSetup` is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any translation, hence they override any language-specific setting, either done at `\zcDeclareTranslations` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The `<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>}
639 \NewDocumentCommand \zcRefTypeSetup { m m }
640 {
641   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
642   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
643   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
644   \keys_set:nn { zref-clever / typesetup } {#2}
645 }
```

(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 4.3.19), we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit at `\keys_set:nn` by means of the `.default:V` property of the key in `\keys_define:nn`. For the technique and some discussion about it, see <https://tex.stackexchange.com/q/614690> (thanks Jonathan P. Spratte, aka ‘Skillmon’, and Phelype Oleinik) and <https://github.com/latex3/latex3/pull/988>.

```
646 \clist_map_inline:nn
647 {
```

Not type-specific options.

```
648   tpairsep ,
649   tlistsep ,
650   tlastsep ,
651   notesep ,
652 }
653 {
654   \keys_define:nn { zref-clever / typesetup }
655   {
656     #1 .code:n =
657     {
658       \msg_warning:nnn { zref-clever }
659         { option-not-type-specific } {#1}
660     } ,
661   }
662 }
663 \clist_map_inline:nn
664 {
```

Possibly type-specific options.

```
665   namefont ,
666   namesep ,
667   pairsep ,
668   listsep ,
669   lastsep ,
670   rangesep ,
671   reffont ,
672   refpre ,
673   refpos ,
674   reffont-in ,
675   refpre-in ,
676   refpos-in ,
```

Necessarily type-specific options.

```
677   Name-sg ,
678   name-sg ,
679   Name-pl ,
680   name-pl ,
681   Name-sg-ab ,
682   name-sg-ab ,
683   Name-pl-ab ,
684   name-pl-ab ,
685 }
```

```

686 {
687   \keys_define:nn { zref-clever / typesetup }
688   {
689     #1 .default:V = \c_novalue_tl ,
690     #1 .code:n =
691     {
692       \tl_if_novalue:nTF {##1}
693       {
694         \prop_remove:cn
695         {
696           l__zrefclever_type_
697           \l__zrefclever_setup_type_tl _options_prop
698         }
699         {#1}
700       }
701       {
702         \prop_put:cnn
703         {
704           l__zrefclever_type_
705           \l__zrefclever_setup_type_tl _options_prop
706         }
707         {#1} {##1}
708       }
709     } ,
710   }
711 }

```

5.2 \zcDeclareTranslations

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

```

\zcDeclareTranslations      \zcDeclareTranslations { $\langle language \rangle$ } { $\langle options \rangle$ }
712 \NewDocumentCommand \zcDeclareTranslations { m m }
713 {
714   \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
715   \tl_clear:N \l__zrefclever_setup_type_tl
716   \keys_set:nn { zref-clever / translations } {#2}
717 }

(End definition for \zcDeclareTranslations.)

718 \keys_define:nn { zref-clever / translations }
719 {
720   type .code:n =
721   {
722     \tl_if_empty:nTF {#1}
723     { \tl_clear:N \l__zrefclever_setup_type_tl }
724     {

```



```

725         \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
726         { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
727         \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
728     }
729 },
730 }
731 \clist_map_inline:nn
732 {

```

Not type-specific options.

```

733     tpairsep ,
734     tlistsep ,
735     tlastsep ,
736     notesep ,
737 }
738 {
739     \keys_define:nn { zref-clever / translations }
740     {
741         #1 .value_required:n = true ,
742         #1 .code:n =
743         {
744             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
745             {
746                 \__zrefclever_declare_transl:xxn
747                 { \l__zrefclever_setup_language_tl }
748                 { zrefclever-default- #1 } {##1}
749             }
750             {
751                 \msg_warning:nnn { zref-clever }
752                 { option-not-type-specific } {#1}
753             }
754         } ,
755     }
756 }
757 \clist_map_inline:nn
758 {

```

Possibly type-specific options.

```

759     namesep ,
760     pairsep ,
761     listsep ,
762     lastsep ,
763     rangesep ,
764     refpre ,
765     refpos ,
766     refpre-in ,
767     refpos-in ,
768 }
769 {
770     \keys_define:nn { zref-clever / translations }
771     {
772         #1 .value_required:n = true ,
773         #1 .code:n =
774         {

```

```

775         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
776         {
777             \__zrefclever_declare_transl:xxn
778             { \l__zrefclever_setup_language_tl }
779             { zrefclever-default- #1 } {##1}
780         }
781         {
782             \__zrefclever_declare_transl:xxn
783             { \l__zrefclever_setup_language_tl }
784             { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
785         }
786     } ,
787 }
788 }
789 \clist_map_inline:nn
790 {

```

Necessarily type-specific options.

```

791     Name-sg ,
792     name-sg ,
793     Name-pl ,
794     name-pl ,
795     Name-sg-ab ,
796     name-sg-ab ,
797     Name-pl-ab ,
798     name-pl-ab ,
799 }
800 {
801     \keys_define:nn { zref-clever / translations }
802     {
803         #1 .value_required:n = true ,
804         #1 .code:n =
805         {
806             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
807             {
808                 \msg_warning:nnn { zref-clever }
809                 { option-only-type-specific } {#1}
810             }
811             {
812                 \__zrefclever_declare_transl:xxn
813                 { \l__zrefclever_setup_language_tl }
814                 { zrefclever-type- \l__zrefclever_setup_type_tl - #1 } {##1}
815             }
816         } ,
817     }
818 }

```

6 User interface

6.1 \zcref

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

```

819 \NewDocumentCommand \zcref { s O { } m }
820 { \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 $\{\langle labels \rangle\}$ as first argument, so that it can be protected by \zref@wrapper@babel in \zcref.

```

      \_zrefclever_zcref:nnnn {\langle labels \rangle} {\langle * \rangle} {\langle options \rangle}
821 \cs_new_protected:Npn \_zrefclever_zcref:nnn #1#2#3
822 {
823   \group_begin:

```

Set options.

```

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

```

Store arguments values.

```

825     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
826     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

Integration with zref-check.

```

827     \bool_lazy_and:nnT
828     { \l__zrefclever_zrefcheck_available_bool }
829     { \l__zrefclever_zcref_with_check_bool }
830     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

831     \bool_lazy_or:nnT
832     { \l__zrefclever_typeset_sort_bool }
833     { \l__zrefclever_typeset_range_bool }
834     { \_zrefclever_sort_labels: }

```

Typeset the references.

```

835     \_zrefclever_typeset_refs:

```

Typeset note.

```

836     \l__zrefclever_notesep_tl
837     \l__zrefclever_zcref_note_tl

```

Integration with zref-check.

```

838     \bool_lazy_and:nnT
839     { \l__zrefclever_zrefcheck_available_bool }
840     { \l__zrefclever_zcref_with_check_bool }
841     {
842       \zrefcheck_zcref_end_label_maybe:
843       \zrefcheck_zcref_run_checks_on_labels:n
844       { \l__zrefclever_zcref_labels_seq }
845     }
846     \group_end:
847 }

```

(End definition for _zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

848 \seq_new:N \l__zrefclever_zcref_labels_seq
849 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for \l__zrefclever_zcref_labels_seq and \l__zrefclever_link_star_bool.)

6.2 \zcpageref

```
\zcpageref          \zcpageref{<*>[<options>]{<labels>}}
850 \NewDocumentCommand \zcpageref { s O { } m }
851 {
852   \IfBooleanTF {#1}
853     { \zcref*[#2, ref = page] {#3} }
854     { \zcref [ #2, ref = page] {#3} }
855 }
```

(End definition for \zcpageref.)

7 Sorting

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

```
\l__zrefclever_label_a_tl 856 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl 857 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_label_type_a_tl 858 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl 859 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclcnt_a_tl 860 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclcnt_b_tl 861 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
\l__zrefclever_label_enclval_a_tl 862 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl 863 \tl_new:N \l__zrefclever_label_enclval_b_tl
```

(End definition for \l__zrefclever_label_a_tl and others.)

```
864 \int_new:N \l__zrefclever_sort_prior_a_int
865 \int_new:N \l__zrefclever_sort_prior_b_int
```

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

```
866 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for \l__zrefclever_sort_decided_bool.)

Variant not provided by the kernel.

```
867 \cs_generate_variant:Nn \tl_reverse_items:n { V }
```

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

```
\__zrefclever_label_type_put_new_right:n {<label>}
868 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
869 {
870   \tl_set:Nx \l__zrefclever_label_type_a_tl
871     { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
872   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
873   {
874     \seq_if_in:NVF
875       \l__zrefclever_label_types_seq
876       \l__zrefclever_label_type_a_tl
877     {
```

```

878         \seq_put_right:NV \l__zrefclever_label_types_seq
879         \l__zrefclever_label_type_a_tl
880     }
881 }
882 }

```

(End definition for __zrefclever_label_type_put_new_right:n.)

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

```

883 \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.

```

884 \cs_new_protected:Npn \__zrefclever_sort_labels:
885 {

```

Store label types sequence.

```

886     \seq_clear:N \l__zrefclever_label_types_seq
887     \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
888     {
889         \seq_map_function:NN \l__zrefclever_zcref_labels_seq
890         \__zrefclever_label_type_put_new_right:n
891     }

```

Sort.

```

892     \seq_sort:Nn \l__zrefclever_zcref_labels_seq
893     {
894         \zref@ifrefundefined {##1}
895         {
896             \zref@ifrefundefined {##2}
897             {
898                 % Neither label is defined.
899                 \sort_return_same:
900             }
901             {
902                 % The second label is defined, but the first isn't, leave the
903                 % undefined first (to be more visible).
904                 \sort_return_same:
905             }
906         }
907         {
908             \zref@ifrefundefined {##2}
909             {
910                 % The first label is defined, but the second isn't, bring the
911                 % second forward.
912                 \sort_return_swapped:
913             }
914             {

```

```

915         % The interesting case: both labels are defined. The
916         % reference to the "default" property/counter or to the page
917         % are quite different from our perspective, they rely on
918         % different fields and even use different information for
919         % sorting, so we branch them here to specialized functions.
920         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
921         { \__zrefclever_sort_page:nn {##1} {##2} }
922         { \__zrefclever_sort_default:nn {##1} {##2} }
923     }
924 }
925 }
926 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_sort_default:nn The heavy-lifting function for sorting of existing labels for “default” references (that is, a standard reference, not to “page”). This function is expected to be called within the sorting loop of __zrefclever_sort_labels: and receives the pair of labels being considered for a change of order or not. It should *always* “return” either \sort_return_same: or \sort_return_swapped:.

```

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

927 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
928 {
929     \tl_set:Nx \l__zrefclever_label_type_a_tl
930     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
931     \tl_set:Nx \l__zrefclever_label_type_b_tl
932     { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
933
934     \bool_if:nTF
935     {
936         % The second label has a type, but the first doesn't, leave the
937         % undefined first (to be more visible).
938         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
939         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
940     }
941     { \sort_return_same: }
942     {
943         \bool_if:nTF
944         {
945             % The first label has a type, but the second doesn't, bring the
946             % second forward.
947             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
948             \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
949         }
950         { \sort_return_swapped: }
951         {
952             \bool_if:nTF
953             {
954                 % The interesting case: both labels have a type...
955                 ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
956                 ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
957             }
958             {

```

```

959 % Here we send this to a couple of auxiliary functions for no
960 % other reason than to keep this long function a little less
961 % unreadable.
962 \tl_if_eq:NNTF
963   \l__zrefclever_label_type_a_tl
964   \l__zrefclever_label_type_b_tl
965   {
966     % ...and it's the same type.
967     \__zrefclever_sort_default_same_type:nn {#1} {#2}
968   }
969   {
970     % ...and they are different types.
971     \__zrefclever_sort_default_different_types:nn {#1} {#2}
972   }
973 }
974 {
975   % Neither of the labels has a type. We can't do much of
976   % meaningful here, but if it's the same counter, compare it.
977   \exp_args:Nxx \tl_if_eq:nnTF
978   { \zref@extractdefault {#1} { counter } { } }
979   { \zref@extractdefault {#2} { counter } { } }
980   {
981     \int_compare:nNnTF
982       { \zref@extractdefault {#1} { zc@cntval } {-1} }
983       >
984       { \zref@extractdefault {#2} { zc@cntval } {-1} }
985       { \sort_return_swapped: }
986       { \sort_return_same: }
987     }
988     { \sort_return_same: }
989   }
990 }
991 }
992 }

```

(End definition for __zrefclever_sort_default:nn.)

_zrefclever_sort_default_same_type:nn

```

993 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
994 {
995   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
996     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
997   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
998     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
999   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1000     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1001   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1002     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1003   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1004     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1005   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1006     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1007   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1008     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }

```

```

1009 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1010 { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1011
1012 \bool_set_false:N \l__zrefclever_sort_decided_bool
1013 % CHECK should I replace the tmp variables here?
1014 \tl_clear:N \l_tmpa_tl
1015 \tl_clear:N \l_tmpb_tl
1016 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1017 {
1018   \tl_set:Nx \l_tmpa_tl
1019   { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1020   \tl_set:Nx \l_tmpb_tl
1021   { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1022
1023   \bool_if:nTF
1024   {
1025     % Both are empty, meaning: neither labels have any (further)
1026     % ‘enclosing counters’ (left).
1027     \tl_if_empty_p:V \l_tmpa_tl &&
1028     \tl_if_empty_p:V \l_tmpb_tl
1029   }
1030   {
1031     \exp_args:Nxx \tl_if_eq:nTF
1032     { \zref@extractdefault {#1} { counter } { } }
1033     { \zref@extractdefault {#2} { counter } { } }
1034     {
1035       \bool_set_true:N \l__zrefclever_sort_decided_bool
1036       \int_compare:nNnTF
1037       { \zref@extractdefault {#1} { zc@cntval } {-1} }
1038       >
1039       { \zref@extractdefault {#2} { zc@cntval } {-1} }
1040       { \sort_return_swapped: }
1041       { \sort_return_same: }
1042     }
1043     {
1044       \msg_warning:nnnn { zref-clever }
1045       { counters-not-nested } {#1} {#2}
1046       \bool_set_true:N \l__zrefclever_sort_decided_bool
1047       \sort_return_same:
1048     }
1049   }
1050   {
1051     \bool_if:nTF
1052     {
1053       % ‘a’ is empty (and ‘b’ is not), meaning: ‘b’ is (possibly)
1054       % nested in ‘a’.
1055       \tl_if_empty_p:V \l_tmpa_tl
1056     }
1057     {
1058       \tl_set:Nx \l_tmpa_tl
1059       { {\zref@extractdefault {#1} { counter } { } } }
1060       \exp_args:NNx \tl_if_in:NnTF
1061       \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1062       {

```



```

1063         \bool_set_true:N \l__zrefclever_sort_decided_bool
1064         \sort_return_same:
1065     }
1066     {
1067         \msg_warning:nnnn { zref-clever }
1068         { counters-not-nested } {#1} {#2}
1069         \bool_set_true:N \l__zrefclever_sort_decided_bool
1070         \sort_return_same:
1071     }
1072 }
1073 {
1074     \bool_if:nTF
1075     {
1076         % 'b' is empty (and 'a' is not), meaning: 'a' is
1077         % (possibly) nested in 'b'.
1078         \tl_if_empty_p:V \l_tmpb_tl
1079     }
1080     {
1081         \tl_set:Nx \l_tmpb_tl
1082         { {\zref@extractdefault {#2} { counter } { }} }
1083         \exp_args:NNx \tl_if_in:NnTF
1084         \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1085         {
1086             \bool_set_true:N \l__zrefclever_sort_decided_bool
1087             \sort_return_swapped:
1088         }
1089         {
1090             \msg_warning:nnnn { zref-clever }
1091             { counters-not-nested } {#1} {#2}
1092             \bool_set_true:N \l__zrefclever_sort_decided_bool
1093             \sort_return_same:
1094         }
1095     }
1096 }
1097 % Neither is empty, meaning: we can (possibly) compare the
1098 % values of the current enclosing counter in the loop, if
1099 % they are equal, we are still in the loop, if they are
1100 % not, a sorting decision can be made directly.
1101 \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
1102 {
1103     \int_compare:nNnTF
1104     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1105     =
1106     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1107     {
1108         \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1109         { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1110         \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1111         { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1112         \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1113         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1114         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1115         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1116     }

```

```

1117         {
1118             \bool_set_true:N \l__zrefclever_sort_decided_bool
1119             \int_compare:nNnTF
1120                 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1121                 >
1122                 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1123                 { \sort_return_swapped: }
1124                 { \sort_return_same: }
1125         }
1126     }
1127     {
1128         \msg_warning:nnnn { zref-clever }
1129         { counters-not-nested } {#1} {#2}
1130         \bool_set_true:N \l__zrefclever_sort_decided_bool
1131         \sort_return_same:
1132     }
1133 }
1134 }
1135 }
1136 }
1137 }

```

(End definition for __zrefclever_sort_default_same_type:nn.)

