

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-29

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	Reference format	9
4.3	Languages	10
4.4	Dictionaries	11
4.5	Options	17
5	Configuration	30
5.1	\zcsetup	30
5.2	\zcRefTypeSetup	30
5.3	\zcLanguageSetup	31
6	User interface	34
6.1	\zceref	34
6.2	\zcpageref	35
7	Sorting	36
8	Typesetting	45
9	Compatibility	69
9.1	\footnote	69
9.2	\appendix	70
9.3	appendix package	71
9.4	listings package	71
9.5	enumitem package	72

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

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

10	Dictionaries	73
10.1	English	73
10.2	German	77
10.3	French	80
10.4	Portuguese	84
10.5	Spanish	87
	Index	91

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 dictionaries (which became the default input encoding in the 2018-04-01 release). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut with the inclusion of the new hook management system (`ltxcmdhooks`), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

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

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

Identify the package.

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

2 Dependencies

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

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

3 zref setup

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

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

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

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

The reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the<counter>` and store it “clean” in `zc@thecnt` for reserved use. Since `\@currentlabel`, which populates the `default` property, is *more reliable* than `\@currentcounter`, `zc@thecnt` is meant to be kept as an *option* (`ref` option), in case there’s need to use `zref-clever` together with `\labelformat`. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@...` prefix.

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

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

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

Since the `default`, `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’).

```

36 \zref@newprop { zc@cntval } [0]
37 { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
38 \zref@addprop \ZREF@mainlist { zc@cntval }
39 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
40 \zref@addprop \ZREF@mainlist { zc@pgval }

```

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

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

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of `\newcounter`, `\@addtoreset`, `\counterwithin`, and related infrastructure). The canonical optional argument of `\newcounter` establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “super-counter”, “parent counter” etc.). This information is a little trickier to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\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 (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresettters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other

means). Therefore, inspecting `\c1@⟨counter⟩` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`__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⟩}

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

```

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

```

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

```

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

`__zrefclever_counter_reset_by:n`

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

```

        \__zrefclever_counter_reset_by:n {<counter>}
61 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
62 {
63     \bool_if:nTF
64     { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
65     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
66     {
67         \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
68         { \__zrefclever_counter_reset_by_aux:nn {#1} }
69     }
70 }
71 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
72 {
73     \cs_if_exist:cT { c@ #2 }
74     {
75         \tl_if_empty:cF { c1@ #2 }
76         {
77             \tl_map_tokens:cn { c1@ #2 }
78             { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
79         }
80     }
81 }
82 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
83 {
84     \str_if_eq:nnT {#2} {#3}
85     { \tl_map_break:n { \seq_map_break:n {#1} } }
86 }

```

(End definition for __zrefclever_counter_reset_by:n.)

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

```

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

```

Another piece of information we need is the page numbering format being used by `\thepage`, so that we know when we can (or not) group a set of page references in a range. Unfortunately, `page` is not a typical counter in ways which complicates things. First, it does commonly get reset along the document, not necessarily by the usual counter reset chains, but rather with `\pagenumbering` or variations thereof. Second, the format of the page number commonly changes in the document (roman, arabic, etc.), not necessarily, though usually, together with a reset. Trying to “parse” `\thepage` to retrieve such information is bound to go wrong: we don’t know, and can’t know, what is within that macro, and that’s the business of the user, or of the documentclass, or of the loaded packages. The technique used by `cleveref`, which we borrow here, is simple and smart: store with the label what `\thepage` would return, if the counter `\c@page` was “1”. That does not allow us to *sort* the references, luckily however, we have `abspage` which solves this problem. But we can decide whether two labels can be compressed into a range or not based on this format: if they are identical, we can compress them, otherwise, we can’t. To do so, we locally redefine `\c@page` to return “1”, thus avoiding

any global spillovers of this trick. Since this operation is not expandable we cannot run it directly from the property definition. Hence, we use a shipout hook, and set `\g__zrefclever_page_format_tl`, which can then be retrieved by the starred definition of `\zref@newprop*{zc@pgfmt}`.

```

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

```

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

4 Plumbing

4.1 Messages

```

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

```

```

132 {
133   Language~'#1'~is-unknown~\msg_line_context:..Using~default.~
134   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
135   '\iow_char:N\zcDeclareLanguageAlias'.
136 }
137 \msg_new:nnn { zref-clever } { dict-loaded }
138 { Loaded~'#1'~dictionary. }
139 \msg_new:nnn { zref-clever } { dict-not-available }
140 { Dictionary~for~'#1'~not~available~\msg_line_context:.. }
141 \msg_new:nnn { zref-clever } { unknown-language-load }
142 {
143   Language~'#1'~is-unknown~\msg_line_context:..Unable~to~load~dictionary.~
144   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
145   '\iow_char:N\zcDeclareLanguageAlias'.
146 }
147 \msg_new:nnn { zref-clever } { missing-zref-titleref }
148 {
149   Option~'ref=title'~requested~\msg_line_context:..
150   But~package~'zref-titleref'~is~not~loaded,~falling~back~to~default~'ref'.
151 }
152 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
153 {
154   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:..
155   Use~the~starred~version~of~'\iow_char:N\zceref'~instead.
156 }
157 \msg_new:nnn { zref-clever } { missing-hyperref }
158 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
159 \msg_new:nnn { zref-clever } { titleref-preamble-only }
160 {
161   Option~'titleref'~only~available~in~the~preamble~\msg_line_context:..
162   Did~you~mean~'ref=title'?
163 }
164 \msg_new:nnn { zref-clever } { missing-zref-check }
165 {
166   Option~'check'~requested~\msg_line_context:..
167   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
168 }
169 \msg_new:nnn { zref-clever } { counters-not-nested }
170 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:.. }
171 \msg_new:nnn { zref-clever } { missing-type }
172 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
173 \msg_new:nnn { zref-clever } { missing-name }
174 { Name~undefined~for~type~'#1'~\msg_line_context:.. }
175 \msg_new:nnn { zref-clever } { missing-string }
176 {
177   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:..
178   But~we~should~have:~throw~a~rock~at~the~maintainer.
179 }
180 \msg_new:nnn { zref-clever } { single-element-range }
181 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
182 \msg_new:nnn { zref-clever } { compat-package }
183 { Loaded~support~for~'#1'~package. }
184 \msg_new:nnn { zref-clever } { compat-class }
185 { Loaded~support~for~'#1'~documentclass. }

```


4.2 Reference format

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

`\l__zrefclever_setup_type_tl` Store “current” type and language in different places for option and translation handling, notably in `__zrefclever_provide_dictionary:n`, `\zcRefTypeSetup`, and `\zcLanguageSetup`. But also for translations retrieval, in `__zrefclever_get_type_transl:nnnN` and `__zrefclever_get_default_transl:nnN`.

```
186 \tl_new:N \l__zrefclever_setup_type_tl
187 \tl_new:N \l__zrefclever_dict_language_tl
```

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

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

```
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

188 \seq_const_from_clist:Nn
189 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
190 {
191     tpairsep ,
192     tlistsep ,
193     tlastsep ,
194     notesep ,
195 }
196 \seq_const_from_clist:Nn
197 \c__zrefclever_ref_options_possibly_type_specific_seq
198 {
199     namesep ,
200     pairsep ,
201     listsep ,
202     lastsep ,
203     rangesep ,
204     refpre ,
205     refpos ,
206     refpre-in ,
207     refpos-in ,
208 }
```

Only “type names” are “necessarily type-specific”, which makes them somewhat special on the retrieval side of things. In short, they don’t have their values queried by `__zrefclever_get_ref_string:nN`, but by `__zrefclever_type_name_setup:`.

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

```

217     name-sg-ab ,
218     Name-pl-ab ,
219     name-pl-ab ,
220 }

```

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

```

221 \seq_const_from_clist:Nn
222   \c__zrefclever_ref_options_font_seq
223 {
224     namefont ,
225     reffont ,
226     reffont-in ,
227 }
228 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
229 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
230   \c__zrefclever_ref_options_possibly_type_specific_seq
231   \c__zrefclever_ref_options_necessarily_type_specific_seq
232 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
233   \c__zrefclever_ref_options_typesetup_seq
234   \c__zrefclever_ref_options_font_seq
235 \seq_new:N \c__zrefclever_ref_options_reference_seq
236 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
237   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
238   \c__zrefclever_ref_options_possibly_type_specific_seq
239 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
240   \c__zrefclever_ref_options_reference_seq
241   \c__zrefclever_ref_options_font_seq

```

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

4.3 Languages

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

```

242 \prop_new:N \g__zrefclever_languages_prop

```

(End definition for `\g__zrefclever_languages_prop`.)

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

```

\zcDeclareLanguage {\language}}

243 \NewDocumentCommand \zcDeclareLanguage { m }
244 {
245     \tl_if_empty:nF {#1}
246     {
247         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}

```

```

248         { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
249         { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
250     }
251 }
252 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

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

```

\zcDeclareLanguageAlias {<language alias>} {<aliased language>}

253 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
254 {
255     \tl_if_empty:nF {#1}
256     {
257         \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
258         {
259             \exp_args:NNnx
260             \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
261             { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
262         }
263         { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
264     }
265 }
266 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

4.4 Dictionaries

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

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

This design decision has also implications to the *form* the dictionary files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`.

This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.lbx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load dictionaries on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it. Hence, `zref-clever`’s built-in dictionary files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/dictionary}` by `__zrefclever_provide_dictionary:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The dictionary file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

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

Provide

<code>\g__zrefclever_loaded_dictionaries_seq</code>	Used to keep track of whether a dictionary has already been loaded or not.
267	<code>\seq_new:N \g__zrefclever_loaded_dictionaries_seq</code>
	(End definition for <code>\g__zrefclever_loaded_dictionaries_seq</code> .)
<code>\l__zrefclever_load_dict_verbose_bool</code>	Controls whether <code>__zrefclever_provide_dictionary:n</code> fails silently or verbosely in case of unknown languages or dictionaries not found.
268	<code>\bool_new:N \l__zrefclever_load_dict_verbose_bool</code>
	(End definition for <code>\l__zrefclever_load_dict_verbose_bool</code> .)
<code>__zrefclever_provide_dictionary:n</code>	Load dictionary for known <code>\langle language \rangle</code> if it is available and if it has not already been loaded.
	<code>__zrefclever_provide_dictionary:n {\langle language \rangle}</code>
269	<code>\cs_new_protected:Npn __zrefclever_provide_dictionary:n #1</code>
270	<code>{</code>
271	<code>\group_begin:</code>
272	<code>\prop_get:NnNTF \g__zrefclever_languages_prop {#1}</code>
273	<code>\l__zrefclever_dict_language_tl</code>
274	<code>{</code>
275	<code>\seq_if_in:NVF</code>
276	<code>\g__zrefclever_loaded_dictionaries_seq</code>
277	<code>\l__zrefclever_dict_language_tl</code>
278	<code>{</code>
279	<code>\exp_args:Nx \file_get:nnNTF</code>
280	<code>{ zref-clever- \l__zrefclever_dict_language_tl .dict }</code>
281	<code>{ \ExplSyntaxOn }</code>

```

282         \l_tmpa_tl
283     {
284         \prop_if_exist:cF
285         {
286             g__zrefclever_dict_
287             \l__zrefclever_dict_language_tl _prop
288         }
289         {
290             \prop_new:c
291             {
292                 g__zrefclever_dict_
293                 \l__zrefclever_dict_language_tl _prop
294             }
295         }
296         \tl_clear:N \l__zrefclever_setup_type_tl
297         \exp_args:NnV
298             \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
299         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
300             \l__zrefclever_dict_language_tl
301         \msg_note:nnx { zref-clever } { dict-loaded }
302             { \l__zrefclever_dict_language_tl }
303     }
304     {
305         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
306         {
307             \msg_warning:nnx { zref-clever } { dict-not-available }
308             { \l__zrefclever_dict_language_tl }
309         }
310     }

```

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

```

310         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
311             \l__zrefclever_dict_language_tl
312     }
313 }
314 }
315 {
316     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
317     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
318 }
319 \group_end:
320 }
321 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }

```

(End definition for `__zrefclever_provide_dictionary:n`.)

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

`__zrefclever_provide_dictionary_verbose:n {<language>}`

```

322 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
323 {
324   \group_begin:
325   \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
326   \__zrefclever_provide_dictionary:n {#1}
327   \group_end:
328 }
329 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn
__zrefclever_provide_dict_default_transl:nn

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

```

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

330 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
331 {
332   \exp_args:Nnx \prop_gput_if_new:cnn
333   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
334   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
335 }
336 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
337 {
338   \prop_gput_if_new:cnn
339   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
340   { default- #1 } {#2}
341 }

```

(End definition for __zrefclever_provide_dict_type_transl:nn and __zrefclever_provide_dict_default_transl:nn.)

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

```

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

```

```

355     {
356       #1 .value_required:n = true ,
357       #1 .code:n =
358       {
359         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
360         { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
361         {
362           \msg_info:nnn { zref-clever }
363           { option-not-type-specific } {#1}
364         }
365       } ,
366     }
367   }
368   \seq_map_inline:Nn
369   \c__zrefclever_ref_options_possibly_type_specific_seq
370   {
371     \keys_define:nn { zref-clever / dictionary }
372     {
373       #1 .value_required:n = true ,
374       #1 .code:n =
375       {
376         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
377         { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
378         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
379       } ,
380     }
381   }
382   \seq_map_inline:Nn
383   \c__zrefclever_ref_options_necessarily_type_specific_seq
384   {
385     \keys_define:nn { zref-clever / dictionary }
386     {
387       #1 .value_required:n = true ,
388       #1 .code:n =
389       {
390         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
391         {
392           \msg_info:nnn { zref-clever }
393           { option-only-type-specific } {#1}
394         }
395         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
396       } ,
397     }
398   }

```

Fallback

All “strings” queried with `__zrefclever_get_ref_string:nN` – in practice, those in either `\c__zrefclever_ref_options_necessarily_not_type_specific_seq` or `\c__zrefclever_ref_options_possibly_type_specific_seq` – must have their values set for “fallback”, even if to empty ones, since this is what will be retrieved in the absence of a proper translation, which will be the case if `babel` or `polyglossia` is loaded and sets a language which `zref-clever` does not know. On the other hand, “type names” are not looked

for in “fallback”, since it is indeed impossible to provide any reasonable value for them for a “specified but unknown language”. Also “font” options – those in `\c__zrefclever_ref_options_font_seq`, and queried with `__zrefclever_get_ref_font:nN` – do not need to be provided here, since the later function sets an empty value if the option is not found.

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

```

399 \prop_new:N \g__zrefclever_fallback_dict_prop
400 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
401 {
402   tpairsep = {,~} ,
403   tlistsep = {,~} ,
404   tlastsep = {,~} ,
405   notesep  = {~} ,
406   namesep  = {\nobreakspace} ,
407   pairsep  = {,~} ,
408   listsep  = {,~} ,
409   lastsep  = {,~} ,
410   rangesep = {\textendash} ,
411   refpre   = {} ,
412   refpos   = {} ,
413   refpre-in = {} ,
414   refpos-in = {} ,
415 }
```

Get translations

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

```

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

416 \prg_new_protected_conditional:Npnn
417 \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
418 {
419   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
420   \l__zrefclever_dict_language_tl
421   {
422     \prop_get:cnNTF
423     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
424     { type- #2 - #3 } #4
425     { \prg_return_true: }
426     { \prg_return_false: }
427   }
428   { \prg_return_false: }
429 }
430 \prg_generate_conditional_variant:Nnn
431 \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }
```

(End definition for `__zrefclever_get_type_transl:nnnNF`.)