_zrefclever_sort_default_different_types:nn

```

1138 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1139 {
1140     \int_zero:N \l__zrefclever_sort_prior_a_int
1141     \int_zero:N \l__zrefclever_sort_prior_b_int
1142     % \cs{l__zrefclever_typesort_seq} was stored in reverse sequence, and
1143     % we compute the sort priorities in the negative range, so that we can
1144     % implicitly rely on '0' being the "last value".
1145     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1146     {
1147         \tl_if_eq:nnTF {##2} {{othertypes}}
1148         {
1149             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1150             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1151             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1152             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1153         }
1154         {
1155             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1156             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1157             {
1158                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1159                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1160             }
1161         }
1162     }
1163     \bool_if:nTF
1164     {
1165         \int_compare_p:nNn
1166         { \l__zrefclever_sort_prior_a_int } <

```

```

1167         { \l__zrefclever_sort_prior_b_int }
1168     }
1169     { \sort_return_same: }
1170     {
1171         \bool_if:nTF
1172         {
1173             \int_compare_p:nNn
1174             { \l__zrefclever_sort_prior_a_int } >
1175             { \l__zrefclever_sort_prior_b_int }
1176         }
1177         { \sort_return_swapped: }
1178         {
1179             % Sort priorities are equal for different types: the type that
1180             % occurs first in 'labels', as given by the user, is kept (or
1181             % brought) forward.
1182             \seq_map_inline:Nn \l__zrefclever_label_types_seq
1183             {
1184                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1185                 { \seq_map_break:n { \sort_return_same: } }
1186                 {
1187                     \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1188                     { \seq_map_break:n { \sort_return_swapped: } }
1189                 }
1190             }
1191         }
1192     }
1193 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```

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

1194 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1195 {
1196     \int_compare:nNnTF
1197     { \zref@extractdefault {#1} { abspage } {-1} }
1198     >
1199     { \zref@extractdefault {#2} { abspage } {-1} }
1200     { \sort_return_swapped: }
1201     { \sort_return_same: }
1202 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

About possible alternatives to signal compression inhibition for individual labels, see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik,

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

Variables

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

```
1203 \bool_new:N \l__zrefclever_typeset_last_bool
1204 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

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

```
1205 \seq_new:N \l__zrefclever_typeset_labels_seq
1206 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1207 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1208 \tl_new:N \l__zrefclever_type_first_label_tl
1209 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

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

```
1210 \int_new:N \l__zrefclever_label_count_int
1211 \int_new:N \l__zrefclever_type_count_int
```

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

`\l_zrefclever_range_count_int` Range related auxiliary variables for `__zrefclever_typeset_refs:`. `\l_zrefclever_range_count_int` counts how many references/labels are in the current ongoing range. `\l_zrefclever_range_same_count_int` counts how many of the references in the current ongoing range are repeated ones. `\l_zrefclever_range_beg_label_tl` stores the label of the reference that starts a range. `\l_zrefclever_next_maybe_range_bool` signals whether the next element is in sequence to the current one. `\l_zrefclever_next_is_same_bool` signals whether the next element repeats the current one. `\l_zrefclever_range_inhibit_next_bool` allows to control/track compression inhibition of the next label.

```

1212 \int_new:N \l__zrefclever_range_count_int
1213 \int_new:N \l__zrefclever_range_same_count_int
1214 \tl_new:N \l__zrefclever_range_beg_label_tl
1215 \bool_new:N \l__zrefclever_next_maybe_range_bool
1216 \bool_new:N \l__zrefclever_next_is_same_bool
1217 \bool_new:N \l__zrefclever_range_inhibit_next_bool

```

(End definition for \l__zrefclever_range_count_int and others.)

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

```

1218 \tl_new:N \l__zrefclever_namefont_tl
1219 \tl_new:N \l__zrefclever_reffont_out_tl
1220 \tl_new:N \l__zrefclever_reffont_in_tl
1221
1222 \tl_new:N \l__zrefclever_namesep_tl
1223 \tl_new:N \l__zrefclever_rangesep_tl
1224 \tl_new:N \l__zrefclever_pairsep_tl
1225 \tl_new:N \l__zrefclever_listsep_tl
1226 \tl_new:N \l__zrefclever_lastsep_tl
1227 \tl_new:N \l__zrefclever_tpairsep_tl
1228 \tl_new:N \l__zrefclever_tlistsep_tl
1229 \tl_new:N \l__zrefclever_tlastsep_tl
1230 \tl_new:N \l__zrefclever_notesep_tl
1231 \tl_new:N \l__zrefclever_refpre_out_tl
1232 \tl_new:N \l__zrefclever_refpos_out_tl
1233 \tl_new:N \l__zrefclever_refpre_in_tl
1234 \tl_new:N \l__zrefclever_refpos_in_tl

```

(End definition for .)

\l__zrefclever_type_name_tl Auxiliary variables for __zrefclever_get_ref_first: and __zrefclever_type_name_setup:.

```

1235 \tl_new:N \l__zrefclever_type_name_tl
1236 \bool_new:N \l__zrefclever_name_in_link_bool
1237 \tl_new:N \l__zrefclever_name_format_tl
1238 \tl_new:N \l__zrefclever_name_format_fallback_tl

```

(End definition for \l__zrefclever_type_name_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

```

1239 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1240 {
1241   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zceref_labels_seq
1242   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1243   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1244   \tl_clear:N \l__zrefclever_type_first_label_tl
1245   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1246   \tl_clear:N \l__zrefclever_range_beg_label_tl
1247   \int_zero:N \l__zrefclever_label_count_int
1248   \int_zero:N \l__zrefclever_type_count_int
1249   \int_zero:N \l__zrefclever_range_count_int

```

```

1250 \int_zero:N \l__zrefclever_range_same_count_int
1251
1252 % Get not-type-specific separators and reppre/pos options.
1253 \__zrefclever_get_option_with_transl:nN {tpairsep} \l__zrefclever_tpairsep_tl
1254 \__zrefclever_get_option_with_transl:nN {tlistsep} \l__zrefclever_tlistsep_tl
1255 \__zrefclever_get_option_with_transl:nN {tlastsep} \l__zrefclever_tlastsep_tl
1256 \__zrefclever_get_option_with_transl:nN {notesep} \l__zrefclever_notesep_tl
1257
1258 % Set the font option for this zcref call.
1259 \l__zrefclever_ref_typeset_font_tl
1260
1261 % Loop over the label list in sequence.
1262 \bool_set_false:N \l__zrefclever_typeset_last_bool
1263 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1264 {
1265   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl
1266   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1267   {
1268     \tl_clear:N \l__zrefclever_label_b_tl
1269     \bool_set_true:N \l__zrefclever_typeset_last_bool
1270   }
1271   { \seq_get_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_b_tl }
1272
1273   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1274   {
1275     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1276     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1277   }
1278   {
1279     \tl_set:Nx \l__zrefclever_label_type_a_tl
1280     {
1281       \zref@extractdefault
1282       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1283     }
1284     \tl_set:Nx \l__zrefclever_label_type_b_tl
1285     {
1286       \zref@extractdefault
1287       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1288     }
1289   }
1290
1291   % First, we establish whether the ‘current label’ (i.e. ‘a’) is the
1292   % last one of its type. This can happen because the ‘next label’
1293   % (i.e. ‘b’) is of a different type (or different definition status),
1294   % or because we are at the end of the list.
1295   \bool_if:NTF \l__zrefclever_typeset_last_bool
1296   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1297   {
1298     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1299     {
1300       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1301       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1302       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1303     }

```

```

1304 {
1305     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1306     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1307     {
1308         % Neither is undefined, we must check the types.
1309         \bool_if:nTF
1310             % Both empty: same ‘type’.
1311             {
1312                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1313                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1314             }
1315             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1316             {
1317                 \bool_if:nTF
1318                     % Neither empty: compare types.
1319                     {
1320                         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1321                         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1322                     }
1323                     {
1324                         \tl_if_eq:NNTF
1325                             \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1326                             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1327                             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1328                     }
1329                     % One empty, the other not: different ‘types’.
1330                     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1331             }
1332         }
1333     }
1334 }
1335
1336 % Handle warnings in case of reference or type undefined.
1337 \zref@refused { \l__zrefclever_label_a_tl }
1338 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1339     {}
1340     {
1341         \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1342         {
1343             \msg_warning:nxx { zref-clever } { missing-type }
1344             { \l__zrefclever_label_a_tl }
1345         }
1346     }
1347
1348 % Get type-specific separators, refpre/pos and font options, once per
1349 % type.
1350 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1351 {
1352     \__zrefclever_get_option_plain:nN {namefont}          \l__zrefclever_namefont_tl
1353     \__zrefclever_get_option_plain:nN {reffont}          \l__zrefclever_reffont_out_tl
1354     \__zrefclever_get_option_plain:nN {reffont-in}       \l__zrefclever_reffont_in_tl
1355     \__zrefclever_get_option_with_transl:nN {namesep}    \l__zrefclever_namesep_tl
1356     \__zrefclever_get_option_with_transl:nN {rangesep}   \l__zrefclever_rangesep_tl
1357     \__zrefclever_get_option_with_transl:nN {pairsep}    \l__zrefclever_pairsep_tl

```

```

1358         \_zrefclever_get_option_with_transl:nN {listsep} \l__zrefclever_listsep_tl
1359         \_zrefclever_get_option_with_transl:nN {lastsep} \l__zrefclever_lastsep_tl
1360         \_zrefclever_get_option_with_transl:nN {refpre} \l__zrefclever_refpre_out_tl
1361         \_zrefclever_get_option_with_transl:nN {refpos} \l__zrefclever_refpos_out_tl
1362         \_zrefclever_get_option_with_transl:nN {refpre-in} \l__zrefclever_refpre_in_tl
1363         \_zrefclever_get_option_with_transl:nN {refpos-in} \l__zrefclever_refpos_in_tl
1364     }
1365
1366     % Here we send this to a couple of auxiliary functions for no other
1367     % reason than to keep this long function a little less unreadable.
1368     \bool_if:NTF \l__zrefclever_last_of_type_bool
1369     {
1370         % There exists no next label of the same type as the current.
1371         \_zrefclever_typeset_refs_aux_last_of_type:
1372     }
1373     {
1374         % There exists a next label of the same type as the current.
1375         \_zrefclever_typeset_refs_aux_not_last_of_type:
1376     }
1377 }
1378 }

```

(End definition for _zrefclever_typeset_refs:.)

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

```

1379 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_last_of_type:
1380 {
1381     % Process the current label to the current queue.
1382     \int_case:nnF { \l__zrefclever_label_count_int }
1383     {
1384         % It is the last label of its type, but also the first one, and that's
1385         % what matters here: just store it.
1386         { 0 }
1387         {
1388             \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1389             \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1390         }
1391
1392         % The last is the second: we have a pair (if not repeated).
1393         { 1 }
1394         {
1395             \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1396             {
1397                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1398                 {
1399                     \exp_not:V \l__zrefclever_pairsep_tl
1400                     \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1401                 }
1402             }
1403         }
1404     }
1405     % If neither the first, nor the second: we have the last label
1406     % on the current type list (if not repeated).
1407     {

```



```

1408 \int_case:nnF { \l__zrefclever_range_count_int }
1409 {
1410   % There was no range going on.
1411   {0}
1412   {
1413     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1414     {
1415       \exp_not:V \l__zrefclever_lastsep_tl
1416       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1417     }
1418   }
1419   % Last in the range is also the second in it.
1420   {1}
1421   {
1422     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1423     {
1424       % We know 'range_beg_label' is not empty, since this is the
1425       % second element in the range, but the third or more in the
1426       % type list.
1427       \exp_not:V \l__zrefclever_listsep_tl
1428       \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1429       \int_compare:nnF { \l__zrefclever_range_same_count_int } = {1}
1430       {
1431         \exp_not:V \l__zrefclever_lastsep_tl
1432         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1433       }
1434     }
1435   }
1436 }
1437 % Last in the range is third or more in it.
1438 {
1439   \int_case:nnF
1440   { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1441   {
1442     % Repetition, not a range.
1443     {0}
1444     {
1445       % If 'range_beg_label' is empty, it means it was also the
1446       % first of the type, and hence was already handled.
1447       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1448       {
1449         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1450         {
1451           \exp_not:V \l__zrefclever_lastsep_tl
1452           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1453         }
1454       }
1455     }
1456     % A 'range', but with no skipped value, treat as list.
1457     {1}
1458     {
1459       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1460       {
1461         % Ditto.

```

```

1462         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1463         {
1464             \exp_not:V \l__zrefclever_listsep_tl
1465             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1466         }
1467         \exp_not:V \l__zrefclever_lastsep_tl
1468         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1469     }
1470 }
1471 }
1472 {
1473     % An actual range.
1474     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1475     {
1476         % Ditto.
1477         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1478         {
1479             \exp_not:V \l__zrefclever_lastsep_tl
1480             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1481         }
1482         \exp_not:V \l__zrefclever_rangesep_tl
1483         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1484     }
1485 }
1486 }
1487 }
1488
1489 % Handle ‘‘range’’ option. The idea is simple: if the queue is not empty,
1490 % we replace it with the end of the range (or pair). We can still
1491 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1492 % be processing the last label of its type at this point.
1493 \bool_if:NT \l__zrefclever_typeset_range_bool
1494 {
1495     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1496     {
1497         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1498         { }
1499         {
1500             \msg_warning:nxx { zref-clever } { single-element-range }
1501             { \l__zrefclever_type_first_label_type_tl }
1502         }
1503     }
1504     {
1505         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1506         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1507         { }
1508         {
1509             \__zrefclever_labels_in_sequence:nn
1510             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
1511         }
1512         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1513         {
1514             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1515             { \exp_not:V \l__zrefclever_pairsep_tl }

```

```

1516         { \exp_not:V \l__zrefclever_rangesep_tl }
1517         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1518     }
1519 }
1520 }
1521
1522 % Now that the type is finished, we can add the name and the first ref to
1523 % the queue. Or, if ‘‘typset’’ option is not ‘‘both’’, handle it here
1524 % too.
1525 \__zrefclever_type_name_setup:
1526 \bool_if:nTF
1527 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1528 {
1529     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1530     { \__zrefclever_get_ref_first: }
1531 }
1532 {
1533     \bool_if:nTF
1534     { \l__zrefclever_typeset_ref_bool }
1535     {
1536         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1537         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1538     }
1539     {
1540         \bool_if:nTF
1541         { \l__zrefclever_typeset_name_bool }
1542         {
1543             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1544             {
1545                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1546                 {
1547                     \exp_not:N \group_begin:
1548                     \exp_not:V \l__zrefclever_namefont_tl
1549                     % It’s two ‘@s’, but escaped for DocStrip.
1550                     \exp_not:N \hyper@@link
1551                     {
1552                         \zref@ifrefcontainsprop
1553                         { \l__zrefclever_type_first_label_tl } { urluse }
1554                         {
1555                             \zref@extractdefault
1556                             { \l__zrefclever_type_first_label_tl }
1557                             { urluse } {}
1558                         }
1559                         {
1560                             \zref@extractdefault
1561                             { \l__zrefclever_type_first_label_tl }
1562                             { url } {}
1563                         }
1564                     }
1565                 }
1566                 \zref@extractdefault
1567                 { \l__zrefclever_type_first_label_tl } { anchor } {}
1568             }
1569             { \exp_not:V \l__zrefclever_type_name_tl }

```

```

1570         \exp_not:N \group_end:
1571     }
1572     {
1573         \exp_not:N \group_begin:
1574         \exp_not:V \l__zrefclever_namefont_tl
1575         \exp_not:V \l__zrefclever_type_name_tl
1576         \exp_not:N \group_end:
1577     }
1578 }
1579 }
1580 {
1581     % This case would correspond to "typeset=none" but should not
1582     % happen, given the options are set up to typeset at least one
1583     % of "ref" or "name", but a sensible fallback, equal to the
1584     % behavior of ‘‘both’’.
1585     \tl_put_left:Nx
1586         \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1587 }
1588 }
1589 }
1590
1591 % Typeset the previous type, if there is one.
1592 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1593 {
1594     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1595     { \l__zrefclever_tlistsep_tl }
1596     \l__zrefclever_typeset_queue_prev_tl
1597 }
1598
1599 % Wrap up loop, or prepare for next iteration.
1600 \bool_if:NTF \l__zrefclever_typeset_last_bool
1601 {
1602     % We are finishing, typeset the current queue.
1603     \int_case:nnF { \l__zrefclever_type_count_int }
1604     {
1605         % Single type.
1606         { 0 }
1607         { \l__zrefclever_typeset_queue_curr_tl }
1608         % Pair of types.
1609         { 1 }
1610         {
1611             \l__zrefclever_tpairsep_tl
1612             \l__zrefclever_typeset_queue_curr_tl
1613         }
1614     }
1615     {
1616         % Last in list of types.
1617         \l__zrefclever_tlastsep_tl
1618         \l__zrefclever_typeset_queue_curr_tl
1619     }
1620 }
1621 {
1622     % There are further labels, set variables for next iteration.
1623     \tl_set_eq:NN

```

```

1624         \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1625     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1626     \tl_clear:N \l__zrefclever_type_first_label_tl
1627     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1628     \tl_clear:N \l__zrefclever_range_beg_label_tl
1629     \int_zero:N \l__zrefclever_label_count_int
1630     \int_incr:N \l__zrefclever_type_count_int
1631     \int_zero:N \l__zrefclever_range_count_int
1632     \int_zero:N \l__zrefclever_range_same_count_int
1633 }
1634 }

```

(End definition for __zrefclever_typeset_refs_aux_last_of_type:.)

efclever_typeset_refs_aux_not_last_of_type: Handles typesetting of when the current label is not the last of its type.