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


```

    \_zrefclever_get_default_transl:nnNF {<language>} {<key>}
    {<tl variable>} {<false code>}}
432 \prg_new_protected_conditional:Npnn
433 \_zrefclever_get_default_transl:nnN #1#2#3 { F }
434 {
435   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
436   \l__zrefclever_dict_language_tl
437   {
438     \prop_get:cnNTF
439     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
440     { default- #2 } #3
441     { \prg_return_true: }
442     { \prg_return_false: }
443   }
444   { \prg_return_false: }
445 }
446 \prg_generate_conditional_variant:Nnn
447 \_zrefclever_get_default_transl:nnN { xnN } { F }
(End definition for \_zrefclever_get_default_transl:nnNF.)

```

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

```

    \_zrefclever_get_fallback_transl:nnNF {<key>}
    {<tl variable>} {<false code>}}
448 % {<key>}<tl var to set>
449 \prg_new_protected_conditional:Npnn
450 \_zrefclever_get_fallback_transl:nnN #1#2 { F }
451 {
452   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
453   { #1 } #2
454   { \prg_return_true: }
455   { \prg_return_false: }
456 }
(End definition for \_zrefclever_get_fallback_transl:nnNF.)

```

4.5 Options

Auxiliary

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

```

    \_zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}
457 \cs_new_protected:Npn \_zrefclever_prop_put_non_empty:Nnn #1#2#3
458 {
459   \tl_if_empty:nTF {#3}
460   { \prop_remove:Nn #1 {#2} }
461   { \prop_put:Nnn #1 {#2} {#3} }
462 }
(End definition for \_zrefclever_prop_put_non_empty:Nnn.)

```

ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these three (or four) alternatives – `default`, `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 current 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.

```
463 \tl_new:N \l__zrefclever_ref_property_tl
464 \keys_define:nn { zref-clever / reference }
465 {
466   ref .choice: ,
467   ref / default .code:n =
468     { \tl_set:Nn \l__zrefclever_ref_property_tl { default } } ,
469   ref / zc@thecnt .code:n =
470     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
471   ref / page .code:n =
472     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
473   ref / title .code:n =
474     {
475       \AddToHook { begindocument }
476       {
477         \@ifpackageloaded { zref-titleref }
478         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
479         {
480           \msg_warning:nn { zref-clever } { missing-zref-titleref }
481           \tl_set:Nn \l__zrefclever_ref_property_tl { default }
482         }
483       }
484     } ,
485   ref .initial:n = default ,
486   ref .default:n = default ,
487   page .meta:n = { ref = page } ,
488   page .value_forbidden:n = true ,
489 }
490 \AddToHook { begindocument }
491 {
492   \@ifpackageloaded { zref-titleref }
493   {
494     \keys_define:nn { zref-clever / reference }
495     {
496       ref / title .code:n =
497         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
498     }
499   }
500   {
501     \keys_define:nn { zref-clever / reference }
502     {
```

```

503         ref / title .code:n =
504         {
505             \msg_warning:nn { zref-clever } { missing-zref-titleref }
506             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
507         }
508     }
509 }
510 }

```

typeset option

```

511 \bool_new:N \l__zrefclever_typeset_ref_bool
512 \bool_new:N \l__zrefclever_typeset_name_bool
513 \keys_define:nn { zref-clever / reference }
514 {
515     typeset .choice: ,
516     typeset / both .code:n =
517     {
518         \bool_set_true:N \l__zrefclever_typeset_ref_bool
519         \bool_set_true:N \l__zrefclever_typeset_name_bool
520     } ,
521     typeset / ref .code:n =
522     {
523         \bool_set_true:N \l__zrefclever_typeset_ref_bool
524         \bool_set_false:N \l__zrefclever_typeset_name_bool
525     } ,
526     typeset / name .code:n =
527     {
528         \bool_set_false:N \l__zrefclever_typeset_ref_bool
529         \bool_set_true:N \l__zrefclever_typeset_name_bool
530     } ,
531     typeset .initial:n = both ,
532     typeset .value_required:n = true ,
533
534     noname .meta:n = { typeset = ref },
535     noname .value_forbidden:n = true ,
536 }

```

sort option

```

537 \bool_new:N \l__zrefclever_typeset_sort_bool
538 \keys_define:nn { zref-clever / reference }
539 {
540     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
541     sort .initial:n = true ,
542     sort .default:n = true ,
543     nosort .meta:n = { sort = false },
544     nosort .value_forbidden:n = true ,
545 }

```

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`.

```

546 \seq_new:N \l__zrefclever_typesort_seq
547 \keys_define:nn { zref-clever / reference }
548 {
549   typesort .code:n =
550   {
551     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
552     \seq_reverse:N \l__zrefclever_typesort_seq
553   } ,
554   typesort .initial:n =
555   { part , chapter , section , paragraph } ,
556   typesort .value_required:n = true ,
557   notypesort .code:n =
558   { \seq_clear:N \l__zrefclever_typesort_seq } ,
559   notypesort .value_forbidden:n = true ,
560 }

```

comp option

```

561 \bool_new:N \l__zrefclever_typeset_compress_bool
562 \keys_define:nn { zref-clever / reference }
563 {
564   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
565   comp .initial:n = true ,
566   comp .default:n = true ,
567   nocomp .meta:n = { comp = false } ,
568   nocomp .value_forbidden:n = true ,
569 }

```

range option

```

570 \bool_new:N \l__zrefclever_typeset_range_bool
571 \keys_define:nn { zref-clever / reference }
572 {
573   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
574   range .initial:n = false ,
575   range .default:n = true ,
576 }

```

cap and capfirst options

```

577 \bool_new:N \l__zrefclever_capitalize_bool
578 \bool_new:N \l__zrefclever_capitalize_first_bool
579 \keys_define:nn { zref-clever / reference }
580 {
581   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
582   cap .initial:n = false ,
583   cap .default:n = true ,
584   nocap .meta:n = { cap = false } ,
585   nocap .value_forbidden:n = true ,
586
587   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
588   capfirst .initial:n = false ,
589   capfirst .default:n = true ,

```

```
590 }
```

abbrev and noabbrevfirst options

```
591 \bool_new:N \l__zrefclever_abbrev_bool
592 \bool_new:N \l__zrefclever_noabbrev_first_bool
593 \keys_define:nn { zref-clever / reference }
594 {
595     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
596     abbrev .initial:n = false ,
597     abbrev .default:n = true ,
598     noabbrev .meta:n = { abbrev = false },
599     noabbrev .value_forbidden:n = true ,
600
601     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
602     noabbrevfirst .initial:n = false ,
603     noabbrevfirst .default:n = true ,
604 }
```

S option

```
605 \keys_define:nn { zref-clever / reference }
606 {
607     S .meta:n =
608     { capfirst = true , noabbrevfirst = true },
609     S .value_forbidden:n = true ,
610 }
```

hyperref option

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

```

637     {
638         \bool_if:NT \l__zrefclever_use_hyperref_bool
639         { \RequirePackage { zref-hyperref } }
640     }
641     {
642         \bool_if:NT \l__zrefclever_warn_hyperref_bool
643         { \msg_warning:nn { zref-clever } { missing-hyperref } }
644         \bool_set_false:N \l__zrefclever_use_hyperref_bool
645     }
646     \keys_define:nn { zref-clever / reference }
647     {
648         hyperref .code:n =
649         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
650     }
651 }

```

nameinlink option

```

652 \str_new:N \l__zrefclever_nameinlink_str
653 \keys_define:nn { zref-clever / reference }
654 {
655     nameinlink .choice: ,
656     nameinlink / true .code:n =
657     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
658     nameinlink / false .code:n =
659     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
660     nameinlink / single .code:n =
661     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
662     nameinlink / tsingle .code:n =
663     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
664     nameinlink .initial:n = tsingle ,
665     nameinlink .default:n = true ,
666 }

```

lang option

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

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

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

```

667 \tl_new:N \l__zrefclever_ref_language_tl
668 \tl_new:N \l__zrefclever_main_language_tl
669 \tl_new:N \l__zrefclever_current_language_tl
670 \AddToHook { begindocument }
671 {
672   \ifpackageloaded { babel }
673   {
674     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
675     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
676   }
677   {
678     \ifpackageloaded { polyglossia }
679     {
680       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
681       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
682     }
683     {
684       \tl_set:Nn \l__zrefclever_current_language_tl { english }
685       \tl_set:Nn \l__zrefclever_main_language_tl { english }
686     }
687   }

```

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

```

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

```

```

707         { current }
708     {
709         \tl_set:Nn \l__zrefclever_ref_language_tl
710             { \l__zrefclever_current_language_tl }
711         \__zrefclever_provide_dictionary_verbose:x
712             { \l__zrefclever_ref_language_tl }
713     }
714 }
715 {
716     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
717     {
718         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
719     }
720     {
721         \msg_warning:nnn { zref-clever }
722             { unknown-language-opt } {#1}
723         \tl_set:Nn \l__zrefclever_ref_language_tl
724             { \l__zrefclever_main_language_tl }
725     }
726     \__zrefclever_provide_dictionary_verbose:x
727         { \l__zrefclever_ref_language_tl }
728 }
729 }
730 } ,
731 lang .value_required:n = true ,
732 }
733 \AddToHook { begindocument / before }
734 {
735     \AddToHook { begindocument }
736     {

```

If any `lang` option has been given by the user, the corresponding language is already loaded, otherwise, ensure the default one (main) gets loaded early, but not verbosely.

```

737     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

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

```



```

753         {
754             \tl_set:Nn \l__zrefclever_ref_language_tl
755                 { \l__zrefclever_current_language_tl }
756             \__zrefclever_provide_dictionary:x
757                 { \l__zrefclever_ref_language_tl }
758         }
759     }
760     {
761         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
762         {
763             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
764         }
765         {
766             \msg_warning:nnn { zref-clever }
767                 { unknown-language-opt } {#1}
768             \tl_set:Nn \l__zrefclever_ref_language_tl
769                 { \l__zrefclever_main_language_tl }
770         }
771         \__zrefclever_provide_dictionary:x
772             { \l__zrefclever_ref_language_tl }
773     }
774     } ,
775     lang .value_required:n = true ,
776 }
777 }
778 }

```

font option

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

```

779 \tl_new:N \l__zrefclever_ref_typeset_font_tl
780 \keys_define:nn { zref-clever / reference }
781 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

782 \keys_define:nn { zref-clever / reference }
783 {
784     titleref .code:n = { \RequirePackage { zref-titleref } } ,
785     titleref .value_forbidden:n = true ,
786 }
787 \AddToHook { begindocument }
788 {
789     \keys_define:nn { zref-clever / reference }
790     {
791         titleref .code:n =
792             { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
793     }
794 }

```

note option

```

795 \tl_new:N \l__zrefclever_zceref_note_tl
796 \keys_define:nn { zref-clever / reference }
797 {
798   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
799   note .value_required:n = true ,
800 }

```

check option

Integration with zref-check.

```

801 \bool_new:N \l__zrefclever_zrefcheck_available_bool
802 \bool_new:N \l__zrefclever_zceref_with_check_bool
803 \keys_define:nn { zref-clever / reference }
804 {
805   check .code:n = { \RequirePackage { zref-check } } ,
806   check .value_forbidden:n = true ,
807 }
808 \AddToHook { begindocument }
809 {
810   \@ifpackageloaded { zref-check }
811   {
812     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
813     \keys_define:nn { zref-clever / reference }
814     {
815       check .code:n =
816       {
817         \bool_set_true:N \l__zrefclever_zceref_with_check_bool
818         \keys_set:nn { zref-check / zcheck } {#1}
819       } ,
820       check .value_required:n = true ,
821     }
822   }
823   {
824     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
825     \keys_define:nn { zref-clever / reference }
826     {
827       check .value_forbidden:n = false ,
828       check .code:n =
829       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
830     }
831   }
832 }

```

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`.

```

833 \prop_new:N \l__zrefclever_counter_type_prop
834 \keys_define:nn { zref-clever / label }
835 {
836   countertype .code:n =
837   {

```

```

838     \keyval_parse:nnn
839     {
840         \msg_warning:nnnn { zref-clever }
841         { key-requires-value } { countertype }
842     }
843     {
844         \__zrefclever_prop_put_non_empty:Nnn
845         \l__zrefclever_counter_type_prop
846     }
847     {#1}
848 } ,
849 countertype .value_required:n = true ,
850 countertype .initial:n =
851 {
852     subsection      = section ,
853     subsubsection   = section ,
854     subparagraph    = paragraph ,
855     enumi            = item ,
856     enumii           = item ,
857     enumiii          = item ,
858     enumiv           = item ,
859     mpfootnote      = footnote ,
860 } ,
861 }

```

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.

```

862 \seq_new:N \l__zrefclever_counter_resetters_seq
863 \keys_define:nn { zref-clever / label }
864 {
865     counterresetters .code:n =
866     {
867         \clist_map_inline:nn {#1}
868         {
869             \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
870             {
871                 \seq_put_right:Nn
872                 \l__zrefclever_counter_resetters_seq {##1}
873             }
874         }
875     } ,
876     counterresetters .initial:n =
877     {
878         part ,
879         chapter ,

```

```

880         section ,
881         subsection ,
882         subsubsection ,
883         paragraph ,
884         subparagraph ,
885     },
886     counterresetters .value_required:n = true ,
887 }

```

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`.

```

888 \prop_new:N \l__zrefclever_counter_resetby_prop
889 \keys_define:nn { zref-clever / label }
890 {
891     counterresetby .code:n =
892     {
893         \keyval_parse:nnn
894         {
895             \msg_warning:nnn { zref-clever }
896             { key-requires-value } { counterresetby }
897         }
898         {
899             \__zrefclever_prop_put_non_empty:Nnn
900             \l__zrefclever_counter_resetby_prop
901         }
902         {#1}
903     } ,
904     counterresetby .value_required:n = true ,
905     counterresetby .initial:n =
906     {

```

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.

```

907         enumii = enumi ,
908         enumiii = enumii ,
909         enumiv = enumiii ,
910     } ,
911 }

```

currentcounter option

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

```

912 \tl_new:N \l__zrefclever_current_counter_tl
913 \keys_define:nn { zref-clever / label }

```

```

914 {
915   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
916   currentcounter .value_required:n = true ,
917   currentcounter .initial:n = \@currentcounter ,
918 }

```

Reference options

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

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

```

919 \prop_new:N \l__zrefclever_ref_options_prop
920 \seq_map_inline:Nn
921   \c__zrefclever_ref_options_reference_seq
922   {
923     \keys_define:nn { zref-clever / reference }
924     {
925       #1 .default:V = \c_novalue_tl ,
926       #1 .code:n =
927       {
928         \tl_if_novalue:nTF {##1}
929         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
930         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
931       } ,
932     }
933 }

```

Package options

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

```

934 \keys_define:nn { }
935 {
936   zref-clever / zcsetup .inherit:n =
937   {
938     zref-clever / label ,
939     zref-clever / reference ,
940   }
941 }

```

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

```

942 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

943 \NewDocumentCommand \zcsetup { m }
944 { \keys_set:nn { zref-clever / zcsetup } {#1} }

(End definition for \zcsetup.)

```

5.2 `\zcRefTypeSetup`

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

```

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

945 \NewDocumentCommand \zcRefTypeSetup { m m }
946 {
947   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
948   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
949   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
950   \keys_set:nn { zref-clever / typesetup } {#2}
951 }

(End definition for \zcRefTypeSetup.)