```

1635 \cs_new_protected:Npn \__zrefclever_typeset_refs_aux_not_last_of_type:
1636 {
1637     % Signal if next label may form a range with the current one (of
1638     % course, only considered if compression is enabled in the first
1639     % place).
1640     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1641     \bool_set_false:N \l__zrefclever_next_is_same_bool
1642     \bool_lazy_and:nnT
1643     { \l__zrefclever_typeset_compress_bool }
1644     % Currently no-op, but kept as ‘‘handle’’ to inhibit compression of
1645     % individual labels.
1646     { ! \l__zrefclever_range_inhibit_next_bool }
1647     {
1648         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1649         { }
1650         {
1651             \__zrefclever_labels_in_sequence:nn
1652             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1653         }
1654     }
1655
1656     % Process the current label to the current queue.
1657     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1658     {
1659         % Current label is the first of its type (also not the last, but it
1660         % doesn't matter here): just store the label.
1661         \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1662         \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1663
1664         % If the next label may be part of a range, we set ‘range_beg_label’
1665         % to ‘‘empty’’ (we deal with it as the ‘‘first’’, and must do it
1666         % there, to handle hyperlinking), but also step the range counters.
1667         \bool_if:NT \l__zrefclever_next_maybe_range_bool
1668         {
1669             \tl_clear:N \l__zrefclever_range_beg_label_tl
1670             \int_incr:N \l__zrefclever_range_count_int
1671             \bool_if:NT \l__zrefclever_next_is_same_bool
1672             { \int_incr:N \l__zrefclever_range_same_count_int }
1673         }
1674     }

```

```

1674 }
1675 {
1676 % Current label is neither the first (nor the last) of its
1677 % type.
1678 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1679 {
1680 % Starting, or continuing a range.
1681 \int_compare:nNnTF
1682 { \l__zrefclever_range_count_int } = {0}
1683 {
1684 % There was no range going, we are starting one.
1685 \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1686 \int_incr:N \l__zrefclever_range_count_int
1687 \bool_if:NT \l__zrefclever_next_is_same_bool
1688 { \int_incr:N \l__zrefclever_range_same_count_int }
1689 }
1690 {
1691 % Second or more in the range, but not the last.
1692 \int_incr:N \l__zrefclever_range_count_int
1693 \bool_if:NT \l__zrefclever_next_is_same_bool
1694 { \int_incr:N \l__zrefclever_range_same_count_int }
1695 }
1696 }
1697 {
1698 % Next element is not in sequence, meaning: there was no range, or
1699 % we are closing one.
1700 \int_case:nnF { \l__zrefclever_range_count_int }
1701 {
1702 % There was no range going on.
1703 {0}
1704 {
1705 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1706 {
1707 \exp_not:V \l__zrefclever_listsep_tl
1708 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1709 }
1710 }
1711 % Last is second in the range: if 'range_same_count' is also
1712 % '1', it's a repetition (drop it), otherwise, it's a 'pair
1713 % within a list', treat as list.
1714 {1}
1715 {
1716 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1717 {
1718 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1719 {
1720 \exp_not:V \l__zrefclever_listsep_tl
1721 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1722 }
1723 \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1724 {
1725 \exp_not:V \l__zrefclever_listsep_tl
1726 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1727 }

```

```

1728     }
1729   }
1730 }
1731 {
1732   % Last is third or more in the range: if 'range_count' and
1733   % 'range_same_count' are the same, its a repetition (drop it),
1734   % if they differ by '1', its a list, if they differ by more,
1735   % it is a real range.
1736   \int_case:nnF
1737     { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1738     {
1739       {0}
1740       {
1741         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1742         {
1743           \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1744           {
1745             \exp_not:V \l__zrefclever_listsep_tl
1746             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1747           }
1748         }
1749       }
1750       {1}
1751       {
1752         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1753         {
1754           \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1755           {
1756             \exp_not:V \l__zrefclever_listsep_tl
1757             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1758           }
1759           \exp_not:V \l__zrefclever_listsep_tl
1760           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1761         }
1762       }
1763     }
1764   {
1765     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1766     {
1767       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1768       {
1769         \exp_not:V \l__zrefclever_listsep_tl
1770         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1771       }
1772       \exp_not:V \l__zrefclever_rangeseq_tl
1773       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1774     }
1775   }
1776 }
1777 % Reset counters.
1778 \int_zero:N \l__zrefclever_range_count_int
1779 \int_zero:N \l__zrefclever_range_same_count_int
1780 }
1781 }

```

```

1782     % Step label counter for next iteration.
1783     \int_incr:N \l__zrefclever_label_count_int
1784 }

```

(End definition for __zrefclever_typeset_refs_aux_not_last_of_type:.)

Aux functions

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

```

1785 \cs_new:Npn \__zrefclever_get_ref:n #1
1786 {
1787   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
1788   {
1789     \bool_if:nTF
1790     { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
1791     {
1792       \exp_not:N \group_begin:
1793       \exp_not:V \l__zrefclever_reffont_out_tl
1794       \exp_not:V \l__zrefclever_refpre_out_tl
1795       \exp_not:N \group_begin:
1796       \exp_not:V \l__zrefclever_reffont_in_tl
1797       % It's two '@s', but escaped for DocStrip.
1798       \exp_not:N \hyper@@link
1799       {
1800         \zref@ifrefcontainsprop {#1} { urluse }
1801         { \zref@extractdefault {#1} { urluse } {} }
1802         { \zref@extractdefault {#1} { url } {} }
1803       }
1804       { \zref@extractdefault {#1} { anchor } {} }
1805       {
1806         \exp_not:V \l__zrefclever_refpre_in_tl
1807         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
1808         \exp_not:V \l__zrefclever_refpos_in_tl
1809       }
1810       \exp_not:N \group_end:
1811       \exp_not:V \l__zrefclever_refpos_out_tl
1812       \exp_not:N \group_end:
1813     }
1814     {
1815       \exp_not:N \group_begin:
1816       \exp_not:V \l__zrefclever_reffont_out_tl
1817       \exp_not:V \l__zrefclever_refpre_out_tl
1818       \exp_not:N \group_begin:
1819       \exp_not:V \l__zrefclever_reffont_in_tl
1820       \exp_not:V \l__zrefclever_refpre_in_tl
1821       \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
1822       \exp_not:V \l__zrefclever_refpos_in_tl
1823       \exp_not:N \group_end:
1824       \exp_not:V \l__zrefclever_refpos_out_tl

```



```

1825         \exp_not:N \group_end:
1826     }
1827 }
1828 { \exp_not:N \zref@default }
1829 }
1830 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

_zrefclever_type_name_setup: Auxiliary function to __zrefclever_typeset_refs:. Sets the type name variable \l__zrefclever_type_name_tl. When it cannot be found, clears it.

```

1831 \cs_new_protected:Npn \__zrefclever_type_name_setup:
1832 {
1833     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1834     { \tl_clear:N \l__zrefclever_type_name_tl }
1835     {
1836         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
1837         { \tl_clear:N \l__zrefclever_type_name_tl }
1838         {

```

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

```

1839         \bool_lazy_or:nnTF
1840         { \l__zrefclever_capitalize_bool }
1841         {
1842             \l__zrefclever_capitalize_first_bool &&
1843             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
1844         }
1845         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
1846         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
1847         % If the queue is empty, we have a singular, otherwise, plural.
1848         \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1849         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
1850         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
1851         \bool_lazy_and:nnTF
1852         { \l__zrefclever_abbrev_bool }
1853         {
1854             ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
1855             ! \l__zrefclever_noabbrev_first_bool
1856         }
1857         {
1858             \tl_set:NV \l__zrefclever_name_format_fallback_tl \l__zrefclever_name_format
1859             \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
1860         }
1861         { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
1862
1863         \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
1864         {
1865             \prop_get:cVNF
1866             { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
1867             \l__zrefclever_name_format_tl
1868             \l__zrefclever_type_name_tl
1869             {
1870                 \__zrefclever_if_transl:xxTF
1871                 { \l__zrefclever_ref_language_tl }
1872                 {

```

```

1873         zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1874         \l__zrefclever_name_format_tl
1875     }
1876     {
1877         \__zrefclever_get_transl:nxx { \l__zrefclever_type_name_tl }
1878         { \l__zrefclever_ref_language_tl }
1879         {
1880             zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1881             \l__zrefclever_name_format_tl
1882         }
1883     }
1884     {
1885         \tl_clear:N \l__zrefclever_type_name_tl
1886         \msg_warning:nxx { zref-clever } { missing-name }
1887         { \l__zrefclever_type_first_label_type_tl }
1888     }
1889 }
1890 }
1891 {
1892     \prop_get:cVNF
1893     { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
1894     \l__zrefclever_name_format_tl
1895     \l__zrefclever_type_name_tl
1896     {
1897         \prop_get:cVNF
1898         { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options
1899         \l__zrefclever_name_format_fallback_tl
1900         \l__zrefclever_type_name_tl
1901         {
1902             \__zrefclever_if_transl:xxTF
1903             { \l__zrefclever_ref_language_tl }
1904             {
1905                 zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1906                 \l__zrefclever_name_format_tl
1907             }
1908             {
1909                 \__zrefclever_get_transl:nxx { \l__zrefclever_type_name_tl }
1910                 { \l__zrefclever_ref_language_tl }
1911                 {
1912                     zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1913                     \l__zrefclever_name_format_tl
1914                 }
1915             }
1916         }
1917         \__zrefclever_if_transl:xxTF
1918         { \l__zrefclever_ref_language_tl }
1919         {
1920             zrefclever-type- \l__zrefclever_type_first_label_type_tl -
1921             \l__zrefclever_name_format_fallback_tl
1922         }
1923         {
1924             \__zrefclever_get_transl:nxx { \l__zrefclever_type_name_tl }
1925             { \l__zrefclever_ref_language_tl }
1926             {

```

```

1927             zrefclever-type- \l__zrefclever_type_first_label_type_tl
1928             \l__zrefclever_name_format_fallback_tl
1929         }
1930     }
1931     {
1932         \tl_clear:N \l__zrefclever_type_name_tl
1933         \msg_warning:nnx { zref-clever } { missing-name }
1934         { \l__zrefclever_type_first_label_type_tl }
1935     }
1936 }
1937 }
1938 }
1939 }
1940 }
1941 }

```

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

```

1942 \bool_lazy_any:nTF
1943 {
1944     { ! \l__zrefclever_use_hyperref_bool }
1945     { \l__zrefclever_link_star_bool }
1946     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
1947     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
1948 }
1949 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
1950 {
1951     \bool_lazy_any:nTF
1952     {
1953         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
1954         {
1955             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
1956             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
1957         }
1958         {
1959             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
1960             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
1961             \l__zrefclever_typeset_last_bool &&
1962             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
1963         }
1964     }
1965     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
1966     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
1967 }
1968 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_get_ref_first: Auxiliary function to __zrefclever_typeset_refs:. Handles a complete “ref-block”, including “pre” and “pos” elements, *hyperlinking*, and the reference type “name”. For use on the first reference of each type.

```

1969 \cs_new:Npn \__zrefclever_get_ref_first:
1970 {
1971     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1972     { \exp_not:N \zref@default }
1973     {

```

```

1974 \bool_if:NTF \l__zrefclever_name_in_link_bool
1975 {
1976   \zref@ifrefcontainsprop
1977   { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
1978   {
1979     % It's two '@s', but escaped for DocStrip.
1980     \exp_not:N \hyper@@link
1981     {
1982       \zref@ifrefcontainsprop
1983       { \l__zrefclever_type_first_label_tl } { urluse }
1984       {
1985         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
1986         { urluse } {}
1987       }
1988       {
1989         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
1990         { url } {}
1991       }
1992     }
1993     {
1994       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
1995       { anchor } {}
1996     }
1997     {
1998       \exp_not:N \group_begin:
1999       \exp_not:V \l__zrefclever_namefont_tl
2000       \exp_not:V \l__zrefclever_type_name_tl
2001       \exp_not:N \group_end:
2002       \exp_not:V \l__zrefclever_namesep_tl
2003       \exp_not:N \group_begin:
2004       \exp_not:V \l__zrefclever_reffont_out_tl
2005       \exp_not:V \l__zrefclever_refpre_out_tl
2006       \exp_not:N \group_begin:
2007       \exp_not:V \l__zrefclever_reffont_in_tl
2008       \exp_not:V \l__zrefclever_refpre_in_tl
2009       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2010       { \l__zrefclever_ref_property_tl } {}
2011       \exp_not:V \l__zrefclever_refpos_in_tl
2012       \exp_not:N \group_end:
2013       % hyperlink makes it's own group, we'd like to close the
2014       % 'refpre-out' group after 'refpos-out', but... we close
2015       % it here, and give the trailing 'refpos-out' its own
2016       % group. This will result that formatting given to
2017       % 'refpre-out' will not reach 'refpos-out', but I see no
2018       % alternative, and this has to be handled specially.
2019       \exp_not:N \group_end:
2020     }
2021     \exp_not:N \group_begin:
2022     % Ditto: special treatment.
2023     \exp_not:V \l__zrefclever_reffont_out_tl
2024     \exp_not:V \l__zrefclever_refpos_out_tl
2025     \exp_not:N \group_end:
2026   }
2027   {

```

```

2028         \exp_not:N \group_begin:
2029         \exp_not:V \l__zrefclever_namefont_tl
2030         \exp_not:V \l__zrefclever_type_name_tl
2031         \exp_not:N \group_end:
2032         \exp_not:V \l__zrefclever_namesep_tl
2033         \exp_not:N \zref@default
2034     }
2035 }
2036 {
2037     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2038     {
2039         \exp_not:N \zref@default
2040         \exp_not:V \l__zrefclever_namesep_tl
2041     }
2042     {
2043         \exp_not:N \group_begin:
2044         \exp_not:V \l__zrefclever_namefont_tl
2045         \exp_not:V \l__zrefclever_type_name_tl
2046         \exp_not:N \group_end:
2047         \exp_not:V \l__zrefclever_namesep_tl
2048     }
2049     \zref@ifrefcontainsprop
2050     { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2051     {
2052         \bool_if:nTF
2053         {
2054             \l__zrefclever_use_hyperref_bool &&
2055             ! \l__zrefclever_link_star_bool
2056         }
2057         {
2058             \exp_not:N \group_begin:
2059             \exp_not:V \l__zrefclever_reffont_out_tl
2060             \exp_not:V \l__zrefclever_refpre_out_tl
2061             \exp_not:N \group_begin:
2062             \exp_not:V \l__zrefclever_reffont_in_tl
2063             % It's two '@s', but escaped for DocStrip.
2064             \exp_not:N \hyper@@link
2065             {
2066                 \zref@ifrefcontainsprop
2067                 { \l__zrefclever_type_first_label_tl } { urluse }
2068                 {
2069                     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2070                     { urluse } {}
2071                 }
2072                 {
2073                     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2074                     { url } {}
2075                 }
2076             }
2077         }
2078         {
2079             \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2080             { anchor } {}
2081         }

```

```

2082         \exp_not:V \l__zrefclever_refpre_in_tl
2083         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2084         { \l__zrefclever_ref_property_tl } {}
2085         \exp_not:V \l__zrefclever_refpos_in_tl
2086     }
2087     \exp_not:N \group_end:
2088     \exp_not:V \l__zrefclever_refpos_out_tl
2089     \exp_not:N \group_end:
2090 }
2091 {
2092     \exp_not:N \group_begin:
2093     \exp_not:V \l__zrefclever_reffont_out_tl
2094     \exp_not:V \l__zrefclever_refpre_out_tl
2095     \exp_not:N \group_begin:
2096     \exp_not:V \l__zrefclever_reffont_in_tl
2097     \exp_not:V \l__zrefclever_refpre_in_tl
2098     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2099     { \l__zrefclever_ref_property_tl } {}
2100     \exp_not:V \l__zrefclever_refpos_in_tl
2101     \exp_not:N \group_end:
2102     \exp_not:V \l__zrefclever_refpos_out_tl
2103     \exp_not:N \group_end:
2104 }
2105 }
2106 { \exp_not:N \zref@default }
2107 }
2108 }
2109 }

```

(End definition for __zrefclever_get_ref_first:.)

_zrefclever_get_option_with_transl:nN