```

Inside `\zcRefTypeSetup` any of the options *can* receive empty values, and those values, if they exist in the property list, will override translations, regardless of their emptiness. In principle, we could live with the situation of, once a setting has been made in `\l__zrefclever_type_<type>_options_prop` or in `\l__zrefclever_ref_options_prop` it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the translations at any given point. So both in `\zcRefTypeSetup` and in setting reference options (see Section 4.5), we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit at `\keys_set:nn` by means of the `.default:V` property of the key in `\keys_define:nn`. For the technique and some discussion about it, see <https://tex.stackexchange.com/q/614690> (thanks Jonathan P. Spratte, aka ‘Skillmon’, and Phelype Oleinik) and <https://github.com/latex3/latex3/pull/988>.

```

952 \seq_map_inline:Nn
953 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
954 {
955   \keys_define:nn { zref-clever / typesetup }
956   {

```

```

957     #1 .code:n =
958     {
959         \msg_warning:nnn { zref-clever }
960         { option-not-type-specific } {#1}
961     } ,
962 }
963 }
964 \seq_map_inline:Nn
965   \c__zrefclever_ref_options_typesetup_seq
966   {
967     \keys_define:nn { zref-clever / typesetup }
968     {
969       #1 .default:V = \c_novalue_tl ,
970       #1 .code:n =
971       {
972         \tl_if_novalue:nTF {##1}
973         {
974           \prop_remove:cn
975           {
976             l__zrefclever_type_
977             \l__zrefclever_setup_type_tl _options_prop
978           }
979           {#1}
980         }
981         {
982           \prop_put:cnn
983           {
984             l__zrefclever_type_
985             \l__zrefclever_setup_type_tl _options_prop
986           }
987           {#1} {##1}
988         }
989       } ,
990     }
991   }

```

5.3 \zcLanguageSetup

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

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
992 \NewDocumentCommand \zcLanguageSetup { m m }
993 {
994   \group_begin:
995   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
996   \l__zrefclever_dict_language_tl

```

```

997     {
998       \tl_clear:N \l__zrefclever_setup_type_tl
999       \keys_set:nn { zref-clever / langsetup } {#2}
1000     }
1001     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1002   \group_end:
1003 }
1004 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

      \_zrefclever_declare_type_transl:nnnn {<language>} {<type>}
      {<key>} {<translation>}
      \_zrefclever_declare_default_transl:nnn {<language>}
      {<key>} {<translation>}

1005 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1006 {
1007   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1008   { type- #2 - #3 } {#4}
1009 }
1010 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
1011 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
1012 {
1013   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1014   { default- #2 } {#3}
1015 }
1016 \cs_generate_variant:Nn \_zrefclever_declare_default_transl:nnn { Vnn }

```

(End definition for _zrefclever_declare_type_transl:nnnn and _zrefclever_declare_default_transl:nnn.)

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

```

1017 \keys_define:nn { zref-clever / langsetup }
1018 {
1019   type .code:n =
1020   {
1021     \tl_if_empty:NTF {#1}
1022     { \tl_clear:N \l__zrefclever_setup_type_tl }
1023     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1024   } ,
1025 }
1026 \seq_map_inline:Nn
1027 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1028 {
1029   \keys_define:nn { zref-clever / langsetup }
1030   {
1031     #1 .value_required:n = true ,
1032     #1 .code:n =
1033     {
1034       \tl_if_empty:NTF \l__zrefclever_setup_type_tl

```



```

1035         {
1036             \__zrefclever_declare_default_transl:Vnn
1037             \l__zrefclever_dict_language_tl
1038             {#1} {##1}
1039         }
1040         {
1041             \msg_warning:nnn { zref-clever }
1042             { option-not-type-specific } {#1}
1043         }
1044     } ,
1045 }
1046 }
1047 \seq_map_inline:Nn
1048   \c__zrefclever_ref_options_possibly_type_specific_seq
1049   {
1050     \keys_define:nn { zref-clever / langsetup }
1051     {
1052       #1 .value_required:n = true ,
1053       #1 .code:n =
1054       {
1055         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1056         {
1057             \__zrefclever_declare_default_transl:Vnn
1058             \l__zrefclever_dict_language_tl
1059             {#1} {##1}
1060         }
1061         {
1062             \__zrefclever_declare_type_transl:VVnn
1063             \l__zrefclever_dict_language_tl
1064             \l__zrefclever_setup_type_tl
1065             {#1} {##1}
1066         }
1067     } ,
1068 }
1069 }
1070 \seq_map_inline:Nn
1071   \c__zrefclever_ref_options_necessarily_type_specific_seq
1072   {
1073     \keys_define:nn { zref-clever / langsetup }
1074     {
1075       #1 .value_required:n = true ,
1076       #1 .code:n =
1077       {
1078         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1079         {
1080             \msg_warning:nnn { zref-clever }
1081             { option-only-type-specific } {#1}
1082         }
1083         {
1084             \__zrefclever_declare_type_transl:VVnn
1085             \l__zrefclever_dict_language_tl
1086             \l__zrefclever_setup_type_tl
1087             {#1} {##1}
1088         }

```

```

1089         } ,
1090     }
1091 }

```

6 User interface

6.1 `\zcref`

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

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

```

1092 \NewDocumentCommand \zcref { s O { } m }
1093 { \zref@wrapper@babel \_zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for `\zcref`.)

`_zrefclever_zcref:nnnn` An intermediate internal function, which does the actual heavy lifting, and places `{<labels>}` as first argument, so that it can be protected by `\zref@wrapper@babel` in `\zcref`.

```
\_zrefclever_zcref:nnnn {<labels>} {<*>} {<options>}
```

```

1094 \cs_new_protected:Npn \_zrefclever_zcref:nnn #1#2#3
1095 {
1096     \group_begin:

```

Set options.

```
1097     \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```

1098     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1099     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

Ensure dictionary for reference language is loaded, if available. We cannot rely on `\keys_set:nn` for the task, since if the `lang` option is set for current, the actual language may have changed outside our control. `_zrefclever_provide_dictionary:x` does nothing if the dictionary is already loaded.

```
1100     \_zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Integration with `zref-check`.

```

1101     \bool_lazy_and:nnT
1102     { \l__zrefclever_zrefcheck_available_bool }
1103     { \l__zrefclever_zcref_with_check_bool }
1104     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1105     \bool_lazy_or:nnT
1106     { \l__zrefclever_typeset_sort_bool }
1107     { \l__zrefclever_typeset_range_bool }
1108     { \_zrefclever_sort_labels: }

```

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

```

1109     \group_begin:
1110     \l__zrefclever_ref_typeset_font_tl
1111     \__zrefclever_typeset_refs:
1112     \group_end:

```

Typeset note.

```

1113     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1114     {
1115         \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1116         \l_tmpa_tl
1117         \l__zrefclever_zcref_note_tl
1118     }

```

Integration with zref-check.

```

1119     \bool_lazy_and:nnT
1120     { \l__zrefclever_zrefcheck_available_bool }
1121     { \l__zrefclever_zcref_with_check_bool }
1122     {
1123         \zrefcheck_zcref_end_label_maybe:
1124         \zrefcheck_zcref_run_checks_on_labels:n
1125         { \l__zrefclever_zcref_labels_seq }
1126     }
1127     \group_end:
1128 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1129 \seq_new:N \l__zrefclever_zcref_labels_seq
1130 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for \l__zrefclever_zcref_labels_seq and \l__zrefclever_link_star_bool.)

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

`\zcpageref{*}[\langle options \rangle]{\langle labels \rangle}`

```

1131 \NewDocumentCommand \zcpageref { s O { } m }
1132 {
1133     \IfBooleanTF {#1}
1134     { \zcref*[#2, ref = page] {#3} }
1135     { \zcref [ #2, ref = page] {#3} }
1136 }

```

(End definition for \zcpageref.)

7 Sorting

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

<code>\l_zrefclever_label_type_a_tl</code>	Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.
<code>\l_zrefclever_label_type_b_tl</code>	
<code>\l_zrefclever_label_enclcnt_a_tl</code>	1137 <code>\tl_new:N \l__zrefclever_label_type_a_tl</code>
<code>\l_zrefclever_label_enclcnt_b_tl</code>	1138 <code>\tl_new:N \l__zrefclever_label_type_b_tl</code>
<code>\l_zrefclever_label_enclval_a_tl</code>	1139 <code>\tl_new:N \l__zrefclever_label_enclcnt_a_tl</code>
<code>\l_zrefclever_label_enclval_b_tl</code>	1140 <code>\tl_new:N \l__zrefclever_label_enclcnt_b_tl</code>
<code>\l_zrefclever_label_extdoc_a_tl</code>	1141 <code>\tl_new:N \l__zrefclever_label_enclval_a_tl</code>
<code>\l_zrefclever_label_extdoc_b_tl</code>	1142 <code>\tl_new:N \l__zrefclever_label_enclval_b_tl</code>
	1143 <code>\tl_new:N \l__zrefclever_label_extdoc_a_tl</code>
	1144 <code>\tl_new:N \l__zrefclever_label_extdoc_b_tl</code>
	(End definition for <code>\l_zrefclever_label_type_a_tl</code> and others.)
<code>\l_zrefclever_sort_decided_bool</code>	Auxiliary variable for <code>__zrefclever_sort_default_same_type:nn</code> , signals if the sorting between two labels has been decided or not.
	1145 <code>\bool_new:N \l__zrefclever_sort_decided_bool</code>
	(End definition for <code>\l_zrefclever_sort_decided_bool</code> .)
<code>\l_zrefclever_sort_prior_a_int</code>	Auxiliary variables for <code>__zrefclever_sort_default_different_types:nn</code> . Store the sort priority of the “current” and “next” labels.
<code>\l_zrefclever_sort_prior_b_int</code>	1146 <code>\int_new:N \l__zrefclever_sort_prior_a_int</code>
	1147 <code>\int_new:N \l__zrefclever_sort_prior_b_int</code>
	(End definition for <code>\l_zrefclever_sort_prior_a_int</code> and <code>\l_zrefclever_sort_prior_b_int</code> .)
<code>\l_zrefclever_label_types_seq</code>	Stores the order in which reference types appear in the label list supplied by the user in <code>\zceref</code> . This variable is populated by <code>__zrefclever_label_type_put_new_right:n</code> at the start of <code>__zrefclever_sort_labels:</code> . This order is required as a “last resort” sort criterion between the reference types, for use in <code>__zrefclever_sort_default_different_types:nn</code> .
	1148 <code>\seq_new:N \l__zrefclever_label_types_seq</code>
	(End definition for <code>\l_zrefclever_label_types_seq</code> .)
<code>__zrefclever_sort_labels:</code>	The main sorting function. It does not receive arguments, but it is expected to be run inside <code>__zrefclever_zceref:nnnn</code> where a number of environment variables are to be set appropriately. In particular, <code>\l_zrefclever_zceref_labels_seq</code> should contain the labels received as argument to <code>\zceref</code> , and the function performs its task by sorting this variable.
	1149 <code>\cs_new_protected:Npn __zrefclever_sort_labels:</code>
	1150 <code>{</code>

Store label types sequence.

```

1151 \seq_clear:N \l__zrefclever_label_types_seq
1152 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1153 {
1154   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1155   \__zrefclever_label_type_put_new_right:n
1156 }

```

Sort.

```

1157 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1158 {
1159   \zref@ifrefundefined {##1}
1160   {
1161     \zref@ifrefundefined {##2}
1162     {
1163       % Neither label is defined.
1164       \sort_return_same:
1165     }
1166     {
1167       % The second label is defined, but the first isn't, leave the
1168       % undefined first (to be more visible).
1169       \sort_return_same:
1170     }
1171   }
1172   {
1173     \zref@ifrefundefined {##2}
1174     {
1175       % The first label is defined, but the second isn't, bring the
1176       % second forward.
1177       \sort_return_swapped:
1178     }
1179     {
1180       % The interesting case: both labels are defined. References
1181       % to the "default" property or to the "page" are quite
1182       % different with regard to sorting, so we branch them here to
1183       % specialized functions.
1184       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1185       { \__zrefclever_sort_page:nn {##1} {##2} }
1186       { \__zrefclever_sort_default:nn {##1} {##2} }
1187     }
1188   }
1189 }
1190 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_label_type_put_new_right:n

Auxiliary function used to store the order in which reference types appear in the label list supplied by the user in `\zcref`. It is expected to be run inside `__zrefclever_sort_labels:`, and stores the types sequence in `\l__zrefclever_label_types_seq`. I have tried to handle the same task inside `\seq_sort:Nn` in `__zrefclever_sort_labels:` to spare mapping over `\l__zrefclever_zcref_labels_seq`, but it turned out it not to be easy to rely on the order the labels get processed at that point, since the variable is being sorted there. Besides, the mapping is simple, not a particularly expensive operation. Anyway, this keeps things clean.

```

    \_zrefclever_label_type_put_new_right:n {\label}}
1191 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1192 {
1193   \tl_set:Nx \l__zrefclever_label_type_a_tl
1194     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1195   \seq_if_in:NVF \l__zrefclever_label_types_seq
1196     \l__zrefclever_label_type_a_tl
1197   {
1198     \seq_put_right:NV \l__zrefclever_label_types_seq
1199       \l__zrefclever_label_type_a_tl
1200   }
1201 }

```

(End definition for _zrefclever_label_type_put_new_right:n.)

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

```

    \_zrefclever_sort_default:nn {\label a} {\label b}
1202 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
1203 {
1204   \tl_set:Nx \l__zrefclever_label_type_a_tl
1205     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1206   \tl_set:Nx \l__zrefclever_label_type_b_tl
1207     { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1208
1209   \bool_if:nTF
1210     {
1211       % The second label has a type, but the first doesn't, leave the
1212       % undefined first (to be more visible).
1213       \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1214       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1215     }
1216     { \sort_return_same: }
1217     {
1218       \bool_if:nTF
1219         {
1220           % The first label has a type, but the second doesn't, bring the
1221           % second forward.
1222           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1223           \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1224         }
1225         { \sort_return_swapped: }
1226         {
1227           \bool_if:nTF
1228             {
1229               % The interesting case: both labels have a type...
1230               ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1231               ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1232             }

```

```

1233     {
1234     \tl_if_eq:NNTF
1235     \l__zrefclever_label_type_a_tl
1236     \l__zrefclever_label_type_b_tl
1237     % ...and it's the same type.
1238     { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1239     % ...and they are different types.
1240     { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1241     }
1242     {
1243     % Neither label has a type. We can't do much of meaningful
1244     % here, but if it's the same counter, compare it.
1245     \exp_args:Nxx \tl_if_eq:nnTF
1246     { \zref@extractdefault {#1} {zc@counter} { } }
1247     { \zref@extractdefault {#2} {zc@counter} { } }
1248     {
1249     \int_compare:nNnTF
1250     { \zref@extractdefault {#1} {zc@cntval} { -1 } }
1251     >
1252     { \zref@extractdefault {#2} {zc@cntval} { -1 } }
1253     { \sort_return_swapped: }
1254     { \sort_return_same: }
1255     }
1256     { \sort_return_same: }
1257     }
1258     }
1259     }
1260     }

```

(End definition for `__zrefclever_sort_default:nn`.)

Variant not provided by the kernel, for use in `__zrefclever_sort_default_-same_type:nn`.

```

1261 \cs_generate_variant:Nn \tl_reverse_items:n { V }

```

```

\__zrefclever_sort_default_same_type:nn    \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
1262 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1263 {
1264     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1265     { \zref@extractdefault {#1} {zc@enclcnt} { \c_empty_tl } }
1266     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1267     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1268     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1269     { \zref@extractdefault {#2} {zc@enclcnt} { \c_empty_tl } }
1270     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1271     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1272     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1273     { \zref@extractdefault {#1} {zc@enclval} { \c_empty_tl } }
1274     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1275     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1276     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1277     { \zref@extractdefault {#2} {zc@enclval} { \c_empty_tl } }
1278     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1279     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1280     \tl_set:Nx \l__zrefclever_label_extdoc_a_tl

```

```

1281     { \zref@extractdefault {#1} { externaldocument } { \c_empty_tl } }
1282 \tl_set:Nx \l__zrefclever_label_extdoc_b_tl
1283     { \zref@extractdefault {#2} { externaldocument } { \c_empty_tl } }
1284
1285 \bool_set_false:N \l__zrefclever_sort_decided_bool
1286
1287 % First we check if there's any "external document" difference (coming
1288 % from 'zref-xr') and, if so, sort based on that.
1289 \tl_if_eq:NNF
1290     \l__zrefclever_label_extdoc_a_tl
1291     \l__zrefclever_label_extdoc_b_tl
1292 {
1293     \bool_if:nTF
1294     {
1295         \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1296         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1297     }
1298     {
1299         \bool_set_true:N \l__zrefclever_sort_decided_bool
1300         \sort_return_same:
1301     }
1302     {
1303         \bool_if:nTF
1304         {
1305             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1306             \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1307         }
1308         {
1309             \bool_set_true:N \l__zrefclever_sort_decided_bool
1310             \sort_return_swapped:
1311         }
1312         {
1313             \bool_set_true:N \l__zrefclever_sort_decided_bool
1314             % Two different "external documents": last resort, sort by the
1315             % document name itself.
1316             \str_compare:eNeTF
1317             { \l__zrefclever_label_extdoc_b_tl } <
1318             { \l__zrefclever_label_extdoc_a_tl }
1319             { \sort_return_swapped: }
1320             { \sort_return_same: }
1321         }
1322     }
1323 }
1324
1325 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1326 {
1327     \bool_if:nTF
1328     {
1329         % Both are empty: neither label has any (further) "enclosing
1330         % counters" (left).
1331         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1332         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1333     }
1334     {

```



```

1335 \exp_args:Nxx \tl_if_eq:nnTF
1336 { \zref@extractdefault {#1} { zc@counter } { } }
1337 { \zref@extractdefault {#2} { zc@counter } { } }
1338 {
1339   \bool_set_true:N \l__zrefclever_sort_decided_bool
1340   \int_compare:nNnTF
1341     { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1342     >
1343     { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1344     { \sort_return_swapped: }
1345     { \sort_return_same: }
1346   }
1347   {
1348     \msg_warning:nnnn { zref-clever }
1349     { counters-not-nested } {#1} {#2}
1350     \bool_set_true:N \l__zrefclever_sort_decided_bool
1351     \sort_return_same:
1352   }
1353 }
1354 {
1355   \bool_if:nTF
1356   {
1357     % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1358     \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1359   }
1360   {
1361     \int_zero:N \l_tmpb_int
1362     \tl_map_inline:Nn \l__zrefclever_label_enclcnt_b_tl
1363     {
1364       \int_incr:N \l_tmpb_int
1365       \exp_args:Nnx \tl_if_eq:nnT {##1}
1366       { \zref@extractdefault {#1} { zc@counter } { } }
1367       {
1368         \tl_map_break:n
1369         {
1370           \int_compare:nNnTF
1371             { \zref@extractdefault {#1} { zc@cntval } { } }
1372             >
1373             {
1374               \tl_item:Nn \l__zrefclever_label_enclval_b_tl
1375               { \l_tmpb_int }
1376             }
1377             { \sort_return_swapped: }
1378             { \sort_return_same: }
1379             \bool_set_true:N \l__zrefclever_sort_decided_bool
1380           }
1381         }
1382       }
1383     }
1384     \bool_if:NF \l__zrefclever_sort_decided_bool
1385     {
1386       \msg_warning:nnnn { zref-clever }
1387       { counters-not-nested } {#1} {#2}
1388       \bool_set_true:N \l__zrefclever_sort_decided_bool
1389       \sort_return_same:

```

```

1389     }
1390   }
1391   {
1392     \bool_if:nTF
1393     {
1394       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1395       \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1396     }
1397     {
1398       \int_zero:N \l_tmpa_int
1399       \tl_map_inline:Nn \l__zrefclever_label_enclcnt_a_tl
1400       {
1401         \int_incr:N \l_tmpa_int
1402         \exp_args:Nnx \tl_if_eq:nnT {##1}
1403         { \zref@extractdefault {#2} { zc@counter } { } }
1404         {
1405           \tl_map_break:n
1406           {
1407             \int_compare:nNnTF
1408             {
1409               \tl_item:Nn
1410               \l__zrefclever_label_enclval_a_tl
1411               { \l_tmpa_int }
1412             }
1413             <
1414             {
1415               \zref@extractdefault {#2}
1416               { zc@cntval } { }
1417             }
1418             { \sort_return_same: }
1419             { \sort_return_swapped: }
1420             \bool_set_true:N
1421             \l__zrefclever_sort_decided_bool
1422           }
1423         }
1424       }
1425       \bool_if:NF \l__zrefclever_sort_decided_bool
1426       {
1427         \msg_warning:nnnn { zref-clever }
1428         { counters-not-nested } {#1} {#2}
1429         \bool_set_true:N \l__zrefclever_sort_decided_bool
1430         \sort_return_same:
1431       }
1432     }
1433   {
1434     % Neither is empty: we can (possibly) compare the values
1435     % of the current enclosing counter in the loop, if they
1436     % are equal, we are still in the loop, if they are not, a
1437     % sorting decision can be made directly.
1438     \exp_args:Nxx \tl_if_eq:nnTF
1439     { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1440     { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1441     {
1442       \int_compare:nNnTF

```

```

1443 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1444 =
1445 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1446 {
1447   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1448     { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1449   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1450     { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1451   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1452     { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1453   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1454     { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1455 }
1456 {
1457   \bool_set_true:N \l__zrefclever_sort_decided_bool
1458   \int_compare:nNnTF
1459     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1460     >
1461     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1462     { \sort_return_swapped: }
1463     { \sort_return_same: }
1464 }
1465 }
1466 {
1467   \msg_warning:nnnn { zref-clever }
1468     { counters-not-nested } {#1} {#2}
1469   \bool_set_true:N \l__zrefclever_sort_decided_bool
1470   \sort_return_same:
1471 }
1472 }
1473 }
1474 }
1475 }
1476 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

```

__zrefclever_sort_default_different_types:nn
1477 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1478 {

```

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

```

1479   \int_zero:N \l__zrefclever_sort_prior_a_int
1480   \int_zero:N \l__zrefclever_sort_prior_b_int
1481   \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1482     {
1483       \tl_if_eq:nnTF {##2} {{othertypes}}
1484       {
1485         \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1486           { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1487         \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1488           { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }

```

```

1489     }
1490     {
1491         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1492         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1493         {
1494             \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1495             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1496         }
1497     }
1498 }

```

Then do the actual sorting.

```

1499 \bool_if:nTF
1500 {
1501     \int_compare_p:nNn
1502     { \l__zrefclever_sort_prior_a_int } <
1503     { \l__zrefclever_sort_prior_b_int }
1504 }
1505 { \sort_return_same: }
1506 {
1507     \bool_if:nTF
1508     {
1509         \int_compare_p:nNn
1510         { \l__zrefclever_sort_prior_a_int } >
1511         { \l__zrefclever_sort_prior_b_int }
1512     }
1513     { \sort_return_swapped: }
1514     {
1515         % Sort priorities are equal: the type that occurs first in
1516         % ‘labels’, as given by the user, is kept (or brought) forward.
1517         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1518         {
1519             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1520             { \seq_map_break:n { \sort_return_same: } }
1521             {
1522                 \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##1}
1523                 { \seq_map_break:n { \sort_return_swapped: } }
1524             }
1525         }
1526     }
1527 }
1528 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```

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

```

```

1529 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1530 {

```

```

1531 \int_compare:nNnTF
1532 { \zref@extractdefault {#1} { abspage } {-1} }
1533 >
1534 { \zref@extractdefault {#2} { abspage } {-1} }
1535 { \sort_return_swapped: }
1536 { \sort_return_same: }
1537 }

```

(End definition for `_zrefclever_sort_page:nn`.)

8 Typesetting

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

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

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

typesetting, we have two of these “queues”: `\l__zrefclever_typeset_queue_curr_tl` and `\l__zrefclever_typeset_queue_prev_tl`.

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

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

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

```
\l__zrefclever_typeset_labels_seq
\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool
1538 \seq_new:N \l__zrefclever_typeset_labels_seq
1539 \bool_new:N \l__zrefclever_typeset_last_bool
1540 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

```
\l__zrefclever_type_count_int
\l__zrefclever_label_count_int
1541 \int_new:N \l__zrefclever_type_count_int
1542 \int_new:N \l__zrefclever_label_count_int
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: main “queue” control and storage.

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_typeset_queue_curr_tl
\l__zrefclever_type_first_label_tl
\l__zrefclever_type_first_label_type_tl
1543 \tl_new:N \l__zrefclever_label_a_tl
1544 \tl_new:N \l__zrefclever_label_b_tl
```

```

1545 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1546 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1547 \tl_new:N \l__zrefclever_type_first_label_tl
1548 \tl_new:N \l__zrefclever_type_first_label_type_tl

```

(End definition for \l__zrefclever_label_a_tl and others.)

```

\l__zrefclever_type_name_tl
  \l__zrefclever_name_in_link_bool
  \l__zrefclever_name_format_tl
\l__zrefclever_name_format_fallback_tl

```

Auxiliary variables for __zrefclever_typeset_refs: type name handling.

```

1549 \tl_new:N \l__zrefclever_type_name_tl
1550 \bool_new:N \l__zrefclever_name_in_link_bool
1551 \tl_new:N \l__zrefclever_name_format_tl
1552 \tl_new:N \l__zrefclever_name_format_fallback_tl

```

(End definition for \l__zrefclever_type_name_tl and others.)

```

  \l__zrefclever_range_count_int
\l__zrefclever_range_same_count_int
  \l__zrefclever_range_beg_label_tl
\l__zrefclever_next_maybe_range_bool
  \l__zrefclever_next_is_same_bool

```

Auxiliary variables for __zrefclever_typeset_refs: range handling.

```

1553 \int_new:N \l__zrefclever_range_count_int
1554 \int_new:N \l__zrefclever_range_same_count_int
1555 \tl_new:N \l__zrefclever_range_beg_label_tl
1556 \bool_new:N \l__zrefclever_next_maybe_range_bool
1557 \bool_new:N \l__zrefclever_next_is_same_bool

```

(End definition for \l__zrefclever_range_count_int and others.)

```

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

```

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

```

1558 \tl_new:N \l__zrefclever_tpairsep_tl
1559 \tl_new:N \l__zrefclever_tlistsep_tl
1560 \tl_new:N \l__zrefclever_tlastsep_tl
1561 \tl_new:N \l__zrefclever_namesep_tl
1562 \tl_new:N \l__zrefclever_pairsep_tl
1563 \tl_new:N \l__zrefclever_listsep_tl
1564 \tl_new:N \l__zrefclever_lastsep_tl
1565 \tl_new:N \l__zrefclever_rangesep_tl
1566 \tl_new:N \l__zrefclever_refpre_out_tl
1567 \tl_new:N \l__zrefclever_refpos_out_tl
1568 \tl_new:N \l__zrefclever_refpre_in_tl
1569 \tl_new:N \l__zrefclever_refpos_in_tl
1570 \tl_new:N \l__zrefclever_namefont_tl
1571 \tl_new:N \l__zrefclever_reffont_out_tl
1572 \tl_new:N \l__zrefclever_reffont_in_tl

```

(End definition for \l__zrefclever_tpairsep_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

```

1573 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1574 {
1575   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1576   \l__zrefclever_zceref_labels_seq
1577   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1578   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1579   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

1580 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1581 \tl_clear:N \l__zrefclever_range_beg_label_tl
1582 \int_zero:N \l__zrefclever_label_count_int
1583 \int_zero:N \l__zrefclever_type_count_int
1584 \int_zero:N \l__zrefclever_range_count_int
1585 \int_zero:N \l__zrefclever_range_same_count_int
1586
1587 % Get type block options (not type-specific).
1588 \__zrefclever_get_ref_string:nN { tpairsep }
1589 \l__zrefclever_tpairsep_tl
1590 \__zrefclever_get_ref_string:nN { tlistsep }
1591 \l__zrefclever_tlistsep_tl
1592 \__zrefclever_get_ref_string:nN { tlastsep }
1593 \l__zrefclever_tlastsep_tl
1594
1595 % Process label stack.
1596 \bool_set_false:N \l__zrefclever_typeset_last_bool
1597 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1598 {
1599   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1600   \l__zrefclever_label_a_tl
1601   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1602   {
1603     \tl_clear:N \l__zrefclever_label_b_tl
1604     \bool_set_true:N \l__zrefclever_typeset_last_bool
1605   }
1606   {
1607     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1608     \l__zrefclever_label_b_tl
1609   }
1610
1611   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1612   {
1613     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1614     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1615   }
1616   {
1617     \tl_set:Nx \l__zrefclever_label_type_a_tl
1618     {
1619       \zref@extractdefault
1620       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1621     }
1622     \tl_set:Nx \l__zrefclever_label_type_b_tl
1623     {
1624       \zref@extractdefault
1625       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1626     }
1627   }
1628
1629   % First, we establish whether the "current label" (i.e. 'a') is the
1630   % last one of its type. This can happen because the "next label"
1631   % (i.e. 'b') is of a different type (or different definition status),
1632   % or because we are at the end of the list.
1633   \bool_if:NTF \l__zrefclever_typeset_last_bool

```



```

1634 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1635 {
1636   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1637   {
1638     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1639     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1640     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1641   }
1642   {
1643     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1644     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1645     {
1646       % Neither is undefined, we must check the types.
1647       \bool_if:nTF
1648       {
1649         % Both empty: same "type".
1650         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1651         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1652       }
1653       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1654       {
1655         \bool_if:nTF
1656         {
1657           % Neither empty: compare types.
1658           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1659           &&
1660           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1661         }
1662         {
1663           \tl_if_eq:NNTF
1664           \l__zrefclever_label_type_a_tl
1665           \l__zrefclever_label_type_b_tl
1666           {
1667             \bool_set_false:N
1668             \l__zrefclever_last_of_type_bool
1669           }
1670           {
1671             \bool_set_true:N
1672             \l__zrefclever_last_of_type_bool
1673           }
1674         }
1675         % One empty, the other not: different "types".
1676         {
1677           \bool_set_true:N
1678           \l__zrefclever_last_of_type_bool
1679         }
1680       }
1681     }
1682   }
1683 }
1684
1685 % Handle warnings in case of reference or type undefined.
1686 \zref@refused { \l__zrefclever_label_a_tl }
1687 \zref@ifrefundefined { \l__zrefclever_label_a_tl }

```

```

1688     {}
1689     {
1690         \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1691         {
1692             \msg_warning:nxx { zref-clever } { missing-type }
1693             { \l__zrefclever_label_a_tl }
1694         }
1695     }
1696
1697 % Get type-specific separators, refpre/pos and font options, once per
1698 % type.
1699 \int_compare:nNtT { \l__zrefclever_label_count_int } = { 0 }
1700 {
1701     \__zrefclever_get_ref_string:nN { namesep      }
1702     \l__zrefclever_namesep_tl
1703     \__zrefclever_get_ref_string:nN { rangesep    }
1704     \l__zrefclever_rangesep_tl
1705     \__zrefclever_get_ref_string:nN { pairsep     }
1706     \l__zrefclever_pairsep_tl
1707     \__zrefclever_get_ref_string:nN { listsep     }
1708     \l__zrefclever_listsep_tl
1709     \__zrefclever_get_ref_string:nN { lastsep     }
1710     \l__zrefclever_lastsep_tl
1711     \__zrefclever_get_ref_string:nN { refpre      }
1712     \l__zrefclever_refpre_out_tl
1713     \__zrefclever_get_ref_string:nN { refpos      }
1714     \l__zrefclever_refpos_out_tl
1715     \__zrefclever_get_ref_string:nN { refpre-in   }
1716     \l__zrefclever_refpre_in_tl
1717     \__zrefclever_get_ref_string:nN { refpos-in   }
1718     \l__zrefclever_refpos_in_tl
1719     \__zrefclever_get_ref_font:nN   { namefont    }
1720     \l__zrefclever_namefont_tl
1721     \__zrefclever_get_ref_font:nN   { reffont     }
1722     \l__zrefclever_reffont_out_tl
1723     \__zrefclever_get_ref_font:nN   { reffont-in  }
1724     \l__zrefclever_reffont_in_tl
1725 }
1726
1727 % Here we send this to a couple of auxiliary functions.
1728 \bool_if:NTF \l__zrefclever_last_of_type_bool
1729 % There exists no next label of the same type as the current.
1730 { \__zrefclever_typeset_refs_last_of_type: }
1731 % There exists a next label of the same type as the current.
1732 { \__zrefclever_typeset_refs_not_last_of_type: }
1733 }
1734 }

```

(End definition for `__zrefclever_typeset_refs:`.)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is *not* the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the “next” label and find something of a different “type” (loose here, maybe different defi-

nition status, maybe end of stack). So, though this is not very strict, `__zrefclever_`
`typeset_refs_last_of_type:` is more of a “wrapping up” function, and it is indeed
 the one which does the actual typesetting, while `__zrefclever_typeset_refs_not_`
`last_of_type:` is more of an “accumulation” function.