```

2110 % \Arg{option} \Arg{var to store result}
2111 \cs_new_protected:Npn \__zrefclever_get_option_with_transl:nN #1#2
2112 {
2113     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2114     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2115     {
2116         % If not found, try the type specific options.
2117         \bool_lazy_all:nTF
2118         {
2119             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2120             {
2121                 \prop_if_exist_p:c
2122                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2123             }
2124             {
2125                 \prop_if_in_p:cn
2126                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2127             }
2128         }
2129         {
2130             \prop_get:cnN
2131             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2

```

```

2132     }
2133     {
2134         % If not found, try the type specific translations.
2135         \__zrefclever_if_transl:xxTF
2136         { \l__zrefclever_ref_language_tl }
2137         { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
2138         {
2139             \__zrefclever_get_transl:nxx {#2}
2140             { \l__zrefclever_ref_language_tl }
2141             { zrefclever-type- \l__zrefclever_label_type_a_tl - #1 }
2142         }
2143         {
2144             % If not found, try general translations. We are not
2145             % controlling for their existence, but we must make sure all
2146             % options being retrieved with
2147             % \cs{__zrefclever_get_option_with_transl:nN} have their values set for
2148             % ‘English’ and ‘fallback’.
2149             \__zrefclever_get_transl:nxx {#2}
2150             { \l__zrefclever_ref_language_tl }
2151             { zrefclever-default- #1 }
2152         }
2153     }
2154 }
2155 }

```

(End definition for __zrefclever_get_option_with_transl:nN.)

__zrefclever_get_option_plain:nN

```

2156 \cs_new_protected:Npn \__zrefclever_get_option_plain:nN #1#2
2157 {
2158     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2159     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2160     {
2161         % If not found, try the type specific options.
2162         \bool_lazy_and:nnTF
2163         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2164         {
2165             \prop_if_exist_p:c
2166             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2167         }
2168         {
2169             \prop_get:cnNF
2170             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2171             { \tl_clear:N #2 }
2172         }
2173         { \tl_clear:N #2 }
2174     }
2175 }

```

(End definition for __zrefclever_get_option_plain:nN.)

__zrefclever_labels_in_sequence:nn

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

```

2176 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2177 {
2178   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2179   {
2180     \exp_args:Nxx \tl_if_eq:nnT
2181     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2182     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2183     {
2184       \int_compare:nNnTF
2185       { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2186       =
2187       { \zref@extractdefault {#2} { zc@pgval } {-1} }
2188       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2189       {
2190         \int_compare:nNnT
2191         { \zref@extractdefault {#1} { zc@pgval } {-1} }
2192         =
2193         { \zref@extractdefault {#2} { zc@pgval } {-1} }
2194         {
2195           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2196           \bool_set_true:N \l__zrefclever_next_is_same_bool
2197         }
2198       }
2199     }
2200   }
2201   {
2202     \exp_args:Nxx \tl_if_eq:nnT
2203     { \zref@extractdefault {#1} { counter } { } }
2204     { \zref@extractdefault {#2} { counter } { } }
2205     {
2206       \exp_args:Nxx \tl_if_eq:nnT
2207       { \zref@extractdefault {#1} { zc@enclval } { } }
2208       { \zref@extractdefault {#2} { zc@enclval } { } }
2209       {
2210         \int_compare:nNnTF
2211         { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
2212         =
2213         { \zref@extractdefault {#2} { zc@cntval } {-1} }
2214         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2215         {
2216           \int_compare:nNnT
2217           { \zref@extractdefault {#1} { zc@cntval } {-1} }
2218           =
2219           { \zref@extractdefault {#2} { zc@cntval } {-1} }
2220           {
2221             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2222             \bool_set_true:N \l__zrefclever_next_is_same_bool
2223           }
2224         }
2225       }
2226     }
2227   }
2228 }

```

(End definition for __zrefclever_labels_in_sequence:nn.)

9 Special handling

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

9.1 `\appendix`

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

9.2 `\newtheorem`

9.3 `enumitem` package

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

10 Translations

10.1 Fallback

All options retrieved with `__zrefclever_get_option_with_transl:nN` must have their values set for ‘fallback’, even if to empty values, since this is what will be retrieved if `babel` or `polyglossia` is loaded and sets a language which `zref-clever` does not know. And `translations` typesets the *key* as a last resource fallback, which we don’t want to happen. On the other hand, type-specific options are not looked for in ‘fallback’.

```
2229 \__zrefclever_declare_fallback_transl:nn { namesep } {\nobreakspace}
2230 \__zrefclever_declare_fallback_transl:nn { pairsep } {,~}
2231 \__zrefclever_declare_fallback_transl:nn { listsep } {,~}
2232 \__zrefclever_declare_fallback_transl:nn { lastsep } {,~}
2233 \__zrefclever_declare_fallback_transl:nn { tpairsep } {,~}
2234 \__zrefclever_declare_fallback_transl:nn { tlistsep } {,~}
2235 \__zrefclever_declare_fallback_transl:nn { tlastsep } {,~}
2236 \__zrefclever_declare_fallback_transl:nn { notesep } {-}
2237 \__zrefclever_declare_fallback_transl:nn { rangesep } {\textendash}
2238 \__zrefclever_declare_fallback_transl:nn { refpre } {}
2239 \__zrefclever_declare_fallback_transl:nn { refpos } {}
2240 \__zrefclever_declare_fallback_transl:nn { refpre-in } {}
2241 \__zrefclever_declare_fallback_transl:nn { refpos-in } {}
```

2242 `\package`

10.2 English

All options retrieved with `__zrefclever_get_option_with_transl:nN` must have their values set for ‘English’, since this is what will be retrieved if no language package is loaded. In other words, ‘English’ is the translations equivalent of ‘fallback’ when no language package is present.

```

2243 {*lang-english}

2244 \ProvideDictionaryFor{English}{zref-clever}
2245
2246 \zcDicDefaultTransl{namesep}{\nobreakspace}
2247 \zcDicDefaultTransl{pairsep}{~and\nobreakspace}
2248 \zcDicDefaultTransl{listsep}{,~}
2249 \zcDicDefaultTransl{lastsep}{~and\nobreakspace}
2250 \zcDicDefaultTransl{tpairsep}{~and\nobreakspace}
2251 \zcDicDefaultTransl{tlistsep}{,~}
2252 \zcDicDefaultTransl{tlastsep}{,~and\nobreakspace}
2253 \zcDicDefaultTransl{notesep}{~}
2254 \zcDicDefaultTransl{rangesep}{~to\nobreakspace}
2255 \zcDicDefaultTransl{refpre}{}
2256 \zcDicDefaultTransl{refpos}{}
2257 \zcDicDefaultTransl{refpre-in}{}
2258 \zcDicDefaultTransl{refpos-in}{}
2259
2260 \zcDicTypeTransl{part}{Name-sg}{Part}
2261 \zcDicTypeTransl{part}{name-sg}{part}
2262 \zcDicTypeTransl{part}{Name-pl}{Parts}
2263 \zcDicTypeTransl{part}{name-pl}{parts}
2264
2265 \zcDicTypeTransl{chapter}{Name-sg}{Chapter}
2266 \zcDicTypeTransl{chapter}{name-sg}{chapter}
2267 \zcDicTypeTransl{chapter}{Name-pl}{Chapters}
2268 \zcDicTypeTransl{chapter}{name-pl}{chapters}
2269
2270 \zcDicTypeTransl{section}{Name-sg}{Section}
2271 \zcDicTypeTransl{section}{name-sg}{section}
2272 \zcDicTypeTransl{section}{Name-pl}{Sections}
2273 \zcDicTypeTransl{section}{name-pl}{sections}
2274
2275 \zcDicTypeTransl{paragraph}{Name-sg}{Paragraph}
2276 \zcDicTypeTransl{paragraph}{name-sg}{paragraph}
2277 \zcDicTypeTransl{paragraph}{Name-pl}{Paragraphs}
2278 \zcDicTypeTransl{paragraph}{name-pl}{paragraphs}
2279 \zcDicTypeTransl{paragraph}{Name-sg-ab}{Par.}
2280 \zcDicTypeTransl{paragraph}{name-sg-ab}{par.}
2281 \zcDicTypeTransl{paragraph}{Name-pl-ab}{Par.}
2282 \zcDicTypeTransl{paragraph}{name-pl-ab}{par.}
2283
2284 \zcDicTypeTransl{appendix}{Name-sg}{Appendix}
2285 \zcDicTypeTransl{appendix}{name-sg}{appendix}
2286 \zcDicTypeTransl{appendix}{Name-pl}{Appendices}
2287 \zcDicTypeTransl{appendix}{name-pl}{appendices}

```

2288
 2289 \zcDicTypeTransl{page}{Name-sg}{Page}
 2290 \zcDicTypeTransl{page}{name-sg}{page}
 2291 \zcDicTypeTransl{page}{Name-pl}{Pages}
 2292 \zcDicTypeTransl{page}{name-pl}{pages}
 2293 \zcDicTypeTransl{page}{name-sg-ab}{p.}
 2294 \zcDicTypeTransl{page}{name-pl-ab}{pp.}
 2295
 2296 \zcDicTypeTransl{line}{Name-sg}{Line}
 2297 \zcDicTypeTransl{line}{name-sg}{line}
 2298 \zcDicTypeTransl{line}{Name-pl}{Lines}
 2299 \zcDicTypeTransl{line}{name-pl}{lines}
 2300
 2301 \zcDicTypeTransl{figure}{Name-sg}{Figure}
 2302 \zcDicTypeTransl{figure}{name-sg}{figure}
 2303 \zcDicTypeTransl{figure}{Name-pl}{Figures}
 2304 \zcDicTypeTransl{figure}{name-pl}{figures}
 2305 \zcDicTypeTransl{figure}{Name-sg-ab}{Fig.}
 2306 \zcDicTypeTransl{figure}{name-sg-ab}{fig.}
 2307 \zcDicTypeTransl{figure}{Name-pl-ab}{Figs.}
 2308 \zcDicTypeTransl{figure}{name-pl-ab}{figs.}
 2309
 2310 \zcDicTypeTransl{table}{Name-sg}{Table}
 2311 \zcDicTypeTransl{table}{name-sg}{table}
 2312 \zcDicTypeTransl{table}{Name-pl}{Tables}
 2313 \zcDicTypeTransl{table}{name-pl}{tables}
 2314
 2315 \zcDicTypeTransl{item}{Name-sg}{Item}
 2316 \zcDicTypeTransl{item}{name-sg}{item}
 2317 \zcDicTypeTransl{item}{Name-pl}{Items}
 2318 \zcDicTypeTransl{item}{name-pl}{items}
 2319
 2320 \zcDicTypeTransl{footnote}{Name-sg}{Footnote}
 2321 \zcDicTypeTransl{footnote}{name-sg}{footnote}
 2322 \zcDicTypeTransl{footnote}{Name-pl}{Footnotes}
 2323 \zcDicTypeTransl{footnote}{name-pl}{footnotes}
 2324
 2325 \zcDicTypeTransl{note}{Name-sg}{Note}
 2326 \zcDicTypeTransl{note}{name-sg}{note}
 2327 \zcDicTypeTransl{note}{Name-pl}{Notes}
 2328 \zcDicTypeTransl{note}{name-pl}{notes}
 2329
 2330 \zcDicTypeTransl{equation}{Name-sg}{Equation}
 2331 \zcDicTypeTransl{equation}{name-sg}{equation}
 2332 \zcDicTypeTransl{equation}{Name-pl}{Equations}
 2333 \zcDicTypeTransl{equation}{name-pl}{equations}
 2334 \zcDicTypeTransl{equation}{Name-sg-ab}{Eq.}
 2335 \zcDicTypeTransl{equation}{name-sg-ab}{eq.}
 2336 \zcDicTypeTransl{equation}{Name-pl-ab}{Eqs.}
 2337 \zcDicTypeTransl{equation}{name-pl-ab}{eqs.}
 2338 \zcDicTypeTransl{equation}{refpre-in}{(}
 2339 \zcDicTypeTransl{equation}{refpos-in}{)}
 2340
 2341 \zcDicTypeTransl{theorem}{Name-sg}{Theorem}

2342 \zcDicTypeTransl{theorem}{Name-sg}{theorem}
2343 \zcDicTypeTransl{theorem}{Name-pl}{Theorems}
2344 \zcDicTypeTransl{theorem}{name-pl}{theorems}
2345
2346 \zcDicTypeTransl{lemma}{Name-sg}{Lemma}
2347 \zcDicTypeTransl{lemma}{name-sg}{lemma}
2348 \zcDicTypeTransl{lemma}{Name-pl}{Lemmas}
2349 \zcDicTypeTransl{lemma}{name-pl}{lemmas}
2350
2351 \zcDicTypeTransl{corollary}{Name-sg}{Corollary}
2352 \zcDicTypeTransl{corollary}{name-sg}{corollary}
2353 \zcDicTypeTransl{corollary}{Name-pl}{Corollaries}
2354 \zcDicTypeTransl{corollary}{name-pl}{corollaries}
2355
2356 \zcDicTypeTransl{proposition}{Name-sg}{Proposition}
2357 \zcDicTypeTransl{proposition}{name-sg}{proposition}
2358 \zcDicTypeTransl{proposition}{Name-pl}{Propositions}
2359 \zcDicTypeTransl{proposition}{name-pl}{propositions}
2360
2361 \zcDicTypeTransl{definition}{Name-sg}{Definition}
2362 \zcDicTypeTransl{definition}{name-sg}{definition}
2363 \zcDicTypeTransl{definition}{Name-pl}{Definitions}
2364 \zcDicTypeTransl{definition}{name-pl}{definitions}
2365
2366 \zcDicTypeTransl{proof}{Name-sg}{Proof}
2367 \zcDicTypeTransl{proof}{name-sg}{proof}
2368 \zcDicTypeTransl{proof}{Name-pl}{Proofs}
2369 \zcDicTypeTransl{proof}{name-pl}{proofs}
2370
2371 \zcDicTypeTransl{result}{Name-sg}{Result}
2372 \zcDicTypeTransl{result}{name-sg}{result}
2373 \zcDicTypeTransl{result}{Name-pl}{Results}
2374 \zcDicTypeTransl{result}{name-pl}{results}
2375
2376 \zcDicTypeTransl{example}{Name-sg}{Example}
2377 \zcDicTypeTransl{example}{name-sg}{example}
2378 \zcDicTypeTransl{example}{Name-pl}{Examples}
2379 \zcDicTypeTransl{example}{name-pl}{examples}
2380
2381 \zcDicTypeTransl{remark}{Name-sg}{Remark}
2382 \zcDicTypeTransl{remark}{name-sg}{remark}
2383 \zcDicTypeTransl{remark}{Name-pl}{Remarks}
2384 \zcDicTypeTransl{remark}{name-pl}{remarks}
2385
2386 \zcDicTypeTransl{algorithm}{Name-sg}{Algorithm}
2387 \zcDicTypeTransl{algorithm}{name-sg}{algorithm}
2388 \zcDicTypeTransl{algorithm}{Name-pl}{Algorithms}
2389 \zcDicTypeTransl{algorithm}{name-pl}{algorithms}
2390
2391 \zcDicTypeTransl{listing}{Name-sg}{Listing}
2392 \zcDicTypeTransl{listing}{name-sg}{listing}
2393 \zcDicTypeTransl{listing}{Name-pl}{Listings}
2394 \zcDicTypeTransl{listing}{name-pl}{listings}
2395

```

2396 \zcDicTypeTransl{exercise}{Name-sg}{Exercise}
2397 \zcDicTypeTransl{exercise}{name-sg}{exercise}
2398 \zcDicTypeTransl{exercise}{Name-pl}{Exercises}
2399 \zcDicTypeTransl{exercise}{name-pl}{exercises}
2400
2401 \zcDicTypeTransl{solution}{Name-sg}{Solution}
2402 \zcDicTypeTransl{solution}{name-sg}{solution}
2403 \zcDicTypeTransl{solution}{Name-pl}{Solutions}
2404 \zcDicTypeTransl{solution}{name-pl}{solutions}
2405 </lang-english>

```

10.3 German