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

```

1735 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1736 {
1737   % Process the current label to the current queue.
1738   \int_case:nnF { \l__zrefclever_label_count_int }
1739   {
1740     % It is the last label of its type, but also the first one, and that's
1741     % what matters here: just store it.
1742     { 0 }
1743     {
1744       \tl_set:NV \l__zrefclever_type_first_label_tl
1745       \l__zrefclever_label_a_tl
1746       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1747       \l__zrefclever_label_type_a_tl
1748     }
1749
1750     % The last is the second: we have a pair (if not repeated).
1751     { 1 }
1752     {
1753       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1754       {
1755         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1756         {
1757           \exp_not:V \l__zrefclever_pairsep_tl
1758           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1759         }
1760       }
1761     }
1762   }
1763   % Last is third or more of its type: without repetition, we'd have the
1764   % last element on a list, but control for possible repetition.
1765   {
1766     \int_case:nnF { \l__zrefclever_range_count_int }
1767     {
1768       % There was no range going on.
1769       { 0 }
1770       {
1771         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1772         {
1773           \exp_not:V \l__zrefclever_lastsep_tl
1774           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1775         }
1776       }
1777       % Last in the range is also the second in it.
1778       { 1 }
1779       {
1780         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1781         {
1782           % We know 'range_beg_label' is not empty, since this is the
  
```

```

1783         % second element in the range, but the third or more in the
1784         % type list.
1785         \exp_not:V \l__zrefclever_listsep_tl
1786         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1787         \int_compare:nNnF
1788             { \l__zrefclever_range_same_count_int } = { 1 }
1789             {
1790                 \exp_not:V \l__zrefclever_lastsep_tl
1791                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1792             }
1793         }
1794     }
1795 }
1796 % Last in the range is third or more in it.
1797 {
1798     \int_case:nnF
1799     {
1800         \l__zrefclever_range_count_int -
1801         \l__zrefclever_range_same_count_int
1802     }
1803     {
1804         % Repetition, not a range.
1805         { 0 }
1806         {
1807             % If 'range_beg_label' is empty, it means it was also the
1808             % first of the type, and hence was already handled.
1809             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1810             {
1811                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1812                 {
1813                     \exp_not:V \l__zrefclever_lastsep_tl
1814                     \__zrefclever_get_ref:V
1815                     \l__zrefclever_range_beg_label_tl
1816                 }
1817             }
1818         }
1819         % A 'range', but with no skipped value, treat as list.
1820         { 1 }
1821         {
1822             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1823             {
1824                 % Ditto.
1825                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1826                 {
1827                     \exp_not:V \l__zrefclever_listsep_tl
1828                     \__zrefclever_get_ref:V
1829                     \l__zrefclever_range_beg_label_tl
1830                 }
1831                 \exp_not:V \l__zrefclever_lastsep_tl
1832                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1833             }
1834         }
1835     }
1836 }

```

```

1837 % An actual range.
1838 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1839 {
1840 % Ditto.
1841 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1842 {
1843 \exp_not:V \l__zrefclever_lastsep_tl
1844 \__zrefclever_get_ref:V
1845 \l__zrefclever_range_beg_label_tl
1846 }
1847 \exp_not:V \l__zrefclever_rangesep_tl
1848 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1849 }
1850 }
1851 }
1852 }
1853
1854 % Handle "range" option. The idea is simple: if the queue is not empty,
1855 % we replace it with the end of the range (or pair). We can still
1856 % retrieve the end of the range from 'label_a' since we know to be
1857 % processing the last label of its type at this point.
1858 \bool_if:NT \l__zrefclever_typeset_range_bool
1859 {
1860 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1861 {
1862 \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1863 { }
1864 {
1865 \msg_warning:nxx { zref-clever } { single-element-range }
1866 { \l__zrefclever_type_first_label_type_tl }
1867 }
1868 }
1869 {
1870 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1871 \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1872 { }
1873 {
1874 \__zrefclever_labels_in_sequence:nn
1875 { \l__zrefclever_type_first_label_tl }
1876 { \l__zrefclever_label_a_tl }
1877 }
1878 \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1879 {
1880 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1881 { \exp_not:V \l__zrefclever_pairsep_tl }
1882 { \exp_not:V \l__zrefclever_rangesep_tl }
1883 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1884 }
1885 }
1886 }
1887
1888 % Now that the type block is finished, we can add the name and the first
1889 % ref to the queue. Also, if "typeset" option is not "both", handle it
1890 % here as well.

```

```

1891 \__zrefclever_type_name_setup:
1892 \bool_if:nTF
1893 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1894 {
1895   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1896   { \__zrefclever_get_ref_first: }
1897 }
1898 {
1899   \bool_if:nTF
1900   { \l__zrefclever_typeset_ref_bool }
1901   {
1902     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1903     { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1904   }
1905   {
1906     \bool_if:nTF
1907     { \l__zrefclever_typeset_name_bool }
1908     {
1909       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1910       {
1911         \bool_if:NTF \l__zrefclever_name_in_link_bool
1912         {
1913           \exp_not:N \group_begin:
1914           \exp_not:V \l__zrefclever_namefont_tl
1915           % It's two '@s', but escaped for DocStrip.
1916           \exp_not:N \hyper@@link
1917           {
1918             \__zrefclever_extract_url:V
1919             \l__zrefclever_type_first_label_tl
1920           }
1921           {
1922             \zref@extractdefault
1923             { \l__zrefclever_type_first_label_tl }
1924             { anchor } {}
1925           }
1926           { \exp_not:V \l__zrefclever_type_name_tl }
1927           \exp_not:N \group_end:
1928         }
1929         {
1930           \exp_not:N \group_begin:
1931           \exp_not:V \l__zrefclever_namefont_tl
1932           \exp_not:V \l__zrefclever_type_name_tl
1933           \exp_not:N \group_end:
1934         }
1935       }
1936     }
1937   }
1938   % Logically, this case would correspond to "typeset=none", but
1939   % it should not occur, given that the options are set up to
1940   % typeset either "ref" or "name". Still, leave here a
1941   % sensible fallback, equal to the behavior of "both".
1942   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1943   { \__zrefclever_get_ref_first: }
1944 }

```

```

1945     }
1946   }
1947
1948   % Typeset the previous type, if there is one.
1949   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1950   {
1951     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1952     { \l__zrefclever_tlistsep_tl }
1953     \l__zrefclever_typeset_queue_prev_tl
1954   }
1955
1956   % Wrap up loop, or prepare for next iteration.
1957   \bool_if:NTF \l__zrefclever_typeset_last_bool
1958   {
1959     % We are finishing, typeset the current queue.
1960     \int_case:nnF { \l__zrefclever_type_count_int }
1961     {
1962       % Single type.
1963       { 0 }
1964       { \l__zrefclever_typeset_queue_curr_tl }
1965       % Pair of types.
1966       { 1 }
1967       {
1968         \l__zrefclever_tpairsep_tl
1969         \l__zrefclever_typeset_queue_curr_tl
1970       }
1971     }
1972     {
1973       % Last in list of types.
1974       \l__zrefclever_tlastsep_tl
1975       \l__zrefclever_typeset_queue_curr_tl
1976     }
1977   }
1978   {
1979     % There are further labels, set variables for next iteration.
1980     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1981       \l__zrefclever_typeset_queue_curr_tl
1982     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1983     \tl_clear:N \l__zrefclever_type_first_label_tl
1984     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1985     \tl_clear:N \l__zrefclever_range_beg_label_tl
1986     \int_zero:N \l__zrefclever_label_count_int
1987     \int_incr:N \l__zrefclever_type_count_int
1988     \int_zero:N \l__zrefclever_range_count_int
1989     \int_zero:N \l__zrefclever_range_same_count_int
1990   }
1991 }

```

(End definition for `__zrefclever_typeset_refs_last_of_type:`)

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

```

1992 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1993 {
1994   % Signal if next label may form a range with the current one (only

```

```

1995 % considered if compression is enabled in the first place).
1996 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1997 \bool_set_false:N \l__zrefclever_next_is_same_bool
1998 \bool_if:NT \l__zrefclever_typeset_compress_bool
1999 {
2000   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2001   { }
2002   {
2003     \__zrefclever_labels_in_sequence:nn
2004     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2005   }
2006 }
2007
2008 % Process the current label to the current queue.
2009 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2010 {
2011   % Current label is the first of its type (also not the last, but it
2012   % doesn't matter here): just store the label.
2013   \tl_set:NV \l__zrefclever_type_first_label_tl
2014   \l__zrefclever_label_a_tl
2015   \tl_set:NV \l__zrefclever_type_first_label_type_tl
2016   \l__zrefclever_label_type_a_tl
2017
2018   % If the next label may be part of a range, we set 'range_beg_label'
2019   % to "empty" (we deal with it as the "first", and must do it there, to
2020   % handle hyperlinking), but also step the range counters.
2021   \bool_if:NT \l__zrefclever_next_maybe_range_bool
2022   {
2023     \tl_clear:N \l__zrefclever_range_beg_label_tl
2024     \int_incr:N \l__zrefclever_range_count_int
2025     \bool_if:NT \l__zrefclever_next_is_same_bool
2026     { \int_incr:N \l__zrefclever_range_same_count_int }
2027   }
2028 }
2029 {
2030   % Current label is neither the first (nor the last) of its type.
2031   \bool_if:NnTF \l__zrefclever_next_maybe_range_bool
2032   {
2033     % Starting, or continuing a range.
2034     \int_compare:nNnTF
2035     { \l__zrefclever_range_count_int } = { 0 }
2036     {
2037       % There was no range going, we are starting one.
2038       \tl_set:NV \l__zrefclever_range_beg_label_tl
2039       \l__zrefclever_label_a_tl
2040       \int_incr:N \l__zrefclever_range_count_int
2041       \bool_if:NT \l__zrefclever_next_is_same_bool
2042       { \int_incr:N \l__zrefclever_range_same_count_int }
2043     }
2044     {
2045       % Second or more in the range, but not the last.
2046       \int_incr:N \l__zrefclever_range_count_int
2047       \bool_if:NT \l__zrefclever_next_is_same_bool
2048       { \int_incr:N \l__zrefclever_range_same_count_int }

```



```

2049     }
2050   }
2051   {
2052     % Next element is not in sequence: there was no range, or we are
2053     % closing one.
2054     \int_case:nnF { \l__zrefclever_range_count_int }
2055     {
2056       % There was no range going on.
2057       { 0 }
2058       {
2059         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2060         {
2061           \exp_not:V \l__zrefclever_listsep_tl
2062           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2063         }
2064       }
2065       % Last is second in the range: if 'range_same_count' is also
2066       % '1', it's a repetition (drop it), otherwise, it's a "pair
2067       % within a list", treat as list.
2068       { 1 }
2069       {
2070         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2071         {
2072           \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2073           {
2074             \exp_not:V \l__zrefclever_listsep_tl
2075             \__zrefclever_get_ref:V
2076             \l__zrefclever_range_beg_label_tl
2077           }
2078           \int_compare:nNnF
2079           { \l__zrefclever_range_same_count_int } = { 1 }
2080           {
2081             \exp_not:V \l__zrefclever_listsep_tl
2082             \__zrefclever_get_ref:V
2083             \l__zrefclever_label_a_tl
2084           }
2085         }
2086       }
2087     }
2088   {
2089     % Last is third or more in the range: if 'range_count' and
2090     % 'range_same_count' are the same, its a repetition (drop it),
2091     % if they differ by '1', its a list, if they differ by more,
2092     % it is a real range.
2093     \int_case:nnF
2094     {
2095       \l__zrefclever_range_count_int -
2096       \l__zrefclever_range_same_count_int
2097     }
2098     {
2099       { 0 }
2100       {
2101         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2102         {

```

```

2103         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2104         {
2105             \exp_not:V \l__zrefclever_listsep_tl
2106             \__zrefclever_get_ref:V
2107             \l__zrefclever_range_beg_label_tl
2108         }
2109     }
2110 }
2111 { 1 }
2112 {
2113     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2114     {
2115         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2116         {
2117             \exp_not:V \l__zrefclever_listsep_tl
2118             \__zrefclever_get_ref:V
2119             \l__zrefclever_range_beg_label_tl
2120         }
2121         \exp_not:V \l__zrefclever_listsep_tl
2122         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2123     }
2124 }
2125 }
2126 {
2127     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2128     {
2129         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2130         {
2131             \exp_not:V \l__zrefclever_listsep_tl
2132             \__zrefclever_get_ref:V
2133             \l__zrefclever_range_beg_label_tl
2134         }
2135         \exp_not:V \l__zrefclever_rangesep_tl
2136         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2137     }
2138 }
2139 }
2140 % Reset counters.
2141 \int_zero:N \l__zrefclever_range_count_int
2142 \int_zero:N \l__zrefclever_range_same_count_int
2143 }
2144 }
2145 % Step label counter for next iteration.
2146 \int_incr:N \l__zrefclever_label_count_int
2147 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`.)

Aux functions

`__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:n` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though,

they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` get called, as they must, in the context of `x` type expansions. But we don't want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can't be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```

2148 \cs_new_protected:Npn \__zrefclever_ref_default:
2149   { \zref@default }
2150 \cs_new_protected:Npn \__zrefclever_name_default:
2151   { \zref@default }

```

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

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

```

\__zrefclever_get_ref:n {<label>}

2152 \cs_new:Npn \__zrefclever_get_ref:n #1
2153   {
2154     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2155     {
2156       \bool_if:nTF
2157         {
2158           \l__zrefclever_use_hyperref_bool &&
2159           ! \l__zrefclever_link_star_bool
2160         }
2161         {
2162           \exp_not:N \group_begin:
2163           \exp_not:V \l__zrefclever_reffont_out_tl
2164           \exp_not:V \l__zrefclever_refpre_out_tl
2165           \exp_not:N \group_begin:
2166           \exp_not:V \l__zrefclever_reffont_in_tl
2167           % It's two '@s', but escaped for DocStrip.
2168           \exp_not:N \hyper@@link
2169             { \__zrefclever_extract_url:n {#1} }
2170             { \zref@extractdefault {#1} { anchor } { } }

```

```

2171         {
2172             \exp_not:V \l__zrefclever_refpre_in_tl
2173             \zref@extractdefault {#1}
2174             { \l__zrefclever_ref_property_tl } { }
2175             \exp_not:V \l__zrefclever_refpos_in_tl
2176         }
2177         \exp_not:N \group_end:
2178         \exp_not:V \l__zrefclever_refpos_out_tl
2179         \exp_not:N \group_end:
2180     }
2181     {
2182         \exp_not:N \group_begin:
2183         \exp_not:V \l__zrefclever_reffont_out_tl
2184         \exp_not:V \l__zrefclever_refpre_out_tl
2185         \exp_not:N \group_begin:
2186         \exp_not:V \l__zrefclever_reffont_in_tl
2187         \exp_not:V \l__zrefclever_refpre_in_tl
2188         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2189         \exp_not:V \l__zrefclever_refpos_in_tl
2190         \exp_not:N \group_end:
2191         \exp_not:V \l__zrefclever_refpos_out_tl
2192         \exp_not:N \group_end:
2193     }
2194 }
2195 { \__zrefclever_ref_default: }
2196 }
2197 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

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

```

2198 \cs_new:Npn \__zrefclever_get_ref_first:
2199 {
2200     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2201     { \__zrefclever_ref_default: }
2202     {
2203         \bool_if:NTF \l__zrefclever_name_in_link_bool
2204         {
2205             \zref@ifrefcontainsprop
2206             { \l__zrefclever_type_first_label_tl }
2207             { \l__zrefclever_ref_property_tl }
2208             {
2209                 % It's two '@s', but escaped for DocStrip.
2210                 \exp_not:N \hyper@@link
2211                 {
2212                     \__zrefclever_extract_url:V
2213                     \l__zrefclever_type_first_label_tl

```

```

2214 }
2215 {
2216   \zref@extractdefault
2217   { \l__zrefclever_type_first_label_tl }
2218   { anchor } { }
2219 }
2220 {
2221   \exp_not:N \group_begin:
2222   \exp_not:V \l__zrefclever_namefont_tl
2223   \exp_not:V \l__zrefclever_type_name_tl
2224   \exp_not:N \group_end:
2225   \exp_not:V \l__zrefclever_namesep_tl
2226   \exp_not:N \group_begin:
2227   \exp_not:V \l__zrefclever_reffont_out_tl
2228   \exp_not:V \l__zrefclever_refpre_out_tl
2229   \exp_not:N \group_begin:
2230   \exp_not:V \l__zrefclever_reffont_in_tl
2231   \exp_not:V \l__zrefclever_refpre_in_tl
2232   \zref@extractdefault
2233   { \l__zrefclever_type_first_label_tl }
2234   { \l__zrefclever_ref_property_tl } { }
2235   \exp_not:V \l__zrefclever_refpos_in_tl
2236   \exp_not:N \group_end:
2237   % hyperlink makes it's own group, we'd like to close the
2238   % 'refpre-out' group after 'refpos-out', but... we close
2239   % it here, and give the trailing 'refpos-out' its own
2240   % group. This will result that formatting given to
2241   % 'refpre-out' will not reach 'refpos-out', but I see no
2242   % alternative, and this has to be handled specially.
2243   \exp_not:N \group_end:
2244 }
2245 \exp_not:N \group_begin:
2246 % Ditto: special treatment.
2247 \exp_not:V \l__zrefclever_reffont_out_tl
2248 \exp_not:V \l__zrefclever_refpos_out_tl
2249 \exp_not:N \group_end:
2250 }
2251 {
2252   \exp_not:N \group_begin:
2253   \exp_not:V \l__zrefclever_namefont_tl
2254   \exp_not:V \l__zrefclever_type_name_tl
2255   \exp_not:N \group_end:
2256   \exp_not:V \l__zrefclever_namesep_tl
2257   \__zrefclever_ref_default:
2258 }
2259 }
2260 {
2261   \tl_if_empty:NTF \l__zrefclever_type_name_tl
2262   {
2263     \__zrefclever_name_default:
2264     \exp_not:V \l__zrefclever_namesep_tl
2265   }
2266   {
2267     \exp_not:N \group_begin:

```

```

2268         \exp_not:V \l__zrefclever_namefont_tl
2269         \exp_not:V \l__zrefclever_type_name_tl
2270         \exp_not:N \group_end:
2271         \exp_not:V \l__zrefclever_namesep_tl
2272     }
2273 \zref@ifrefcontainsprop
2274 { \l__zrefclever_type_first_label_tl }
2275 { \l__zrefclever_ref_property_tl }
2276 {
2277     \bool_if:nTF
2278     {
2279         \l__zrefclever_use_hyperref_bool &&
2280         ! \l__zrefclever_link_star_bool
2281     }
2282     {
2283         \exp_not:N \group_begin:
2284         \exp_not:V \l__zrefclever_reffont_out_tl
2285         \exp_not:V \l__zrefclever_refpre_out_tl
2286         \exp_not:N \group_begin:
2287         \exp_not:V \l__zrefclever_reffont_in_tl
2288         % It's two '@s', but escaped for DocStrip.
2289         \exp_not:N \hyper@@link
2290         {
2291             \__zrefclever_extract_url:V
2292             \l__zrefclever_type_first_label_tl
2293         }
2294         {
2295             \zref@extractdefault
2296             { \l__zrefclever_type_first_label_tl }
2297             { anchor } { }
2298         }
2299         {
2300             \exp_not:V \l__zrefclever_refpre_in_tl
2301             \zref@extractdefault
2302             { \l__zrefclever_type_first_label_tl }
2303             { \l__zrefclever_ref_property_tl } { }
2304             \exp_not:V \l__zrefclever_refpos_in_tl
2305         }
2306         \exp_not:N \group_end:
2307         \exp_not:V \l__zrefclever_refpos_out_tl
2308         \exp_not:N \group_end:
2309     }
2310     {
2311         \exp_not:N \group_begin:
2312         \exp_not:V \l__zrefclever_reffont_out_tl
2313         \exp_not:V \l__zrefclever_refpre_out_tl
2314         \exp_not:N \group_begin:
2315         \exp_not:V \l__zrefclever_reffont_in_tl
2316         \exp_not:V \l__zrefclever_refpre_in_tl
2317         \zref@extractdefault
2318         { \l__zrefclever_type_first_label_tl }
2319         { \l__zrefclever_ref_property_tl } { }
2320         \exp_not:V \l__zrefclever_refpos_in_tl
2321         \exp_not:N \group_end:

```

```

2322         \exp_not:V \l__zrefclever_refpos_out_tl
2323         \exp_not:N \group_end:
2324     }
2325 }
2326 { \__zrefclever_ref_default: }
2327 }
2328 }
2329 }

```

(End definition for __zrefclever_get_ref_first:.)

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

```

2330 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2331 {
2332     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2333     { \tl_clear:N \l__zrefclever_type_name_tl }
2334     {
2335         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2336         { \tl_clear:N \l__zrefclever_type_name_tl }
2337         {
2338             % Determine whether we should use capitalization, abbreviation,
2339             % and plural.
2340             \bool_lazy_or:nnTF
2341             { \l__zrefclever_capitalize_bool }
2342             {
2343                 \l__zrefclever_capitalize_first_bool &&
2344                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2345             }
2346             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2347             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2348             % If the queue is empty, we have a singular, otherwise, plural.
2349             \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2350             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2351             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2352             \bool_lazy_and:nnTF
2353             { \l__zrefclever_abbrev_bool }
2354             {
2355                 ! \int_compare_p:nNn
2356                 { \l__zrefclever_type_count_int } = { 0 } ||
2357                 ! \l__zrefclever_noabbrev_first_bool
2358             }
2359             {
2360                 \tl_set:NV \l__zrefclever_name_format_fallback_tl

```

```

2361         \l__zrefclever_name_format_tl
2362         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2363     }
2364     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2365
2366 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2367 {
2368     \prop_get:cVNF
2369     {
2370         l__zrefclever_type_
2371         \l__zrefclever_type_first_label_type_tl _options_prop
2372     }
2373     \l__zrefclever_name_format_tl
2374     \l__zrefclever_type_name_tl
2375     {
2376         \__zrefclever_get_type_transl:xxxNF
2377         { \l__zrefclever_ref_language_tl }
2378         { \l__zrefclever_type_first_label_type_tl }
2379         { \l__zrefclever_name_format_tl }
2380         \l__zrefclever_type_name_tl
2381         {
2382             \tl_clear:N \l__zrefclever_type_name_tl
2383             \msg_warning:nxx { zref-clever } { missing-name }
2384             { \l__zrefclever_type_first_label_type_tl }
2385         }
2386     }
2387 }
2388 {
2389     \prop_get:cVNF
2390     {
2391         l__zrefclever_type_
2392         \l__zrefclever_type_first_label_type_tl _options_prop
2393     }
2394     \l__zrefclever_name_format_tl
2395     \l__zrefclever_type_name_tl
2396     {
2397         \prop_get:cVNF
2398         {
2399             l__zrefclever_type_
2400             \l__zrefclever_type_first_label_type_tl _options_prop
2401         }
2402         \l__zrefclever_name_format_fallback_tl
2403         \l__zrefclever_type_name_tl
2404         {
2405             \__zrefclever_get_type_transl:xxxNF
2406             { \l__zrefclever_ref_language_tl }
2407             { \l__zrefclever_type_first_label_type_tl }
2408             { \l__zrefclever_name_format_tl }
2409             \l__zrefclever_type_name_tl
2410             {
2411                 \__zrefclever_get_type_transl:xxxNF
2412                 { \l__zrefclever_ref_language_tl }
2413                 { \l__zrefclever_type_first_label_type_tl }
2414                 { \l__zrefclever_name_format_fallback_tl }

```



```

2415         \l__zrefclever_type_name_tl
2416     {
2417         \tl_clear:N \l__zrefclever_type_name_tl
2418         \msg_warning:nmx { zref-clever }
2419         { missing-name }
2420         { \l__zrefclever_type_first_label_type_tl }
2421     }
2422 }
2423 }
2424 }
2425 }
2426 }
2427 }
2428
2429 % Signal whether the type name is to be included in the hyperlink or not.
2430 \bool_lazy_any:nTF
2431 {
2432     { ! \l__zrefclever_use_hyperref_bool }
2433     { \l__zrefclever_link_star_bool }
2434     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2435     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2436 }
2437 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2438 {
2439     \bool_lazy_any:nTF
2440     {
2441         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2442         {
2443             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2444             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2445         }
2446         {
2447             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2448             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2449             \l__zrefclever_typeset_last_bool &&
2450             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2451         }
2452     }
2453     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2454     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2455 }
2456 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_extract_url:n A convenience auxiliary function for extraction of the url / urluse property, provided by the zref-xr module.

```

2457 \cs_new:Npn \__zrefclever_extract_url:n #1
2458 {
2459     \zref@ifpropundefined { urluse }
2460     { \zref@extractdefault {#1} { url } { \c_empty_tl } }
2461     {
2462         \zref@ifrefcontainsprop {#1} { urluse }
2463         { \zref@extractdefault {#1} { urluse } { \c_empty_tl } }

```

```

2464         { \zref@extractdefault {#1} { url } { \c_empty_tl } }
2465     }
2466 }
2467 \cs_generate_variant:Nn \__zrefclever_extract_url:n { V }

```

(End definition for __zrefclever_extract_url:n.)

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

```

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

2468 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2469 {
2470     \tl_set:Nx \l__zrefclever_label_extdoc_a_tl
2471     { \zref@extractdefault {#1} { externaldocument } { \c_empty_tl } }
2472     \tl_set:Nx \l__zrefclever_label_extdoc_b_tl
2473     { \zref@extractdefault {#2} { externaldocument } { \c_empty_tl } }
2474
2475     \tl_if_eq:NNT
2476     \l__zrefclever_label_extdoc_a_tl
2477     \l__zrefclever_label_extdoc_b_tl
2478     {
2479         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2480         {
2481             \exp_args:Nxx \tl_if_eq:nnT
2482             { \zref@extractdefault {#1} { zc@pgfmt } { } }
2483             { \zref@extractdefault {#2} { zc@pgfmt } { } }
2484             {
2485                 \int_compare:nNnTF
2486                 { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2487                 =
2488                 { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2489                 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2490                 {
2491                     \int_compare:nNnT
2492                     { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2493                     =
2494                     { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2495                     {
2496                         \bool_set_true:N
2497                         \l__zrefclever_next_maybe_range_bool
2498                         \bool_set_true:N
2499                         \l__zrefclever_next_is_same_bool
2500                     }
2501                 }
2502             }
2503         }
2504     }
2505     \exp_args:Nxx \tl_if_eq:nnT

```

```

2506 { \zref@extractdefault {#1} { zc@counter } { } }
2507 { \zref@extractdefault {#2} { zc@counter } { } }
2508 {
2509   \exp_args:Nxx \tl_if_eq:nnT
2510   { \zref@extractdefault {#1} { zc@enclval } { } }
2511   { \zref@extractdefault {#2} { zc@enclval } { } }
2512   {
2513     \int_compare:nNnTF
2514     { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2515     =
2516     { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2517     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2518     {
2519       \int_compare:nNnT
2520       { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2521       =
2522       { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2523       {
2524         \bool_set_true:N
2525         \l__zrefclever_next_maybe_range_bool
2526         \bool_set_true:N
2527         \l__zrefclever_next_is_same_bool
2528       }
2529     }
2530   }
2531 }
2532 }
2533 }
2534 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

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

```

\__zrefclever_get_ref_string:nN      \__zrefclever_get_ref_string:nN {<option>} {<tl variable>}
2535 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2536 {
2537   % First attempt: general options.
2538   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2539   {
2540     % If not found, try type specific options.
2541     \bool_lazy_all:nTF
2542     {

```

```

2543 { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2544 {
2545   \prop_if_exist_p:c
2546   {
2547     l__zrefclever_type_
2548     \l__zrefclever_label_type_a_tl _options_prop
2549   }
2550 }
2551 {
2552   \prop_if_in_p:cn
2553   {
2554     l__zrefclever_type_
2555     \l__zrefclever_label_type_a_tl _options_prop
2556   }
2557   {#1}
2558 }
2559 }
2560 {
2561   \prop_get:cnN
2562   {
2563     l__zrefclever_type_
2564     \l__zrefclever_label_type_a_tl _options_prop
2565   }
2566   {#1} #2
2567 }
2568 {
2569   % If not found, try type specific translations.
2570   \__zrefclever_get_type_transl:xxnNF
2571   { \l__zrefclever_ref_language_tl }
2572   { \l__zrefclever_label_type_a_tl }
2573   {#1} #2
2574   {
2575     % If not found, try default translations.
2576     \__zrefclever_get_default_transl:xnNF
2577     { \l__zrefclever_ref_language_tl }
2578     {#1} #2
2579     {
2580       % If not found, try fallback.
2581       \__zrefclever_get_fallback_transl:nNF {#1} #2
2582       {
2583         \tl_clear:N #2
2584         \msg_warning:nnn { zref-clever }
2585           { missing-string } {#1}
2586       }
2587     }
2588   }
2589 }
2590 }
2591 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2592 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2

```

```

2593 {
2594   % First attempt: general options.
2595   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2596   {
2597     % If not found, try type specific options.
2598     \bool_lazy_and:nnTF
2599     { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2600     {
2601       \prop_if_exist_p:c
2602       {
2603         l__zrefclever_type_
2604         \l__zrefclever_label_type_a_tl _options_prop
2605       }
2606     }
2607     {
2608       \prop_get:cnNF
2609       {
2610         l__zrefclever_type_
2611         \l__zrefclever_label_type_a_tl _options_prop
2612       }
2613       {#1} #2
2614       { \tl_clear:N #2 }
2615     }
2616     { \tl_clear:N #2 }
2617   }
2618 }

```

(End definition for `__zrefclever_get_ref_font:nN`.)

9 Compatibility

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 `\footnote`

I’d love not to have to tamper with the `\footnote`’s machinery. . . . However, it is too basic a feature not to work out-of-the-box and, unfortunately, it neither uses `\refstepcounter` nor sets `\@currentcounter`. So there’s really not much to do here except trust in the new hook management system.

I have made a feature request though, for having `\@currentcounter` recorded there too: <https://github.com/latex3/latex2e/issues/687>.

CHECK See if the FR has been implemented or not and, if so, remove this.