```

2406 <*lang-german>
2407 \ProvideDictionaryFor{German}{zref-clever}
2408
2409 \zcDicDefaultTransl{namesep}{\nobreakspace}
2410 \zcDicDefaultTransl{pairsep}{~\und\nobreakspace}
2411 \zcDicDefaultTransl{listsep}{,~}
2412 \zcDicDefaultTransl{lastsep}{~\und\nobreakspace}
2413 \zcDicDefaultTransl{tpairsep}{~\und\nobreakspace}
2414 \zcDicDefaultTransl{tlistsep}{,~}
2415 \zcDicDefaultTransl{tlastsep}{~\und\nobreakspace}
2416 \zcDicDefaultTransl{notesep}{~}
2417 \zcDicDefaultTransl{rangesep}{~bis\nobreakspace}
2418
2419 \zcDicTypeTransl{part}{Name-sg}{Teil}
2420 \zcDicTypeTransl{part}{name-sg}{Teil}
2421 \zcDicTypeTransl{part}{Name-pl}{Teile}
2422 \zcDicTypeTransl{part}{name-pl}{Teile}
2423
2424 \zcDicTypeTransl{chapter}{Name-sg}{Kapitel}
2425 \zcDicTypeTransl{chapter}{name-sg}{Kapitel}
2426 \zcDicTypeTransl{chapter}{Name-pl}{Kapitel}
2427 \zcDicTypeTransl{chapter}{name-pl}{Kapitel}
2428
2429 \zcDicTypeTransl{section}{Name-sg}{Abschnitt}
2430 \zcDicTypeTransl{section}{name-sg}{Abschnitt}
2431 \zcDicTypeTransl{section}{Name-pl}{Abschnitte}
2432 \zcDicTypeTransl{section}{name-pl}{Abschnitte}
2433
2434 \zcDicTypeTransl{paragraph}{Name-sg}{Absatz}
2435 \zcDicTypeTransl{paragraph}{name-sg}{Absatz}
2436 \zcDicTypeTransl{paragraph}{Name-pl}{Absätze}
2437 \zcDicTypeTransl{paragraph}{name-pl}{Absätze}
2438
2439 \zcDicTypeTransl{appendix}{Name-sg}{Anhang}
2440 \zcDicTypeTransl{appendix}{name-sg}{Anhang}
2441 \zcDicTypeTransl{appendix}{Name-pl}{Anhänge}
2442 \zcDicTypeTransl{appendix}{name-pl}{Anhänge}
2443
2444 \zcDicTypeTransl{page}{Name-sg}{Seite}
2445 \zcDicTypeTransl{page}{name-sg}{Seite}

```

2446 \zcDicTypeTransl{page}{Name-pl}{Seiten}
 2447 \zcDicTypeTransl{page}{name-pl}{Seiten}
 2448
 2449 \zcDicTypeTransl{line}{Name-sg}{Zeile}
 2450 \zcDicTypeTransl{line}{name-sg}{Zeile}
 2451 \zcDicTypeTransl{line}{Name-pl}{Zeilen}
 2452 \zcDicTypeTransl{line}{name-pl}{Zeilen}
 2453
 2454 \zcDicTypeTransl{figure}{Name-sg}{Abbildung}
 2455 \zcDicTypeTransl{figure}{name-sg}{Abbildung}
 2456 \zcDicTypeTransl{figure}{Name-pl}{Abbildungen}
 2457 \zcDicTypeTransl{figure}{name-pl}{Abbildungen}
 2458 \zcDicTypeTransl{figure}{Name-sg-ab}{Abb.}
 2459 \zcDicTypeTransl{figure}{name-sg-ab}{Abb.}
 2460 \zcDicTypeTransl{figure}{Name-pl-ab}{Abb.}
 2461 \zcDicTypeTransl{figure}{name-pl-ab}{Abb.}
 2462
 2463 \zcDicTypeTransl{table}{Name-sg}{Tabelle}
 2464 \zcDicTypeTransl{table}{name-sg}{Tabelle}
 2465 \zcDicTypeTransl{table}{Name-pl}{Tabellen}
 2466 \zcDicTypeTransl{table}{name-pl}{Tabellen}
 2467
 2468 \zcDicTypeTransl{item}{Name-sg}{Punkt}
 2469 \zcDicTypeTransl{item}{name-sg}{Punkt}
 2470 \zcDicTypeTransl{item}{Name-pl}{Punkte}
 2471 \zcDicTypeTransl{item}{name-pl}{Punkte}
 2472
 2473 \zcDicTypeTransl{footnote}{Name-sg}{Fußnote}
 2474 \zcDicTypeTransl{footnote}{name-sg}{Fußnote}
 2475 \zcDicTypeTransl{footnote}{Name-pl}{Fußnoten}
 2476 \zcDicTypeTransl{footnote}{name-pl}{Fußnoten}
 2477
 2478 \zcDicTypeTransl{note}{Name-sg}{Anmerkung}
 2479 \zcDicTypeTransl{note}{name-sg}{Anmerkung}
 2480 \zcDicTypeTransl{note}{Name-pl}{Anmerkungen}
 2481 \zcDicTypeTransl{note}{name-pl}{Anmerkungen}
 2482
 2483 \zcDicTypeTransl{equation}{Name-sg}{Gleichung}
 2484 \zcDicTypeTransl{equation}{name-sg}{Gleichung}
 2485 \zcDicTypeTransl{equation}{Name-pl}{Gleichungen}
 2486 \zcDicTypeTransl{equation}{name-pl}{Gleichungen}
 2487 \zcDicTypeTransl{equation}{refpre-in}{(}
 2488 \zcDicTypeTransl{equation}{refpos-in}{)}
 2489
 2490 \zcDicTypeTransl{theorem}{Name-sg}{Theorem}
 2491 \zcDicTypeTransl{theorem}{name-sg}{Theorem}
 2492 \zcDicTypeTransl{theorem}{Name-pl}{Theoreme}
 2493 \zcDicTypeTransl{theorem}{name-pl}{Theoreme}
 2494
 2495 \zcDicTypeTransl{lemma}{Name-sg}{Lemma}
 2496 \zcDicTypeTransl{lemma}{name-sg}{Lemma}
 2497 \zcDicTypeTransl{lemma}{Name-pl}{Lemmata}
 2498 \zcDicTypeTransl{lemma}{name-pl}{Lemmata}
 2499

```

2500 \zcDicTypeTransl{corollary}{Name-sg}{Korollar}
2501 \zcDicTypeTransl{corollary}{name-sg}{Korollar}
2502 \zcDicTypeTransl{corollary}{Name-pl}{Korollare}
2503 \zcDicTypeTransl{corollary}{name-pl}{Korollare}
2504
2505 \zcDicTypeTransl{proposition}{Name-sg}{Satz}
2506 \zcDicTypeTransl{proposition}{name-sg}{Satz}
2507 \zcDicTypeTransl{proposition}{Name-pl}{Sätze}
2508 \zcDicTypeTransl{proposition}{name-pl}{Sätze}
2509
2510 \zcDicTypeTransl{definition}{Name-sg}{Definition}
2511 \zcDicTypeTransl{definition}{name-sg}{Definition}
2512 \zcDicTypeTransl{definition}{Name-pl}{Definitionen}
2513 \zcDicTypeTransl{definition}{name-pl}{Definitionen}
2514
2515 \zcDicTypeTransl{proof}{Name-sg}{Beweis}
2516 \zcDicTypeTransl{proof}{name-sg}{Beweis}
2517 \zcDicTypeTransl{proof}{Name-pl}{Beweise}
2518 \zcDicTypeTransl{proof}{name-pl}{Beweise}
2519
2520 \zcDicTypeTransl{result}{Name-sg}{Ergebnis}
2521 \zcDicTypeTransl{result}{name-sg}{Ergebnis}
2522 \zcDicTypeTransl{result}{Name-pl}{Ergebnisse}
2523 \zcDicTypeTransl{result}{name-pl}{Ergebnisse}
2524
2525 \zcDicTypeTransl{example}{Name-sg}{Beispiel}
2526 \zcDicTypeTransl{example}{name-sg}{Beispiel}
2527 \zcDicTypeTransl{example}{Name-pl}{Beispiele}
2528 \zcDicTypeTransl{example}{name-pl}{Beispiele}
2529
2530 \zcDicTypeTransl{remark}{Name-sg}{Bemerkung}
2531 \zcDicTypeTransl{remark}{name-sg}{Bemerkung}
2532 \zcDicTypeTransl{remark}{Name-pl}{Bemerkungen}
2533 \zcDicTypeTransl{remark}{name-pl}{Bemerkungen}
2534
2535 \zcDicTypeTransl{algorithm}{Name-sg}{Algorithmus}
2536 \zcDicTypeTransl{algorithm}{name-sg}{Algorithmus}
2537 \zcDicTypeTransl{algorithm}{Name-pl}{Algorithmen}
2538 \zcDicTypeTransl{algorithm}{name-pl}{Algorithmen}
2539
2540 \zcDicTypeTransl{listing}{Name-sg}{Listing} % CHECK
2541 \zcDicTypeTransl{listing}{name-sg}{Listing} % CHECK
2542 \zcDicTypeTransl{listing}{Name-pl}{Listings} % CHECK
2543 \zcDicTypeTransl{listing}{name-pl}{Listings} % CHECK
2544
2545 \zcDicTypeTransl{exercise}{Name-sg}{Übungsaufgabe}
2546 \zcDicTypeTransl{exercise}{name-sg}{Übungsaufgabe}
2547 \zcDicTypeTransl{exercise}{Name-pl}{Übungsaufgaben}
2548 \zcDicTypeTransl{exercise}{name-pl}{Übungsaufgaben}
2549
2550 \zcDicTypeTransl{solution}{Name-sg}{Lösung}
2551 \zcDicTypeTransl{solution}{name-sg}{Lösung}
2552 \zcDicTypeTransl{solution}{Name-pl}{Lösungen}
2553 \zcDicTypeTransl{solution}{name-pl}{Lösungen}

```

2554 \langle /lang-german \rangle

10.4 French

2555 \langle *lang-french \rangle

2556 \backslash ProvideDictionaryFor{French}{zref-clever}

2557

2558 \backslash zcDicDefaultTransl{namesep}{\nobreakspace}

2559 \backslash zcDicDefaultTransl{pairsep}{~et\nobreakspace}

2560 \backslash zcDicDefaultTransl{listsep}{,~}

2561 \backslash zcDicDefaultTransl{lastsep}{~et\nobreakspace}

2562 \backslash zcDicDefaultTransl{tpairsep}{~et\nobreakspace}

2563 \backslash zcDicDefaultTransl{tlistsep}{,~}

2564 \backslash zcDicDefaultTransl{tlastsep}{~et\nobreakspace}

2565 \backslash zcDicDefaultTransl{notesep}{~}

2566 \backslash zcDicDefaultTransl{rangesep}{~à\nobreakspace}

2567

2568 \backslash zcDicTypeTransl{part}{Name-sg}{Partie}

2569 \backslash zcDicTypeTransl{part}{name-sg}{partie}

2570 \backslash zcDicTypeTransl{part}{Name-pl}{Parties}

2571 \backslash zcDicTypeTransl{part}{name-pl}{parties}

2572

2573 \backslash zcDicTypeTransl{chapter}{Name-sg}{Chapitre}

2574 \backslash zcDicTypeTransl{chapter}{name-sg}{chapitre}

2575 \backslash zcDicTypeTransl{chapter}{Name-pl}{Chapitres}

2576 \backslash zcDicTypeTransl{chapter}{name-pl}{chapitres}

2577

2578 \backslash zcDicTypeTransl{section}{Name-sg}{Section}

2579 \backslash zcDicTypeTransl{section}{name-sg}{section}

2580 \backslash zcDicTypeTransl{section}{Name-pl}{Sections}

2581 \backslash zcDicTypeTransl{section}{name-pl}{sections}

2582

2583 \backslash zcDicTypeTransl{paragraph}{Name-sg}{Paragraphe}

2584 \backslash zcDicTypeTransl{paragraph}{name-sg}{paragraphe}

2585 \backslash zcDicTypeTransl{paragraph}{Name-pl}{Paragraphes}

2586 \backslash zcDicTypeTransl{paragraph}{name-pl}{paragraphes}

2587

2588 \backslash zcDicTypeTransl{appendix}{Name-sg}{Annexe}

2589 \backslash zcDicTypeTransl{appendix}{name-sg}{annexe}

2590 \backslash zcDicTypeTransl{appendix}{Name-pl}{Annexes}

2591 \backslash zcDicTypeTransl{appendix}{name-pl}{annexes}

2592

2593 \backslash zcDicTypeTransl{page}{Name-sg}{Page}

2594 \backslash zcDicTypeTransl{page}{name-sg}{page}

2595 \backslash zcDicTypeTransl{page}{Name-pl}{Pages}

2596 \backslash zcDicTypeTransl{page}{name-pl}{pages}

2597

2598 \backslash zcDicTypeTransl{line}{Name-sg}{Ligne}

2599 \backslash zcDicTypeTransl{line}{name-sg}{ligne}

2600 \backslash zcDicTypeTransl{line}{Name-pl}{Lignes}

2601 \backslash zcDicTypeTransl{line}{name-pl}{lignes}

2602

2603 \backslash zcDicTypeTransl{figure}{Name-sg}{Figure}

2604 \backslash zcDicTypeTransl{figure}{name-sg}{figure}

2605 \backslash zcDicTypeTransl{figure}{Name-pl}{Figures}

2606 \zcDicTypeTransl{figure}{name-pl}{figures}
 2607
 2608 \zcDicTypeTransl{table}{Name-sg}{Table}
 2609 \zcDicTypeTransl{table}{name-sg}{table}
 2610 \zcDicTypeTransl{table}{Name-pl}{Tables}
 2611 \zcDicTypeTransl{table}{name-pl}{tables}
 2612
 2613 \zcDicTypeTransl{item}{Name-sg}{Point}
 2614 \zcDicTypeTransl{item}{name-sg}{point}
 2615 \zcDicTypeTransl{item}{Name-pl}{Points}
 2616 \zcDicTypeTransl{item}{name-pl}{points}
 2617
 2618 \zcDicTypeTransl{footnote}{Name-sg}{Note}
 2619 \zcDicTypeTransl{footnote}{name-sg}{note}
 2620 \zcDicTypeTransl{footnote}{Name-pl}{Notes}
 2621 \zcDicTypeTransl{footnote}{name-pl}{notes}
 2622
 2623 \zcDicTypeTransl{note}{Name-sg}{Note}
 2624 \zcDicTypeTransl{note}{name-sg}{note}
 2625 \zcDicTypeTransl{note}{Name-pl}{Notes}
 2626 \zcDicTypeTransl{note}{name-pl}{notes}
 2627
 2628 \zcDicTypeTransl{equation}{Name-sg}{Équation}
 2629 \zcDicTypeTransl{equation}{name-sg}{équation}
 2630 \zcDicTypeTransl{equation}{Name-pl}{Équations}
 2631 \zcDicTypeTransl{equation}{name-pl}{équations}
 2632 \zcDicTypeTransl{equation}{refpre-in}{(}
 2633 \zcDicTypeTransl{equation}{refpos-in}{)}
 2634
 2635 \zcDicTypeTransl{theorem}{Name-sg}{Théorème}
 2636 \zcDicTypeTransl{theorem}{name-sg}{théorème}
 2637 \zcDicTypeTransl{theorem}{Name-pl}{Théorèmes}
 2638 \zcDicTypeTransl{theorem}{name-pl}{théorèmes}
 2639
 2640 \zcDicTypeTransl{lemma}{Name-sg}{Lemme}
 2641 \zcDicTypeTransl{lemma}{name-sg}{lemme}
 2642 \zcDicTypeTransl{lemma}{Name-pl}{Lemmes}
 2643 \zcDicTypeTransl{lemma}{name-pl}{lemmes}
 2644
 2645 \zcDicTypeTransl{corollary}{Name-sg}{Corollaire}
 2646 \zcDicTypeTransl{corollary}{name-sg}{corollaire}
 2647 \zcDicTypeTransl{corollary}{Name-pl}{Corollaires}
 2648 \zcDicTypeTransl{corollary}{name-pl}{corollaires}
 2649
 2650 \zcDicTypeTransl{proposition}{Name-sg}{Proposition}
 2651 \zcDicTypeTransl{proposition}{name-sg}{proposition}
 2652 \zcDicTypeTransl{proposition}{Name-pl}{Propositions}
 2653 \zcDicTypeTransl{proposition}{name-pl}{propositions}
 2654
 2655 \zcDicTypeTransl{definition}{Name-sg}{Définition}
 2656 \zcDicTypeTransl{definition}{name-sg}{définition}
 2657 \zcDicTypeTransl{definition}{Name-pl}{Définitions}
 2658 \zcDicTypeTransl{definition}{name-pl}{définitions}
 2659