```

2619 \tl_new:N \l__zrefclever_footnote_type_tl
2620 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
2621 \AddToHook { env / minipage / begin }
2622 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }

```

```

2623 \AddToHook { cmd / @makefntext / before }
2624 {
2625     \exp_args:Nx \zcsetup
2626     { currentcounter = \l__zrefclever_footnote_type_tl }
2627 }

```

9.2 \appendix

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

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

```

2628 \AddToHook { cmd / appendix / before }
2629 {
2630     \zcsetup
2631     {
2632         countertype =
2633         {
2634             chapter      = appendix ,
2635             section      = appendix ,
2636             subsection   = appendix ,
2637             subsubsection = appendix ,
2638         }
2639     }
2640 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmdhooks` (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (`##`) the patch to add the hook, if it needs to be done with the `\scantokens` method, may fail noisily (see <https://tex.stackexchange.com/q/617905>, thanks Phelype Oleinik). The 2021-11-15 kernel release should already handle this gracefully. In the meantime, given we cannot really expect to know what `\appendix` may contain in general, since it potentially gets redefined in quite a number of classes and packages, a user facing workaround may be needed in case of trouble. Phelype Oleinik recommends activating/providing the generic hook in question, so that `ltxcmdhooks` considers the patch as already done, and do the patch ourselves with `etoolbox` (<https://tex.stackexchange.com/a/617998>). Like so:

```

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

```

9.3 appendix package

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

```

2641 \AddToHook { begindocument }
2642 {
2643   \@ifpackageloaded { appendix }
2644   {
2645     \AddToHook { env / appendices / begin }
2646     {
2647       \zcsetup
2648       {
2649         countertype =
2650         {
2651           chapter      = appendix ,
2652           section      = appendix ,
2653           subsection   = appendix ,
2654           subsubsection = appendix ,
2655         }
2656       }
2657     }
2658     \AddToHook { env / subappendices / begin }
2659     {
2660       \zcsetup
2661       {
2662         countertype =
2663         {
2664           chapter      = subappendix ,
2665           section      = subappendix ,
2666           subsection   = subappendix ,
2667           subsubsection = subappendix ,
2668         }
2669       }
2670     }
2671     \msg_info:nnn { zref-clever } { compat-package } { appendix }
2672   }
2673   {}
2674 }

```

9.4 listings package

```

2675 \AddToHook { begindocument }
2676 {
2677   \@ifpackageloaded { listings }

```

```

2678 {
2679   \zcsetup
2680   {
2681     countertype =
2682     {
2683       lstlisting = listing ,
2684       lstnumber = line ,
2685     } ,
2686     counterresetby = { lstnumber = lstlisting } ,
2687   }
2688   \lst@AddToHook { Init }
2689   {

```

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

```

2690       \tl_if_empty:NF \lst@label
2691       { \zlabel { \lst@label } }

```

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

```

2692       \zcsetup { currentcounter = lstnumber }
2693     }
2694     \msg_info:nnn { zref-clever } { compat-package } { listings }
2695   }
2696   {}
2697 }

```

9.5 enumitem package

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

The only real reason to “renew” `enumerate` itself is to change `{\max-depth}`. `\renewlist` *hard-codes* `max-depth` in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But `\renewlist` also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from `zref-clever`’s perspective. Since the first four are defined by the kernel and already setup for `zref-clever` by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```

2698 \AddToHook { begindocument }
2699 {
2700   \@ifpackageloaded { enumitem }
2701   {
2702     \int_set:Nn \l_tmpa_int { 5 }
2703     \bool_while_do:nn

```



```

2704     {
2705       \cs_if_exist_p:c
2706       { c@ enum \int_to_roman:n { \l_tmpa_int } }
2707     }
2708     {
2709       \exp_args:Nx \zcsetup
2710       {
2711         counterresetby =
2712         {
2713           enum \int_to_roman:n { \l_tmpa_int } =
2714           enum \int_to_roman:n { \l_tmpa_int - 1 }
2715         } ,
2716         countertype =
2717         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
2718       }
2719       \int_incr:N \l_tmpa_int
2720     }
2721     \int_compare:nNnT { \l_tmpa_int } > { 5 }
2722     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
2723   }
2724   {}
2725 }
2726 </package>

```

10 Dictionaries

10.1 English

```

2727 <package>\zcDeclareLanguage { english }
2728 <package>\zcDeclareLanguageAlias { american } { english }
2729 <package>\zcDeclareLanguageAlias { australian } { english }
2730 <package>\zcDeclareLanguageAlias { british } { english }
2731 <package>\zcDeclareLanguageAlias { canadian } { english }
2732 <package>\zcDeclareLanguageAlias { newzealand } { english }
2733 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2734 <package>\zcDeclareLanguageAlias { USenglish } { english }
2735 <*dict-english>
2736 namesep = {\nobreakspace} ,
2737 pairsep = {\~and\nobreakspace} ,
2738 listsep = {\~,~} ,
2739 lastsep = {\~and\nobreakspace} ,
2740 tpairsep = {\~and\nobreakspace} ,
2741 tlistsep = {\~,~} ,
2742 tlastsep = {\~,~and\nobreakspace} ,
2743 notesep = {\~} ,
2744 rangesep = {\~to\nobreakspace} ,
2745
2746 type = part ,
2747 Name-sg = Part ,
2748 name-sg = part ,
2749 Name-pl = Parts ,
2750 name-pl = parts ,

```

```

2751
2752 type = chapter ,
2753     Name-sg = Chapter ,
2754     name-sg = chapter ,
2755     Name-pl = Chapters ,
2756     name-pl = chapters ,
2757
2758 type = section ,
2759     Name-sg = Section ,
2760     name-sg = section ,
2761     Name-pl = Sections ,
2762     name-pl = sections ,
2763
2764 type = paragraph ,
2765     Name-sg = Paragraph ,
2766     name-sg = paragraph ,
2767     Name-pl = Paragraphs ,
2768     name-pl = paragraphs ,
2769     Name-sg-ab = Par. ,
2770     name-sg-ab = par. ,
2771     Name-pl-ab = Par. ,
2772     name-pl-ab = par. ,
2773
2774 type = appendix ,
2775     Name-sg = Appendix ,
2776     name-sg = appendix ,
2777     Name-pl = Appendices ,
2778     name-pl = appendices ,
2779
2780 type = subappendix ,
2781     Name-sg = Appendix ,
2782     name-sg = appendix ,
2783     Name-pl = Appendices ,
2784     name-pl = appendices ,
2785
2786 type = page ,
2787     Name-sg = Page ,
2788     name-sg = page ,
2789     Name-pl = Pages ,
2790     name-pl = pages ,
2791     name-sg-ab = p. ,
2792     name-pl-ab = pp. ,
2793
2794 type = line ,
2795     Name-sg = Line ,
2796     name-sg = line ,
2797     Name-pl = Lines ,
2798     name-pl = lines ,
2799
2800 type = figure ,
2801     Name-sg = Figure ,
2802     name-sg = figure ,
2803     Name-pl = Figures ,
2804     name-pl = figures ,

```

```

2805 Name-sg-ab = Fig. ,
2806 name-sg-ab = fig. ,
2807 Name-pl-ab = Figs. ,
2808 name-pl-ab = figs. ,
2809
2810 type = table ,
2811 Name-sg = Table ,
2812 name-sg = table ,
2813 Name-pl = Tables ,
2814 name-pl = tables ,
2815
2816 type = item ,
2817 Name-sg = Item ,
2818 name-sg = item ,
2819 Name-pl = Items ,
2820 name-pl = items ,
2821
2822 type = footnote ,
2823 Name-sg = Footnote ,
2824 name-sg = footnote ,
2825 Name-pl = Footnotes ,
2826 name-pl = footnotes ,
2827
2828 type = note ,
2829 Name-sg = Note ,
2830 name-sg = note ,
2831 Name-pl = Notes ,
2832 name-pl = notes ,
2833
2834 type = equation ,
2835 Name-sg = Equation ,
2836 name-sg = equation ,
2837 Name-pl = Equations ,
2838 name-pl = equations ,
2839 Name-sg-ab = Eq. ,
2840 name-sg-ab = eq. ,
2841 Name-pl-ab = Eqs. ,
2842 name-pl-ab = eqs. ,
2843 refpre-in = {} ,
2844 refpos-in = {} ,
2845
2846 type = theorem ,
2847 Name-sg = Theorem ,
2848 name-sg = theorem ,
2849 Name-pl = Theorems ,
2850 name-pl = theorems ,
2851
2852 type = lemma ,
2853 Name-sg = Lemma ,
2854 name-sg = lemma ,
2855 Name-pl = Lemmas ,
2856 name-pl = lemmas ,
2857
2858 type = corollary ,

```

```

2859 Name-sg = Corollary ,
2860 name-sg = corollary ,
2861 Name-pl = Corollaries ,
2862 name-pl = corollaries ,
2863
2864 type = proposition ,
2865 Name-sg = Proposition ,
2866 name-sg = proposition ,
2867 Name-pl = Propositions ,
2868 name-pl = propositions ,
2869
2870 type = definition ,
2871 Name-sg = Definition ,
2872 name-sg = definition ,
2873 Name-pl = Definitions ,
2874 name-pl = definitions ,
2875
2876 type = proof ,
2877 Name-sg = Proof ,
2878 name-sg = proof ,
2879 Name-pl = Proofs ,
2880 name-pl = proofs ,
2881
2882 type = result ,
2883 Name-sg = Result ,
2884 name-sg = result ,
2885 Name-pl = Results ,
2886 name-pl = results ,
2887
2888 type = remark ,
2889 Name-sg = Remark ,
2890 name-sg = remark ,
2891 Name-pl = Remarks ,
2892 name-pl = remarks ,
2893
2894 type = example ,
2895 Name-sg = Example ,
2896 name-sg = example ,
2897 Name-pl = Examples ,
2898 name-pl = examples ,
2899
2900 type = algorithm ,
2901 Name-sg = Algorithm ,
2902 name-sg = algorithm ,
2903 Name-pl = Algorithms ,
2904 name-pl = algorithms ,
2905
2906 type = listing ,
2907 Name-sg = Listing ,
2908 name-sg = listing ,
2909 Name-pl = Listings ,
2910 name-pl = listings ,
2911
2912 type = exercise ,

```

```

2913   Name-sg = Exercise ,
2914   name-sg = exercise ,
2915   Name-pl = Exercises ,
2916   name-pl = exercises ,
2917
2918   type = solution ,
2919   Name-sg = Solution ,
2920   name-sg = solution ,
2921   Name-pl = Solutions ,
2922   name-pl = solutions ,
2923 </dict-english>

```

10.2 German

```

2924 <package>\zcDeclareLanguage { german }
2925 <package>\zcDeclareLanguageAlias { austrian      } { german }
2926 <package>\zcDeclareLanguageAlias { germanb       } { german }
2927 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2928 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2929 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2930 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2931 <*dict-german>
2932 namesep = {\nobreakspace} ,
2933 pairsep = {\~und\nobreakspace} ,
2934 listsep = {,~} ,
2935 lastsep = {\~und\nobreakspace} ,
2936 tpairsep = {\~und\nobreakspace} ,
2937 tlistsep = {,~} ,
2938 tlastsep = {\~und\nobreakspace} ,
2939 notesep = {\~} ,
2940 rangesep = {\~bis\nobreakspace} ,
2941
2942 type = part ,
2943   Name-sg = Teil ,
2944   name-sg = Teil ,
2945   Name-pl = Teile ,
2946   name-pl = Teile ,
2947
2948 type = chapter ,
2949   Name-sg = Kapitel ,
2950   name-sg = Kapitel ,
2951   Name-pl = Kapitel ,
2952   name-pl = Kapitel ,
2953
2954 type = section ,
2955   Name-sg = Abschnitt ,
2956   name-sg = Abschnitt ,
2957   Name-pl = Abschnitte ,
2958   name-pl = Abschnitte ,
2959
2960 type = paragraph ,
2961   Name-sg = Absatz ,
2962   name-sg = Absatz ,
2963   Name-pl = Absätze ,

```

```

2964     name-pl = Absätze ,
2965
2966 type = appendix ,
2967     Name-sg = Anhang ,
2968     name-sg = Anhang ,
2969     Name-pl = Anhänge ,
2970     name-pl = Anhänge ,
2971
2972 type = subappendix ,
2973     Name-sg = Anhang ,
2974     name-sg = Anhang ,
2975     Name-pl = Anhänge ,
2976     name-pl = Anhänge ,
2977
2978 type = page ,
2979     Name-sg = Seite ,
2980     name-sg = Seite ,
2981     Name-pl = Seiten ,
2982     name-pl = Seiten ,
2983
2984 type = line ,
2985     Name-sg = Zeile ,
2986     name-sg = Zeile ,
2987     Name-pl = Zeilen ,
2988     name-pl = Zeilen ,
2989
2990 type = figure ,
2991     Name-sg = Abbildung ,
2992     name-sg = Abbildung ,
2993     Name-pl = Abbildungen ,
2994     name-pl = Abbildungen ,
2995     Name-sg-ab = Abb. ,
2996     name-sg-ab = Abb. ,
2997     Name-pl-ab = Abb. ,
2998     name-pl-ab = Abb. ,
2999
3000 type = table ,
3001     Name-sg = Tabelle ,
3002     name-sg = Tabelle ,
3003     Name-pl = Tabellen ,
3004     name-pl = Tabellen ,
3005
3006 type = item ,
3007     Name-sg = Punkt ,
3008     name-sg = Punkt ,
3009     Name-pl = Punkte ,
3010     name-pl = Punkte ,
3011
3012 type = footnote ,
3013     Name-sg = Fußnote ,
3014     name-sg = Fußnote ,
3015     Name-pl = Fußnoten ,
3016     name-pl = Fußnoten ,
3017

```

```

3018 type = note ,
3019     Name-sg = Anmerkung ,
3020     name-sg = Anmerkung ,
3021     Name-pl = Anmerkungen ,
3022     name-pl = Anmerkungen ,
3023
3024 type = equation ,
3025     Name-sg = Gleichung ,
3026     name-sg = Gleichung ,
3027     Name-pl = Gleichungen ,
3028     name-pl = Gleichungen ,
3029     refpre-in = {} ,
3030     refpos-in = {} ,
3031
3032 type = theorem ,
3033     Name-sg = Theorem ,
3034     name-sg = Theorem ,
3035     Name-pl = Theoreme ,
3036     name-pl = Theoreme ,
3037
3038 type = lemma ,
3039     Name-sg = Lemma ,
3040     name-sg = Lemma ,
3041     Name-pl = Lemmata ,
3042     name-pl = Lemmata ,
3043
3044 type = corollary ,
3045     Name-sg = Korollar ,
3046     name-sg = Korollar ,
3047     Name-pl = Korollare ,
3048     name-pl = Korollare ,
3049
3050 type = proposition ,
3051     Name-sg = Satz ,
3052     name-sg = Satz ,
3053     Name-pl = Sätze ,
3054     name-pl = Sätze ,
3055
3056 type = definition ,
3057     Name-sg = Definition ,
3058     name-sg = Definition ,
3059     Name-pl = Definitionen ,
3060     name-pl = Definitionen ,
3061
3062 type = proof ,
3063     Name-sg = Beweis ,
3064     name-sg = Beweis ,
3065     Name-pl = Beweise ,
3066     name-pl = Beweise ,
3067
3068 type = result ,
3069     Name-sg = Ergebnis ,
3070     name-sg = Ergebnis ,
3071     Name-pl = Ergebnisse ,

```