```

2660 \zcDicTypeTransl{proof}{Name-sg}{Démonstration}
2661 \zcDicTypeTransl{proof}{name-sg}{démonstration}
2662 \zcDicTypeTransl{proof}{Name-pl}{Démonstrations}
2663 \zcDicTypeTransl{proof}{name-pl}{démonstrations}
2664
2665 \zcDicTypeTransl{result}{Name-sg}{Résultat}
2666 \zcDicTypeTransl{result}{name-sg}{résultat}
2667 \zcDicTypeTransl{result}{Name-pl}{Résultats}
2668 \zcDicTypeTransl{result}{name-pl}{résultats}
2669
2670 \zcDicTypeTransl{example}{Name-sg}{Exemple}
2671 \zcDicTypeTransl{example}{name-sg}{exemple}
2672 \zcDicTypeTransl{example}{Name-pl}{Exemples}
2673 \zcDicTypeTransl{example}{name-pl}{exemples}
2674
2675 \zcDicTypeTransl{remark}{Name-sg}{Remarque}
2676 \zcDicTypeTransl{remark}{name-sg}{remarque}
2677 \zcDicTypeTransl{remark}{Name-pl}{Remarques}
2678 \zcDicTypeTransl{remark}{name-pl}{remarques}
2679
2680 \zcDicTypeTransl{algorithm}{Name-sg}{Algorithme}
2681 \zcDicTypeTransl{algorithm}{name-sg}{algorithme}
2682 \zcDicTypeTransl{algorithm}{Name-pl}{Algorithmes}
2683 \zcDicTypeTransl{algorithm}{name-pl}{algorithmes}
2684
2685 \zcDicTypeTransl{listing}{Name-sg}{Liste}
2686 \zcDicTypeTransl{listing}{name-sg}{liste}
2687 \zcDicTypeTransl{listing}{Name-pl}{Listes}
2688 \zcDicTypeTransl{listing}{name-pl}{listes}
2689
2690 \zcDicTypeTransl{exercise}{Name-sg}{Exercice}
2691 \zcDicTypeTransl{exercise}{name-sg}{exercice}
2692 \zcDicTypeTransl{exercise}{Name-pl}{Exercices}
2693 \zcDicTypeTransl{exercise}{name-pl}{exercices}
2694
2695 \zcDicTypeTransl{solution}{Name-sg}{Solution}
2696 \zcDicTypeTransl{solution}{name-sg}{solution}
2697 \zcDicTypeTransl{solution}{Name-pl}{Solutions}
2698 \zcDicTypeTransl{solution}{name-pl}{solutions}
2699 </lang-french>

```

10.5 Portuguese

```

2700 <*lang-portuguese>
2701 \ProvideDictionaryFor{Portuguese}{zref-clever}
2702
2703 \zcDicDefaultTransl{namesep}{\nobreakspace}
2704 \zcDicDefaultTransl{pairsep}{~e\nobreakspace}
2705 \zcDicDefaultTransl{listsep}{~,~}
2706 \zcDicDefaultTransl{lastsep}{~e\nobreakspace}
2707 \zcDicDefaultTransl{tpairsep}{~e\nobreakspace}
2708 \zcDicDefaultTransl{tlistsep}{~,~}
2709 \zcDicDefaultTransl{tlastsep}{~e\nobreakspace}
2710 \zcDicDefaultTransl{notesep}{~}

```

2711 \zcDicDefaultTransl{rangeseq}{~a\nobreakspace}
2712
2713 \zcDicTypeTransl{part}{Name-sg}{Parte}
2714 \zcDicTypeTransl{part}{name-sg}{parte}
2715 \zcDicTypeTransl{part}{Name-pl}{Partes}
2716 \zcDicTypeTransl{part}{name-pl}{partes}
2717
2718 \zcDicTypeTransl{chapter}{Name-sg}{Capítulo}
2719 \zcDicTypeTransl{chapter}{name-sg}{capítulo}
2720 \zcDicTypeTransl{chapter}{Name-pl}{Capítulos}
2721 \zcDicTypeTransl{chapter}{name-pl}{capítulos}
2722
2723 \zcDicTypeTransl{section}{Name-sg}{Seção}
2724 \zcDicTypeTransl{section}{name-sg}{seção}
2725 \zcDicTypeTransl{section}{Name-pl}{Seções}
2726 \zcDicTypeTransl{section}{name-pl}{seções}
2727
2728 \zcDicTypeTransl{paragraph}{Name-sg}{Parágrafo}
2729 \zcDicTypeTransl{paragraph}{name-sg}{parágrafo}
2730 \zcDicTypeTransl{paragraph}{Name-pl}{Parágrafos}
2731 \zcDicTypeTransl{paragraph}{name-pl}{parágrafos}
2732 \zcDicTypeTransl{paragraph}{Name-sg-ab}{Par.}
2733 \zcDicTypeTransl{paragraph}{name-sg-ab}{par.}
2734 \zcDicTypeTransl{paragraph}{Name-pl-ab}{Par.}
2735 \zcDicTypeTransl{paragraph}{name-pl-ab}{par.}
2736
2737 \zcDicTypeTransl{appendix}{Name-sg}{Apêndice}
2738 \zcDicTypeTransl{appendix}{name-sg}{apêndice}
2739 \zcDicTypeTransl{appendix}{Name-pl}{Apêndices}
2740 \zcDicTypeTransl{appendix}{name-pl}{apêndices}
2741
2742 \zcDicTypeTransl{page}{Name-sg}{Página}
2743 \zcDicTypeTransl{page}{name-sg}{página}
2744 \zcDicTypeTransl{page}{Name-pl}{Páginas}
2745 \zcDicTypeTransl{page}{name-pl}{páginas}
2746 \zcDicTypeTransl{page}{name-sg-ab}{p.}
2747 \zcDicTypeTransl{page}{name-pl-ab}{pp.}
2748
2749 \zcDicTypeTransl{line}{Name-sg}{Linha}
2750 \zcDicTypeTransl{line}{name-sg}{linha}
2751 \zcDicTypeTransl{line}{Name-pl}{Linhas}
2752 \zcDicTypeTransl{line}{name-pl}{linhas}
2753
2754 \zcDicTypeTransl{figure}{Name-sg}{Figura}
2755 \zcDicTypeTransl{figure}{name-sg}{figura}
2756 \zcDicTypeTransl{figure}{Name-pl}{Figuras}
2757 \zcDicTypeTransl{figure}{name-pl}{figuras}
2758 \zcDicTypeTransl{figure}{Name-sg-ab}{Fig.}
2759 \zcDicTypeTransl{figure}{name-sg-ab}{fig.}
2760 \zcDicTypeTransl{figure}{Name-pl-ab}{Figs.}
2761 \zcDicTypeTransl{figure}{name-pl-ab}{figs.}
2762
2763 \zcDicTypeTransl{table}{Name-sg}{Tabela}
2764 \zcDicTypeTransl{table}{name-sg}{tabela}

2765 \zcDicTypeTransl{table}{Name-pl}{Tabelas}
 2766 \zcDicTypeTransl{table}{name-pl}{tabelas}
 2767
 2768 \zcDicTypeTransl{item}{Name-sg}{Item}
 2769 \zcDicTypeTransl{item}{name-sg}{item}
 2770 \zcDicTypeTransl{item}{Name-pl}{Itens}
 2771 \zcDicTypeTransl{item}{name-pl}{itens}
 2772
 2773 \zcDicTypeTransl{footnote}{Name-sg}{Nota}
 2774 \zcDicTypeTransl{footnote}{name-sg}{nota}
 2775 \zcDicTypeTransl{footnote}{Name-pl}{Notas}
 2776 \zcDicTypeTransl{footnote}{name-pl}{notas}
 2777
 2778 \zcDicTypeTransl{note}{Name-sg}{Nota}
 2779 \zcDicTypeTransl{note}{name-sg}{nota}
 2780 \zcDicTypeTransl{note}{Name-pl}{Notas}
 2781 \zcDicTypeTransl{note}{name-pl}{notas}
 2782
 2783 \zcDicTypeTransl{equation}{Name-sg}{Equação}
 2784 \zcDicTypeTransl{equation}{name-sg}{equação}
 2785 \zcDicTypeTransl{equation}{Name-pl}{Equações}
 2786 \zcDicTypeTransl{equation}{name-pl}{equações}
 2787 \zcDicTypeTransl{equation}{Name-sg-ab}{Eq.}
 2788 \zcDicTypeTransl{equation}{name-sg-ab}{eq.}
 2789 \zcDicTypeTransl{equation}{Name-pl-ab}{Eqs.}
 2790 \zcDicTypeTransl{equation}{name-pl-ab}{eqs.}
 2791 \zcDicTypeTransl{equation}{refpre-in}{({}
 2792 \zcDicTypeTransl{equation}{refpos-in}{)})
 2793
 2794 \zcDicTypeTransl{theorem}{Name-sg}{Teorema}
 2795 \zcDicTypeTransl{theorem}{name-sg}{teorema}
 2796 \zcDicTypeTransl{theorem}{Name-pl}{Teoremas}
 2797 \zcDicTypeTransl{theorem}{name-pl}{teoremas}
 2798
 2799 \zcDicTypeTransl{lemma}{Name-sg}{Lema}
 2800 \zcDicTypeTransl{lemma}{name-sg}{lema}
 2801 \zcDicTypeTransl{lemma}{Name-pl}{Lemas}
 2802 \zcDicTypeTransl{lemma}{name-pl}{lemas}
 2803
 2804 \zcDicTypeTransl{corollary}{Name-sg}{Corolário}
 2805 \zcDicTypeTransl{corollary}{name-sg}{corolário}
 2806 \zcDicTypeTransl{corollary}{Name-pl}{Corolários}
 2807 \zcDicTypeTransl{corollary}{name-pl}{corolários}
 2808
 2809 \zcDicTypeTransl{proposition}{Name-sg}{Proposição}
 2810 \zcDicTypeTransl{proposition}{name-sg}{proposição}
 2811 \zcDicTypeTransl{proposition}{Name-pl}{Proposições}
 2812 \zcDicTypeTransl{proposition}{name-pl}{proposições}
 2813
 2814 \zcDicTypeTransl{definition}{Name-sg}{Definição}
 2815 \zcDicTypeTransl{definition}{name-sg}{definição}
 2816 \zcDicTypeTransl{definition}{Name-pl}{Definições}
 2817 \zcDicTypeTransl{definition}{name-pl}{definições}
 2818

```

2819 \zcDicTypeTransl{proof}{Name-sg}{Demonstração}
2820 \zcDicTypeTransl{proof}{name-sg}{demonstração}
2821 \zcDicTypeTransl{proof}{Name-pl}{Demonstrações}
2822 \zcDicTypeTransl{proof}{name-pl}{demonstrações}
2823
2824 \zcDicTypeTransl{result}{Name-sg}{Resultado}
2825 \zcDicTypeTransl{result}{name-sg}{resultado}
2826 \zcDicTypeTransl{result}{Name-pl}{Resultados}
2827 \zcDicTypeTransl{result}{name-pl}{resultados}
2828
2829 \zcDicTypeTransl{example}{Name-sg}{Exemplo}
2830 \zcDicTypeTransl{example}{name-sg}{exemplo}
2831 \zcDicTypeTransl{example}{Name-pl}{Exemplos}
2832 \zcDicTypeTransl{example}{name-pl}{exemplos}
2833
2834 \zcDicTypeTransl{remark}{Name-sg}{Observação}
2835 \zcDicTypeTransl{remark}{name-sg}{observação}
2836 \zcDicTypeTransl{remark}{Name-pl}{Observações}
2837 \zcDicTypeTransl{remark}{name-pl}{observações}
2838
2839 \zcDicTypeTransl{algorithm}{Name-sg}{Algoritmo}
2840 \zcDicTypeTransl{algorithm}{name-sg}{algoritmo}
2841 \zcDicTypeTransl{algorithm}{Name-pl}{Algoritmos}
2842 \zcDicTypeTransl{algorithm}{name-pl}{algoritmos}
2843
2844 \zcDicTypeTransl{listing}{Name-sg}{Listagem}
2845 \zcDicTypeTransl{listing}{name-sg}{listagem}
2846 \zcDicTypeTransl{listing}{Name-pl}{Listagens}
2847 \zcDicTypeTransl{listing}{name-pl}{listagens}
2848
2849 \zcDicTypeTransl{exercise}{Name-sg}{Exercício}
2850 \zcDicTypeTransl{exercise}{name-sg}{exercício}
2851 \zcDicTypeTransl{exercise}{Name-pl}{Exercícios}
2852 \zcDicTypeTransl{exercise}{name-pl}{exercícios}
2853
2854 \zcDicTypeTransl{solution}{Name-sg}{Solução}
2855 \zcDicTypeTransl{solution}{name-sg}{solução}
2856 \zcDicTypeTransl{solution}{Name-pl}{Soluções}
2857 \zcDicTypeTransl{solution}{name-pl}{soluções}
2858 </lang-portuguese>

```

10.6 Spanish

```

2859 <*lang-spanish>
2860 \ProvideDictionaryFor{Spanish}{zref-clever}
2861
2862 \zcDicDefaultTransl{namesep}{\nobreakspace}
2863 \zcDicDefaultTransl{pairsep}{~y\nobreakspace}
2864 \zcDicDefaultTransl{listsep}{~,~}
2865 \zcDicDefaultTransl{lastsep}{~y\nobreakspace}
2866 \zcDicDefaultTransl{tpairsep}{~y\nobreakspace}
2867 \zcDicDefaultTransl{tlistsep}{~,~}
2868 \zcDicDefaultTransl{tlastsep}{~y\nobreakspace}
2869 \zcDicDefaultTransl{notesep}{~}

```

2870 \zcDicDefaultTransl{rangesepl}{~a\nobreakspace}
 2871
 2872 \zcDicTypeTransl{part}{Name-sg}{Parte}
 2873 \zcDicTypeTransl{part}{name-sg}{parte}
 2874 \zcDicTypeTransl{part}{Name-pl}{Partes}
 2875 \zcDicTypeTransl{part}{name-pl}{partes}
 2876
 2877 \zcDicTypeTransl{chapter}{Name-sg}{Capítulo}
 2878 \zcDicTypeTransl{chapter}{name-sg}{capítulo}
 2879 \zcDicTypeTransl{chapter}{Name-pl}{Capítulos}
 2880 \zcDicTypeTransl{chapter}{name-pl}{capítulos}
 2881
 2882 \zcDicTypeTransl{section}{Name-sg}{Sección}
 2883 \zcDicTypeTransl{section}{name-sg}{sección}
 2884 \zcDicTypeTransl{section}{Name-pl}{Secciones}
 2885 \zcDicTypeTransl{section}{name-pl}{secciones}
 2886
 2887 \zcDicTypeTransl{paragraph}{Name-sg}{Párrafo}
 2888 \zcDicTypeTransl{paragraph}{name-sg}{párrafo}
 2889 \zcDicTypeTransl{paragraph}{Name-pl}{Párrafos}
 2890 \zcDicTypeTransl{paragraph}{name-pl}{párrafos}
 2891
 2892 \zcDicTypeTransl{appendix}{Name-sg}{Apéndice}
 2893 \zcDicTypeTransl{appendix}{name-sg}{apéndice}
 2894 \zcDicTypeTransl{appendix}{Name-pl}{Apéndices}
 2895 \zcDicTypeTransl{appendix}{name-pl}{apéndices}
 2896
 2897 \zcDicTypeTransl{page}{Name-sg}{Página}
 2898 \zcDicTypeTransl{page}{name-sg}{página}
 2899 \zcDicTypeTransl{page}{Name-pl}{Páginas}
 2900 \zcDicTypeTransl{page}{name-pl}{páginas}
 2901
 2902 \zcDicTypeTransl{line}{Name-sg}{Línea}
 2903 \zcDicTypeTransl{line}{name-sg}{línea}
 2904 \zcDicTypeTransl{line}{Name-pl}{Líneas}
 2905 \zcDicTypeTransl{line}{name-pl}{líneas}
 2906
 2907 \zcDicTypeTransl{figure}{Name-sg}{Figura}
 2908 \zcDicTypeTransl{figure}{name-sg}{figura}
 2909 \zcDicTypeTransl{figure}{Name-pl}{Figuras}
 2910 \zcDicTypeTransl{figure}{name-pl}{figuras}
 2911
 2912 \zcDicTypeTransl{table}{Name-sg}{Cuadro}
 2913 \zcDicTypeTransl{table}{name-sg}{cuadro}
 2914 \zcDicTypeTransl{table}{Name-pl}{Cuadros}
 2915 \zcDicTypeTransl{table}{name-pl}{cuadros}
 2916
 2917 \zcDicTypeTransl{item}{Name-sg}{Punto}
 2918 \zcDicTypeTransl{item}{name-sg}{punto}
 2919 \zcDicTypeTransl{item}{Name-pl}{Puntos}
 2920 \zcDicTypeTransl{item}{name-pl}{puntos}
 2921
 2922 \zcDicTypeTransl{footnote}{Name-sg}{Nota}
 2923 \zcDicTypeTransl{footnote}{name-sg}{nota}

2924 \zcDicTypeTransl{footnote}{Name-pl}{Notas}
 2925 \zcDicTypeTransl{footnote}{name-pl}{notas}
 2926
 2927 \zcDicTypeTransl{note}{Name-sg}{Nota}
 2928 \zcDicTypeTransl{note}{name-sg}{nota}
 2929 \zcDicTypeTransl{note}{Name-pl}{Notas}
 2930 \zcDicTypeTransl{note}{name-pl}{notas}
 2931
 2932 \zcDicTypeTransl{equation}{Name-sg}{Ecuación}
 2933 \zcDicTypeTransl{equation}{name-sg}{ecuación}
 2934 \zcDicTypeTransl{equation}{Name-pl}{Ecuaciones}
 2935 \zcDicTypeTransl{equation}{name-pl}{ecuaciones}
 2936 \zcDicTypeTransl{equation}{refpre-in}{(}
 2937 \zcDicTypeTransl{equation}{refpos-in}{)}
 2938
 2939 \zcDicTypeTransl{theorem}{Name-sg}{Teorema}
 2940 \zcDicTypeTransl{theorem}{name-sg}{teorema}
 2941 \zcDicTypeTransl{theorem}{Name-pl}{Teoremas}
 2942 \zcDicTypeTransl{theorem}{name-pl}{teoremas}
 2943
 2944 \zcDicTypeTransl{lemma}{Name-sg}{Lema}
 2945 \zcDicTypeTransl{lemma}{name-sg}{lema}
 2946 \zcDicTypeTransl{lemma}{Name-pl}{Lemas}
 2947 \zcDicTypeTransl{lemma}{name-pl}{lemas}
 2948
 2949 \zcDicTypeTransl{corollary}{Name-sg}{Corolario}
 2950 \zcDicTypeTransl{corollary}{name-sg}{corolario}
 2951 \zcDicTypeTransl{corollary}{Name-pl}{Corolarios}
 2952 \zcDicTypeTransl{corollary}{name-pl}{corolarios}
 2953
 2954 \zcDicTypeTransl{proposition}{Name-sg}{Proposición}
 2955 \zcDicTypeTransl{proposition}{name-sg}{proposición}
 2956 \zcDicTypeTransl{proposition}{Name-pl}{Proposiciones}
 2957 \zcDicTypeTransl{proposition}{name-pl}{proposiciones}
 2958
 2959 \zcDicTypeTransl{definition}{Name-sg}{Definición}
 2960 \zcDicTypeTransl{definition}{name-sg}{definición}
 2961 \zcDicTypeTransl{definition}{Name-pl}{Definiciones}
 2962 \zcDicTypeTransl{definition}{name-pl}{definiciones}
 2963
 2964 \zcDicTypeTransl{proof}{Name-sg}{Demostración}
 2965 \zcDicTypeTransl{proof}{name-sg}{demostración}
 2966 \zcDicTypeTransl{proof}{Name-pl}{Demostraciones}
 2967 \zcDicTypeTransl{proof}{name-pl}{demostraciones}
 2968
 2969 \zcDicTypeTransl{result}{Name-sg}{Resultado}
 2970 \zcDicTypeTransl{result}{name-sg}{resultado}
 2971 \zcDicTypeTransl{result}{Name-pl}{Resultados}
 2972 \zcDicTypeTransl{result}{name-pl}{resultados}
 2973
 2974 \zcDicTypeTransl{example}{Name-sg}{Ejemplo}
 2975 \zcDicTypeTransl{example}{name-sg}{ejemplo}
 2976 \zcDicTypeTransl{example}{Name-pl}{Ejemplos}
 2977 \zcDicTypeTransl{example}{name-pl}{ejemplos}

```

2978
2979 \zcDicTypeTransl{remark}{Name-sg}{Observación}
2980 \zcDicTypeTransl{remark}{name-sg}{observación}
2981 \zcDicTypeTransl{remark}{Name-pl}{Observaciones}
2982 \zcDicTypeTransl{remark}{name-pl}{observaciones}
2983
2984 \zcDicTypeTransl{algorithm}{Name-sg}{Algoritmo}
2985 \zcDicTypeTransl{algorithm}{name-sg}{algoritmo}
2986 \zcDicTypeTransl{algorithm}{Name-pl}{Algoritmos}
2987 \zcDicTypeTransl{algorithm}{name-pl}{algoritmos}
2988
2989 \zcDicTypeTransl{listing}{Name-sg}{Listado}
2990 \zcDicTypeTransl{listing}{name-sg}{listado}
2991 \zcDicTypeTransl{listing}{Name-pl}{Listados}
2992 \zcDicTypeTransl{listing}{name-pl}{listados}
2993
2994 \zcDicTypeTransl{exercise}{Name-sg}{Ejercicio}
2995 \zcDicTypeTransl{exercise}{name-sg}{ejercicio}
2996 \zcDicTypeTransl{exercise}{Name-pl}{Ejercicios}
2997 \zcDicTypeTransl{exercise}{name-pl}{ejercicios}
2998
2999 \zcDicTypeTransl{solution}{Name-sg}{Solución}
3000 \zcDicTypeTransl{solution}{name-sg}{solución}
3001 \zcDicTypeTransl{solution}{Name-pl}{Soluciones}
3002 \zcDicTypeTransl{solution}{name-pl}{soluciones}
3003 </lang-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.

A	
\AddToHook 91, 255, 270, 380, 453, 487, 513, 573
\appendix 57
\appendixname 57
\Arg 2110
\AtEndOfPackage 16, 485
B	
\baselanguage 476, 528, 545
bool commands:	
\bool_case_true: 2
\bool_if:NTF 384, 388, 1295, 1368, 1493, 1514, 1545, 1600, 1667, 1671, 1678, 1687, 1693, 1974
\bool_if:nTF	... 59, 934, 943, 952, 1023, 1051, 1074, 1163, 1171, 1309, 1317, 1526, 1533, 1540, 1789, 2052
\bool_lazy_all:nTF 2117
\bool_lazy_and:nnTF 827, 838, 1642, 1851, 2162
\bool_lazy_any:nTF 1942, 1951
\bool_lazy_or:nnTF 831, 1839
\bool_new:N 291, 292, 317, 341, 350, 357, 358, 413, 414, 431, 432, 566, 567, 849, 866, 1203, 1204, 1215, 1216, 1217, 1236
\bool_set:Nn 826
\bool_set_false:N 304, 308, 365, 374, 375, 390, 588, 1012, 1262, 1301, 1315, 1326, 1505, 1640, 1641, 1949, 1966
\bool_set_true:N 298, 299, 303, 309, 364, 369, 370, 577, 582, 1035, 1046, 1063, 1069, 1086, 1092, 1118, 1130, 1269, 1296, 1302, 1306, 1327, 1330, 1965, 2188, 2195, 2196, 2214, 2221, 2222
\bool_until_do:Nn 1016, 1263

C		<code>\fmtversion</code> 3
clist commands:		G
<code>\clist_map_inline:Nn</code> 521, 538		group commands:
<code>\clist_map_inline:nn</code>		<code>\group_begin:</code>
.... 200, 597, 646, 663, 731, 757, 789		.. 93, 823, 1547, 1573, 1792, 1795,
<code>\counterwithin</code> 4		1815, 1818, 1998, 2003, 2006, 2021,
<code>\cs</code> 1142, 1491, 2113, 2147, 2158		2028, 2043, 2058, 2061, 2092, 2095
cs commands:		<code>\group_end:</code>
<code>\cs_generate_variant:Nn</code> 96, 846, 1570, 1576, 1810, 1812,
..... 55, 56, 149, 152, 867, 1830		1823, 1825, 2001, 2012, 2019, 2025,
<code>\cs_if_exist:NTF</code> 39, 48, 69		2031, 2046, 2087, 2089, 2101, 2103
<code>\cs_new:Npn</code> 37, 46, 57, 67, 78, 1785, 1969		H
<code>\cs_new_protected:Npn</code>		<code>\hyperlink</code> 48
.... 147, 150, 153, 161, 821, 868,		I
884, 927, 993, 1138, 1194, 1239,		<code>\IfBooleanTF</code> 852
1379, 1635, 1831, 2111, 2156, 2176		<code>\IfFormatAtLeastTF</code> 3, 4
<code>\cs_new_protected:Npx</code> 90		<code>\IfTranslation</code> 142
<code>\cs_set_eq:NN</code> 94		int commands:
D		<code>\int_case:nnTF</code>
<code>\declaretranslation</code> 151		.. 1382, 1408, 1439, 1603, 1700, 1736
<code>\declaretranslationfallback</code> 154		<code>\int_compare:nNnTF</code> 981,
E		1036, 1103, 1119, 1149, 1151, 1196,
<code>\endinput</code> 12		1350, 1395, 1429, 1592, 1594, 1657,
exp commands:		1681, 1723, 2184, 2190, 2210, 2216
<code>\exp_args:NNe</code> 27		<code>\int_compare_p:nNn</code>
<code>\exp_args:NNx</code> 95, 1060, 1083	 1165, 1173, 1843, 1854, 1962
<code>\exp_args:Nx</code> 471, 475, 523, 527, 540, 544		<code>\int_eval:n</code> 90
<code>\exp_args:Nxx</code>		<code>\int_incr:N</code> 1630, 1670,
..... 977, 1031, 2180, 2202, 2206		1672, 1686, 1688, 1692, 1694, 1783
<code>\exp_not:N</code> 103,		<code>\int_new:N</code>
109, 1547, 1550, 1570, 1573, 1576,	 864, 865, 1210, 1211, 1212, 1213
1792, 1795, 1798, 1810, 1812, 1815,		<code>\int_set:Nn</code> ... 1150, 1152, 1156, 1159
1818, 1823, 1825, 1828, 1972, 1980,		<code>\int_use:N</code> 33, 35, 50
1998, 2001, 2003, 2006, 2012, 2019,		<code>\int_zero:N</code>
2021, 2025, 2028, 2031, 2033, 2039,	 1140, 1141, 1247, 1248, 1249,
2043, 2046, 2058, 2061, 2064, 2087,		1250, 1629, 1631, 1632, 1778, 1779
2089, 2092, 2095, 2101, 2103, 2106		iow commands:
<code>\exp_not:n</code> . 1399, 1415, 1427, 1431,		<code>\iow_newline:</code> 120, 124
1451, 1464, 1467, 1479, 1482, 1515,		K
1516, 1548, 1569, 1574, 1575, 1707,		keys commands:
1720, 1725, 1745, 1756, 1759, 1769,		<code>\keys_define:nn</code> 22, 168, 196,
1772, 1793, 1794, 1796, 1806, 1808,		222, 246, 274, 281, 293, 318, 327,
1811, 1816, 1817, 1819, 1820, 1822,		342, 351, 359, 392, 399, 415, 433,
1824, 1999, 2000, 2002, 2004, 2005,		449, 489, 558, 561, 568, 578, 589,
2007, 2008, 2011, 2023, 2024, 2029,		617, 628, 654, 687, 718, 739, 770, 801
2030, 2032, 2040, 2044, 2045, 2047,		<code>\keys_set:nn</code> 22, 583, 634, 644, 716, 824
2059, 2060, 2062, 2082, 2085, 2088,		keyval commands:
2093, 2094, 2096, 2097, 2100, 2102		<code>\keyval_parse:nnn</code> 172, 226
F		L
file commands:		<code>\labelformat</code> 3
<code>\file_if_exist:nTF</code>		<code>\language</code> 16
..... 471, 475, 523, 527, 540, 544		

<code>\LoadDictionaryFor</code>	16, 17, 473, 477, 525, 529, 542, 546, 552
M	
<code>\MessageBreak</code>	10
msg commands:	
<code>\msg_line_context:</code>	102, 108, 115, 129, 133, 135, 137, 139
<code>\msg_new:nnn</code>	100, 106, 111, 113, 118, 123, 125, 127, 132, 134, 136, 138
<code>\msg_warning:nn</code>	260, 285, 389, 395, 571, 592
<code>\msg_warning:nnn</code>	228, 658, 751, 808, 1343, 1500, 1886, 1933
<code>\msg_warning:nnnn</code>	174, 1044, 1067, 1090, 1128
N	
<code>\newcounter</code>	4
<code>\NewDocumentCommand</code>	155, 157, 633, 639, 712, 819, 850
<code>\NewHook</code>	448
<code>\newtheorem</code>	57
<code>\nobreakspace</code>	2229, 2246, 2247, 2249, 2250, 2252, 2254, 2409, 2410, 2412, 2413, 2415, 2417, 2558, 2559, 2561, 2562, 2564, 2566, 2703, 2704, 2706, 2707, 2709, 2711, 2862, 2863, 2865, 2866, 2868, 2870
<code>\noexpand</code>	121
P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	6
prg commands:	
<code>\prg_generate_conditional_-variant:Nnn</code>	146
<code>\prg_new_conditional:Npnn</code>	140
<code>\prg_return_false:</code>	144
<code>\prg_return_true:</code>	143
<code>\ProcessKeysOptions</code>	16, 636
prop commands:	
<code>\prop_get:NnN</code>	2130
<code>\prop_get:NnNTF</code>	1865, 1892, 1897, 2114, 2159, 2169
<code>\prop_if_exist:NTF</code>	641, 725
<code>\prop_if_exist_p:N</code>	2121, 2165
<code>\prop_if_in:NnTF</code>	25
<code>\prop_if_in_p:Nn</code>	60, 2125
<code>\prop_item:Nn</code>	27, 61
<code>\prop_new:N</code>	167, 221, 596, 642, 726
<code>\prop_put:Nnn</code>	165, 624, 702
<code>\prop_remove:Nn</code>	164, 623, 694
<code>\providecommand</code>	3
<code>\ProvideDictionaryFor</code>	9, 2244, 2407, 2556, 2701, 2860
<code>\ProvideDictTranslation</code>	9, 156, 158
<code>\ProvidesExplPackage</code>	14
R	
<code>\refstepcounter</code>	3
<code>\RequirePackage</code>	16, 17, 18, 19, 20, 385, 635
S	
<code>\SaveTranslationFor</code>	148
seq commands:	
<code>\seq_clear:N</code>	338, 886
<code>\seq_get_left:NN</code>	1271
<code>\seq_if_empty:NTF</code>	1266
<code>\seq_if_in:NnTF</code>	202, 874
<code>\seq_map_break:n</code>	81, 1185, 1188
<code>\seq_map_function:NN</code>	889
<code>\seq_map_indexed_inline:Nn</code>	13, 1145
<code>\seq_map_inline:Nn</code>	1182
<code>\seq_map_tokens:Nn</code>	63
<code>\seq_new:N</code>	195, 326, 848, 883, 1205
<code>\seq_pop_left:NN</code>	1265
<code>\seq_put_right:Nn</code>	204, 878
<code>\seq_reverse:N</code>	332
<code>\seq_set_eq:NN</code>	1241
<code>\seq_set_from_clist:Nn</code>	331, 825
<code>\seq_sort:Nn</code>	892
sort commands:	
<code>\sort_return_same:</code>	30, 35, 899, 904, 941, 986, 988, 1041, 1047, 1064, 1070, 1093, 1124, 1131, 1169, 1185, 1201
<code>\sort_return_swapped:</code>	30, 35, 912, 950, 985, 1040, 1087, 1123, 1177, 1188, 1200
str commands:	
<code>\str_case:nnTF</code>	455, 493
<code>\str_if_eq:nnTF</code>	80
<code>\str_if_eq_p:nn</code>	1947, 1953, 1955, 1959
<code>\str_new:N</code>	398
<code>\str_set:Nn</code>	403, 405, 407, 409
T	
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@Alph</code>	57
<code>\@addtoreset</code>	4
<code>\@chapapp</code>	57
<code>\@currentcounter</code>	4, 21, 25, 28, 30, 33, 84, 86
<code>\@currentlabel</code>	3
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	257, 272, 382, 517, 534, 575

\@onlypreamble	159, 160	\tl_if_empty:nTF	163, 722, 1447, 1462, 1477, 1718, 1743, 1754, 1767, 1836
\@trnslt@current@language	16, 516	\tl_if_empty_p:N	938, 939, 947, 948, 955, 956, 1312, 1313, 1320, 1321, 1946, 1956, 1960, 2119, 2163
\@trnslt@language	472, 524, 541	\tl_if_empty_p:n	1027, 1028, 1055, 1078
\bbl@loaded	521	\tl_if_eq:NNTF	962, 1101, 1324
\bbl@main@language	16, 520	\tl_if_eq:NnTF	887, 920, 1155, 1158, 1184, 1187, 1273, 2178
\c@	4	\tl_if_eq:nnTF	977, 1031, 1147, 2180, 2202, 2206
\c@page	6, 94	\tl_if_in:NnTF	1060, 1083
\cl@	4, 5	\tl_if_novalue:nTF	622, 692
\hyper@link	1550, 1798, 1980, 2064	\tl_map_break:n	81
\p@...	3	\tl_map_tokens:Nn	73
\xpg@loaded	538	\tl_new:N	89, 245, 445, 446, 447, 557, 560, 637, 638, 856, 857, 858, 859, 860, 861, 862, 863, 1206, 1207, 1208, 1209, 1214, 1218, 1219, 1220, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1237, 1238
\xpg@main@language	16, 537	\tl_put_left:Nn	1529, 1536, 1585
\zref@addprop	22, 32, 34, 36, 87, 88, 99	\tl_put_right:Nn	1397, 1413, 1422, 1449, 1459, 1474, 1705, 1716, 1741, 1752, 1765, 1849, 1850, 1859
\zref@default	48, 1828, 1972, 2033, 2039, 2106	\tl_reverse_items:n	867, 998, 1002, 1006, 1010
\zref@extractdefault	48, 871, 930, 932, 978, 979, 982, 984, 996, 1000, 1004, 1008, 1032, 1033, 1037, 1039, 1059, 1082, 1197, 1199, 1281, 1286, 1555, 1560, 1566, 1801, 1802, 1804, 1807, 1821, 1985, 1989, 1994, 2009, 2069, 2073, 2078, 2083, 2098, 2181, 2182, 2185, 2187, 2191, 2193, 2203, 2204, 2207, 2208, 2211, 2213, 2217, 2219	\tl_set:Nn	250, 252, 258, 261, 277, 286, 470, 507, 551, 643, 714, 727, 870, 929, 931, 995, 997, 999, 1001, 1003, 1005, 1007, 1009, 1018, 1020, 1058, 1081, 1108, 1110, 1112, 1114, 1275, 1276, 1279, 1284, 1388, 1389, 1512, 1543, 1661, 1662, 1685, 1845, 1846, 1858
\zref@ifpropundefined	12	\tl_set_eq:NN	459, 465, 497, 503, 515, 519, 536, 1623
\zref@ifrefcontainsprop	12, 1552, 1787, 1800, 1976, 1982, 2049, 2066	\tl_tail:N	1109, 1111, 1113, 1115
\zref@ifrefundefined	894, 896, 908, 1298, 1300, 1305, 1338, 1497, 1506, 1648, 1833, 1971	\l_tmpa_tl	1014, 1018, 1027, 1055, 1058, 1061, 1101
\ZREF@mainlist	22, 32, 34, 36, 87, 88, 99	\l_tmpb_tl	1015, 1020, 1028, 1078, 1081, 1084, 1101
\zref@newprop	5, 21, 23, 33, 35, 83, 85, 98		
\zref@refused	1337		
\zref@wrapper@babel	26, 820		
\textendash	2237		
\the	3		
\thechapter	57		
\thepage	6, 95		
\thesection	57		
tl commands:			
\c_empty_tl	871, 930, 932, 996, 1000, 1004, 1008, 1282, 1287		
\c_novalue_tl	619, 689		
\tl_clear:N	715, 723, 1014, 1015, 1242, 1243, 1244, 1245, 1246, 1268, 1625, 1626, 1627, 1628, 1669, 1834, 1837, 1861, 1885, 1932, 2171, 2173		
\tl_gset:Nn	95		
\tl_head:N	1019, 1021, 1104, 1106, 1120, 1122		
\tl_if_empty:NNTF	71, 744, 775, 806, 872, 1341, 1495, 1848, 1863, 2037		

U

use commands:

\use:N	21
\UseHook	555

Z

\zcDeclareTranslations	21, 22, 24, 103, 712
\zcDicDefaultTransl	9, 155
\zcDicTypeTransl	9, 155

<code>\zcheck</code>	121	<code>_zrefclever_get_transl:nnn</code> ...	
<code>\zcpageref</code>	28, 850	. 8, 147 , 1877, 1909, 1924, 2139, 2149	
<code>\zceref</code> . 20, 21, 26, 29, 36, 37, 819 , 853, 854		<code>_zrefclever_if_transl:nn</code> . 140, 146	
<code>\zcRefTypeSetup</code>	21, 22, 109, 639	<code>_zrefclever_if_transl:nnTF</code> ...	
<code>\zcsetup</code>	20, 21, 633 8, 140 , 1870, 1902, 1917, 2135	
zrefcheck commands:		<code>\l_zrefclever_label_a_tl</code>	
<code>\zrefcheck_zceref_beg_label:</code> ...	830 856 , 1265, 1282, 1298, 1337,	
<code>\zrefcheck_zceref_end_label_-</code>		1338, 1344, 1388, 1400, 1416, 1432,	
maybe:	842	1468, 1483, 1510, 1517, 1648, 1652,	
<code>\zrefcheck_zceref_run_checks_on_-</code>		1661, 1685, 1708, 1726, 1760, 1773	
labels:n	843	<code>\l_zrefclever_label_b_tl</code>	
zrefclever internal commands:		856 , 1268, 1271, 1287, 1300, 1305, 1652	
<code>\l_zrefclever_abbrev_bool</code>		<code>\l_zrefclever_label_count_int</code> ..	
..... 431, 435, 1852	 36 , 1210 ,	
<code>\l_zrefclever_capitalize_bool</code> ..		1247, 1350, 1382, 1629, 1657, 1783	
..... 413, 417, 1840		<code>\l_zrefclever_label_enclcnt_a_-</code>	
<code>\l_zrefclever_capitalize_first_-</code>		tl	
bool	414, 423, 1842 856 ,	
<code>_zrefclever_counter_reset_by:n</code>		995, 997, 998, 1019, 1084, 1108, 1109	
5, 6, 10, 11, 39, 41, 43, 48, 50, 52, 57		<code>\l_zrefclever_label_enclcnt_b_-</code>	
<code>_zrefclever_counter_reset_by_-</code>		tl	
aux:nn	64, 67 856 ,	
<code>_zrefclever_counter_reset_by_-</code>		999, 1001, 1002, 1021, 1061, 1110, 1111	
auxi:nnn	74, 78	<code>\l_zrefclever_label_enclval_a_-</code>	
<code>\l_zrefclever_counter_resetby_-</code>		tl	
prop	5, 11, 60, 61, 221, 233 856 , 1003,	
<code>\l_zrefclever_counter_resettters_-</code>		1005, 1006, 1104, 1112, 1113, 1120	
seq .. 4, 5, 10, 11, 63, 195, 202, 205		<code>\l_zrefclever_label_enclval_b_-</code>	
<code>\l_zrefclever_counter_type_prop</code>		tl	
..... 3, 9, 25, 28, 167, 179	 856 , 1007,	
<code>\l_zrefclever_current_language_-</code>		1009, 1010, 1106, 1114, 1115, 1122	
tl	16, 447, 466, 504, 515	<code>\l_zrefclever_label_type_a_tl</code> ..	
<code>_zrefclever_declare_fallback_-</code>	 856 , 870, 872, 876,	
transl:nn	9, 153	879, 929, 938, 947, 955, 963, 1155,	
<code>_zrefclever_declare_transl:nnn</code>		1184, 1275, 1279, 1312, 1320, 1325,	
..... 8, 150 , 746, 777, 782, 812		1341, 1389, 1662, 2119, 2122, 2126,	
<code>_zrefclever_get_enclosing_-</code>		2131, 2137, 2141, 2163, 2166, 2170	
counters:n	5, 37 , 42, 84	<code>\l_zrefclever_label_type_b_tl</code> ..	
<code>_zrefclever_get_enclosing_-</code>	 856 , 931, 939, 948, 956, 964, 1158,	
counters_value:n ...	5, 37 , 51, 86	1187, 1276, 1284, 1313, 1321, 1325	
<code>_zrefclever_get_option_-</code>		<code>_zrefclever_label_type_put_-</code>	
plain:nN		new_right:n	
..... 20, 21, 1352, 1353, 1354, 2156	 28, 868 , 890	
<code>_zrefclever_get_option_with_-</code>		<code>\l_zrefclever_label_types_seq</code> ..	
transl:nN 28, 875, 878, 883 , 886, 1182	
..... 20, 21, 57, 58, 1253, 1254,		<code>_zrefclever_labels_in_sequence:nn</code>	
1255, 1256, 1355, 1356, 1357, 1358,	 1509, 1651, 2176	
1359, 1360, 1361, 1362, 1363, 2110		<code>\l_zrefclever_last_of_type_bool</code>	
<code>_zrefclever_get_ref:n</code> 1400, 1416,	 36, 1203 , 1296, 1301, 1302,	
1428, 1432, 1452, 1465, 1468, 1480,		1306, 1315, 1326, 1327, 1330, 1368	
1483, 1517, 1537, 1708, 1721, 1726,		<code>\l_zrefclever_lastsep_tl</code> . 1226,	
1746, 1757, 1760, 1770, 1773, 1785		1359, 1415, 1431, 1451, 1467, 1479	
<code>_zrefclever_get_ref_first:</code> ...		<code>\l_zrefclever_link_star_bool</code> ...	
..... 37, 48, 1530, 1586, 1969	 826, 848 , 1790, 1945, 2055	
		<code>\l_zrefclever_listsep_tl</code>	
		... 1225, 1358, 1427, 1464, 1707,	
		1720, 1725, 1745, 1756, 1759, 1769	
		<code>\l_zrefclever_main_language_tl</code> .	
	 16, 446, 460, 498, 519, 536, 551	

\l_zrefclever_name_format_- fallback_tl	1235, 1858, 1861, 1863, 1899, 1921, 1928	497, 503, 507, 1871, 1878, 1903, 1910, 1918, 1925, 2136, 2140, 2150
\l_zrefclever_name_format_tl	1235, 1845, 1846, 1849, 1850, 1858, 1859, 1867, 1874, 1881, 1894, 1906, 1913	\l_zrefclever_ref_options_prop
\l_zrefclever_name_in_link_bool	1235, 1545, 1949, 1965, 1966, 1974	20, 22, 596, 623, 624, 2114, 2159
\l_zrefclever_namefont_tl	1218, 1352, 1548, 1574, 1999, 2029, 2044	\l_zrefclever_ref_property_tl
\l_zrefclever_nameinlink_str	398, 403, 405, 407, 409, 1947, 1953, 1955, 1959	11, 12, 245, 250, 252, 258, 261, 277, 286, 887, 920, 1273, 1787, 1807, 1821, 1977, 2010, 2050, 2084, 2099, 2178
\l_zrefclever_namesep_tl	1222, 1355, 2002, 2032, 2040, 2047	\l_zrefclever_ref_typeset_font_- tl
\l_zrefclever_next_is_same_bool	36, 55, 1212, 1641, 1671, 1687, 1693, 2196, 2222	557, 559, 1259
\l_zrefclever_next_maybe_range_- bool	36, 55, 1212, 1505, 1514, 1640, 1667, 1678, 2188, 2195, 2214, 2221	\l_zrefclever_reffont_in_tl 1220, 1354, 1796, 1819, 2007, 2062, 2096
\l_zrefclever_noabbrev_first_- bool	432, 441, 1855	\l_zrefclever_reffont_out_tl
\l_zrefclever_notesep_tl	836, 1230, 1256	1219, 1353, 1793, 1816, 2004, 2023, 2059, 2093
_zrefclever_page_format_aux:	90, 94	\l_zrefclever_refpos_in_tl 1234, 1363, 1808, 1822, 2011, 2085, 2100
\g_zrefclever_page_format_tl	7, 89, 95, 98	\l_zrefclever_refpos_out_tl 1232, 1361, 1811, 1824, 2024, 2088, 2102
\l_zrefclever_pairsep_tl	1224, 1357, 1399, 1515	\l_zrefclever_refpre_in_tl 1233, 1362, 1806, 1820, 2008, 2082, 2097
_zrefclever_prop_put_non_- empty:Nnn	9, 161, 178, 232	\l_zrefclever_refpre_out_tl 1231, 1360, 1794, 1817, 2005, 2060, 2094
\l_zrefclever_range_beg_label_- tl	36, 1212, 1246, 1428, 1447, 1452, 1462, 1465, 1477, 1480, 1628, 1669, 1685, 1718, 1721, 1743, 1746, 1754, 1757, 1767, 1770	\l_zrefclever_setup_language_tl
\l_zrefclever_range_count_int	36, 1212, 1249, 1408, 1440, 1631, 1670, 1682, 1686, 1692, 1700, 1737, 1778	637, 714, 747, 778, 783, 813
\l_zrefclever_range_inhibit_- next_bool	36, 1212, 1646	\l_zrefclever_setup_type_tl
\l_zrefclever_range_same_count_- int	36, 1212, 1250, 1395, 1429, 1440, 1632, 1672, 1688, 1694, 1723, 1737, 1779	637, 643, 697, 705, 715, 723, 727, 744, 775, 784, 806, 814
\l_zrefclever_rangesep_tl	1223, 1356, 1482, 1516, 1772	\l_zrefclever_sort_decided_bool
\l_zrefclever_ref_language_tl	16, 445, 459, 465, 470,	866, 1012, 1016, 1035, 1046, 1063, 1069, 1086, 1092, 1118, 1130
		_zrefclever_sort_default:nn
		28–30, 922, 927
		_zrefclever_sort_default_- different_types:nn
		13, 971, 1138
		_zrefclever_sort_default_same_- type:nn
		967, 993
		_zrefclever_sort_labels:
		28, 30, 35, 834, 884
		_zrefclever_sort_page:nn
		35, 921, 1194
		\l_zrefclever_sort_prior_a_int
		864, 1140, 1149, 1150, 1156, 1166, 1174
		\l_zrefclever_sort_prior_b_int
		865, 1141, 1151, 1152, 1159, 1167, 1175
		\l_zrefclever_tlastsep_tl
		1229, 1255, 1617
		\l_zrefclever_tlistsep_tl
		1228, 1254, 1595
		\l_zrefclever_tpairsep_tl
		1227, 1253, 1611
		\l_zrefclever_type_<type>- options_prop
		22

\l__zrefclever_type_count_int ...	1612, 1618, 1624, 1625, 1705, 1716,
..... 36, <u>1210</u> , 1248, 1592,	1741, 1752, 1765, 1848, 1956, 1960
1594, 1603, 1630, 1843, 1854, 1962	
\l__zrefclever_type_first_label_-	\l__zrefclever_typeset_queue_-
tl <u>1205</u> , <u>1244</u> , 1388, 1497,	prev_tl <u>1205</u> , <u>1242</u> , 1596, 1624
1506, 1510, 1537, 1553, 1556, 1561,	\l__zrefclever_typeset_range_-
1567, 1626, 1661, 1833, 1971, 1977,	bool 350, 353, 833, 1493
1983, 1985, 1989, 1994, 2009, 2050,	\l__zrefclever_typeset_ref_bool .
2067, 2069, 2073, 2078, 2083, 2098 291, 298, 303, 308, 1527, 1534
\l__zrefclever_type_first_label_-	__zrefclever_typeset_refs:
type_tl <u>1205</u> , 36, 37, 48, 49, 51, 835, <u>1239</u>
1245, 1389, 1501, 1627, 1662, 1836,	__zrefclever_typeset_refs_aux_-
1866, 1873, 1880, 1887, 1893,	last_of_type: <u>1371</u> , <u>1379</u>
1898, 1905, 1912, 1920, 1927, 1934	__zrefclever_typeset_refs_aux_-
__zrefclever_type_name_setup: ..	not_last_of_type: <u>1375</u> , <u>1635</u>
..... 37, 1525, <u>1831</u>	\l__zrefclever_typeset_sort_bool
\l__zrefclever_type_name_tl . 49, 317, 320, 832
<u>1235</u> , 1569, 1575, 1834, 1837, 1868,	\l__zrefclever_typesort_seq
1877, 1885, 1895, 1900, 1909, 1924, 13, 326, 331, 332, 338, 1145
1932, 1946, 2000, 2030, 2037, 2045	\l__zrefclever_use_hyperref_bool
\l__zrefclever_typeset_compress_- 357, 364,
bool 341, 344, 1643	369, 374, 384, 390, 1790, 1944, 2054
\l__zrefclever_typeset_labels_-	\l__zrefclever_warn_hyperref_-
seq ... <u>1205</u> , <u>1241</u> , <u>1265</u> , <u>1266</u> , <u>1271</u>	bool 358, 365, 370, 375, 388
\l__zrefclever_typeset_last_bool	__zrefclever_zcref:nnn 820, 821
..... 36, <u>1203</u> ,	__zrefclever_zcref:nnnn . 26, 29, <u>821</u>
1262, 1263, 1269, 1295, 1600, 1961	\l__zrefclever_zcref_labels_seq .
\l__zrefclever_typeset_name_bool 29, 825, 844, <u>848</u> , 889, 892, 1241
..... 292, 299, 304, 309, 1527, 1541	\l__zrefclever_zcref_note_tl ...
\l__zrefclever_typeset_queue_- 560, 563, 837
curr_tl <u>1205</u> , <u>1243</u> , 1397,	\l__zrefclever_zcref_with_check_-
1413, 1422, 1449, 1459, 1474, 1495,	bool 567, 582, 829, 840
1512, 1529, 1536, 1543, 1586, 1607,	\l__zrefclever_zrefcheck_-
	available_bool 566, 577, 588, 828, 839