```

3072     name-pl = Ergebnisse ,
3073
3074 type = remark ,
3075     Name-sg = Bemerkung ,
3076     name-sg = Bemerkung ,
3077     Name-pl = Bemerkungen ,
3078     name-pl = Bemerkungen ,
3079
3080 type = example ,
3081     Name-sg = Beispiel ,
3082     name-sg = Beispiel ,
3083     Name-pl = Beispiele ,
3084     name-pl = Beispiele ,
3085
3086 type = algorithm ,
3087     Name-sg = Algorithmus ,
3088     name-sg = Algorithmus ,
3089     Name-pl = Algorithmen ,
3090     name-pl = Algorithmen ,
3091
3092 type = listing ,
3093     Name-sg = Listing ,
3094     name-sg = Listing ,
3095     Name-pl = Listings ,
3096     name-pl = Listings ,
3097
3098 type = exercise ,
3099     Name-sg = Übungsaufgabe ,
3100     name-sg = Übungsaufgabe ,
3101     Name-pl = Übungsaufgaben ,
3102     name-pl = Übungsaufgaben ,
3103
3104 type = solution ,
3105     Name-sg = Lösung ,
3106     name-sg = Lösung ,
3107     Name-pl = Lösungen ,
3108     name-pl = Lösungen ,
3109 </dict-german>

```

10.3 French

```

3110 <package>\zcDeclareLanguage { french }
3111 <package>\zcDeclareLanguageAlias { acadian } { french }
3112 <package>\zcDeclareLanguageAlias { canadien } { french }
3113 <package>\zcDeclareLanguageAlias { francais } { french }
3114 <package>\zcDeclareLanguageAlias { frenchb } { french }
3115 <*dict-french>
3116 namesep = {\nobreakspace} ,
3117 pairsep = {\~et\nobreakspace} ,
3118 listsep = {,~} ,
3119 lastsep = {\~et\nobreakspace} ,
3120 tpairsep = {\~et\nobreakspace} ,
3121 tlistsep = {,~} ,
3122 tlastsep = {\~et\nobreakspace} ,

```



```

3123 notesep = {~} ,
3124 rangeseq = {~\nobreakspace} ,
3125
3126 type = part ,
3127   Name-sg = Partie ,
3128   name-sg = partie ,
3129   Name-pl = Parties ,
3130   name-pl = parties ,
3131
3132 type = chapter ,
3133   Name-sg = Chapitre ,
3134   name-sg = chapitre ,
3135   Name-pl = Chapitres ,
3136   name-pl = chapitres ,
3137
3138 type = section ,
3139   Name-sg = Section ,
3140   name-sg = section ,
3141   Name-pl = Sections ,
3142   name-pl = sections ,
3143
3144 type = paragraph ,
3145   Name-sg = Paragraphe ,
3146   name-sg = paragraphe ,
3147   Name-pl = Paragraphes ,
3148   name-pl = paragraphes ,
3149
3150 type = appendix ,
3151   Name-sg = Annexe ,
3152   name-sg = annexe ,
3153   Name-pl = Annexes ,
3154   name-pl = annexes ,
3155
3156 type = subappendix ,
3157   Name-sg = Annexe ,
3158   name-sg = annexe ,
3159   Name-pl = Annexes ,
3160   name-pl = annexes ,
3161
3162 type = page ,
3163   Name-sg = Page ,
3164   name-sg = page ,
3165   Name-pl = Pages ,
3166   name-pl = pages ,
3167
3168 type = line ,
3169   Name-sg = Ligne ,
3170   name-sg = ligne ,
3171   Name-pl = Lignes ,
3172   name-pl = lignes ,
3173
3174 type = figure ,
3175   Name-sg = Figure ,
3176   name-sg = figure ,

```

```

3177     Name-pl = Figures ,
3178     name-pl = figures ,
3179
3180 type = table ,
3181     Name-sg = Table ,
3182     name-sg = table ,
3183     Name-pl = Tables ,
3184     name-pl = tables ,
3185
3186 type = item ,
3187     Name-sg = Point ,
3188     name-sg = point ,
3189     Name-pl = Points ,
3190     name-pl = points ,
3191
3192 type = footnote ,
3193     Name-sg = Note ,
3194     name-sg = note ,
3195     Name-pl = Notes ,
3196     name-pl = notes ,
3197
3198 type = note ,
3199     Name-sg = Note ,
3200     name-sg = note ,
3201     Name-pl = Notes ,
3202     name-pl = notes ,
3203
3204 type = equation ,
3205     Name-sg = Équation ,
3206     name-sg = équation ,
3207     Name-pl = Équations ,
3208     name-pl = équations ,
3209     refpre-in = {() ,
3210     refpos-in = {} } ,
3211
3212 type = theorem ,
3213     Name-sg = Théorème ,
3214     name-sg = théorème ,
3215     Name-pl = Théorèmes ,
3216     name-pl = théorèmes ,
3217
3218 type = lemma ,
3219     Name-sg = Lemme ,
3220     name-sg = lemme ,
3221     Name-pl = Lemmes ,
3222     name-pl = lemmes ,
3223
3224 type = corollary ,
3225     Name-sg = Corollaire ,
3226     name-sg = corollaire ,
3227     Name-pl = Corollaires ,
3228     name-pl = corollaires ,
3229
3230 type = proposition ,

```

```

3231 Name-sg = Proposition ,
3232 name-sg = proposition ,
3233 Name-pl = Propositions ,
3234 name-pl = propositions ,
3235
3236 type = definition ,
3237 Name-sg = Définition ,
3238 name-sg = définition ,
3239 Name-pl = Définitions ,
3240 name-pl = définitions ,
3241
3242 type = proof ,
3243 Name-sg = Démonstration ,
3244 name-sg = démonstration ,
3245 Name-pl = Démonstrations ,
3246 name-pl = démonstrations ,
3247
3248 type = result ,
3249 Name-sg = Résultat ,
3250 name-sg = résultat ,
3251 Name-pl = Résultats ,
3252 name-pl = résultats ,
3253
3254 type = remark ,
3255 Name-sg = Remarque ,
3256 name-sg = remarque ,
3257 Name-pl = Remarques ,
3258 name-pl = remarques ,
3259
3260 type = example ,
3261 Name-sg = Exemple ,
3262 name-sg = exemple ,
3263 Name-pl = Exemples ,
3264 name-pl = exemples ,
3265
3266 type = algorithm ,
3267 Name-sg = Algorithme ,
3268 name-sg = algorithme ,
3269 Name-pl = Algorithmes ,
3270 name-pl = algorithmes ,
3271
3272 type = listing ,
3273 Name-sg = Liste ,
3274 name-sg = liste ,
3275 Name-pl = Listes ,
3276 name-pl = listes ,
3277
3278 type = exercise ,
3279 Name-sg = Exercice ,
3280 name-sg = exercice ,
3281 Name-pl = Exercices ,
3282 name-pl = exercices ,
3283
3284 type = solution ,

```

```

3285   Name-sg = Solution ,
3286   name-sg = solution ,
3287   Name-pl = Solutions ,
3288   name-pl = solutions ,
3289 </dict-french>

```

10.4 Portuguese

```

3290 <package>\zcDeclareLanguage { portuguese }
3291 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3292 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
3293 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3294 <*dict-portuguese>

3295 namesep = {\nobreakspace} ,
3296 pairsep = {\~e\nobreakspace} ,
3297 listsep = {,~} ,
3298 lastsep = {\~e\nobreakspace} ,
3299 tpairsep = {\~e\nobreakspace} ,
3300 tlistsep = {,~} ,
3301 tlastsep = {\~e\nobreakspace} ,
3302 notesep = {\~} ,
3303 rangesep = {\~a\nobreakspace} ,
3304
3305 type = part ,
3306   Name-sg = Parte ,
3307   name-sg = parte ,
3308   Name-pl = Partes ,
3309   name-pl = partes ,
3310
3311 type = chapter ,
3312   Name-sg = Capítulo ,
3313   name-sg = capítulo ,
3314   Name-pl = Capítulos ,
3315   name-pl = capítulos ,
3316
3317 type = section ,
3318   Name-sg = Seção ,
3319   name-sg = seção ,
3320   Name-pl = Seções ,
3321   name-pl = seções ,
3322
3323 type = paragraph ,
3324   Name-sg = Parágrafo ,
3325   name-sg = parágrafo ,
3326   Name-pl = Parágrafos ,
3327   name-pl = parágrafos ,
3328   Name-sg-ab = Par. ,
3329   name-sg-ab = par. ,
3330   Name-pl-ab = Par. ,
3331   name-pl-ab = par. ,
3332
3333 type = appendix ,
3334   Name-sg = Apêndice ,
3335   name-sg = apêndice ,

```

```

3336     Name-pl = Apêndices ,
3337     name-pl = apêndices ,
3338
3339     type = subappendix ,
3340     Name-sg = Apêndice ,
3341     name-sg = apêndice ,
3342     Name-pl = Apêndices ,
3343     name-pl = apêndices ,
3344
3345     type = page ,
3346     Name-sg = Página ,
3347     name-sg = página ,
3348     Name-pl = Páginas ,
3349     name-pl = páginas ,
3350     name-sg-ab = p. ,
3351     name-pl-ab = pp. ,
3352
3353     type = line ,
3354     Name-sg = Linha ,
3355     name-sg = linha ,
3356     Name-pl = Linhas ,
3357     name-pl = linhas ,
3358
3359     type = figure ,
3360     Name-sg = Figura ,
3361     name-sg = figura ,
3362     Name-pl = Figuras ,
3363     name-pl = figuras ,
3364     Name-sg-ab = Fig. ,
3365     name-sg-ab = fig. ,
3366     Name-pl-ab = Figs. ,
3367     name-pl-ab = figs. ,
3368
3369     type = table ,
3370     Name-sg = Tabela ,
3371     name-sg = tabela ,
3372     Name-pl = Tabelas ,
3373     name-pl = tabelas ,
3374
3375     type = item ,
3376     Name-sg = Item ,
3377     name-sg = item ,
3378     Name-pl = Itens ,
3379     name-pl = itens ,
3380
3381     type = footnote ,
3382     Name-sg = Nota ,
3383     name-sg = nota ,
3384     Name-pl = Notas ,
3385     name-pl = notas ,
3386
3387     type = note ,
3388     Name-sg = Nota ,
3389     name-sg = nota ,

```

```

3390     Name-pl = Notas ,
3391     name-pl = notas ,
3392
3393     type = equation ,
3394     Name-sg = Equação ,
3395     name-sg = equação ,
3396     Name-pl = Equações ,
3397     name-pl = equações ,
3398     Name-sg-ab = Eq. ,
3399     name-sg-ab = eq. ,
3400     Name-pl-ab = Eqs. ,
3401     name-pl-ab = eqs. ,
3402     refpre-in = {()} ,
3403     refpos-in = {} ,
3404
3405     type = theorem ,
3406     Name-sg = Teorema ,
3407     name-sg = teorema ,
3408     Name-pl = Teoremas ,
3409     name-pl = teoremas ,
3410
3411     type = lemma ,
3412     Name-sg = Lema ,
3413     name-sg = lema ,
3414     Name-pl = Lemas ,
3415     name-pl = lemas ,
3416
3417     type = corollary ,
3418     Name-sg = Corolário ,
3419     name-sg = corolário ,
3420     Name-pl = Corolários ,
3421     name-pl = corolários ,
3422
3423     type = proposition ,
3424     Name-sg = Proposição ,
3425     name-sg = proposição ,
3426     Name-pl = Proposições ,
3427     name-pl = proposições ,
3428
3429     type = definition ,
3430     Name-sg = Definição ,
3431     name-sg = definição ,
3432     Name-pl = Definições ,
3433     name-pl = definições ,
3434
3435     type = proof ,
3436     Name-sg = Demonstração ,
3437     name-sg = demonstração ,
3438     Name-pl = Demonstrações ,
3439     name-pl = demonstrações ,
3440
3441     type = result ,
3442     Name-sg = Resultado ,
3443     name-sg = resultado ,

```

```

3444   Name-pl = Resultados ,
3445   name-pl = resultados ,
3446
3447   type = remark ,
3448   Name-sg = Observação ,
3449   name-sg = observação ,
3450   Name-pl = Observações ,
3451   name-pl = observações ,
3452
3453   type = example ,
3454   Name-sg = Exemplo ,
3455   name-sg = exemplo ,
3456   Name-pl = Exemplos ,
3457   name-pl = exemplos ,
3458
3459   type = algorithm ,
3460   Name-sg = Algoritmo ,
3461   name-sg = algoritmo ,
3462   Name-pl = Algoritmos ,
3463   name-pl = algoritmos ,
3464
3465   type = listing ,
3466   Name-sg = Listagem ,
3467   name-sg = listagem ,
3468   Name-pl = Listagens ,
3469   name-pl = listagens ,
3470
3471   type = exercise ,
3472   Name-sg = Exercício ,
3473   name-sg = exercício ,
3474   Name-pl = Exercícios ,
3475   name-pl = exercícios ,
3476
3477   type = solution ,
3478   Name-sg = Solução ,
3479   name-sg = solução ,
3480   Name-pl = Soluções ,
3481   name-pl = soluções ,
3482 </dict-portuguese>

```

10.5 Spanish

```

3483 <package>\zcDeclareLanguage { spanish }
3484 <*dict-spanish>
3485 namesep = {\nobreakspace} ,
3486 pairsep = {\~y\nobreakspace} ,
3487 listsep = {,~} ,
3488 lastsep = {\~y\nobreakspace} ,
3489 tpairsep = {\~y\nobreakspace} ,
3490 tlistsep = {,~} ,
3491 tlastsep = {\~y\nobreakspace} ,
3492 notesep = {\~} ,
3493 rangesep = {\~a\nobreakspace} ,
3494

```

```

3495 type = part ,
3496     Name-sg = Parte ,
3497     name-sg = parte ,
3498     Name-pl = Partes ,
3499     name-pl = partes ,
3500
3501 type = chapter ,
3502     Name-sg = Capítulo ,
3503     name-sg = capítulo ,
3504     Name-pl = Capítulos ,
3505     name-pl = capítulos ,
3506
3507 type = section ,
3508     Name-sg = Sección ,
3509     name-sg = sección ,
3510     Name-pl = Secciones ,
3511     name-pl = secciones ,
3512
3513 type = paragraph ,
3514     Name-sg = Párrafo ,
3515     name-sg = párrafo ,
3516     Name-pl = Párrafos ,
3517     name-pl = párrafos ,
3518
3519 type = appendix ,
3520     Name-sg = Apéndice ,
3521     name-sg = apéndice ,
3522     Name-pl = Apéndices ,
3523     name-pl = apéndices ,
3524
3525 type = subappendix ,
3526     Name-sg = Apéndice ,
3527     name-sg = apéndice ,
3528     Name-pl = Apéndices ,
3529     name-pl = apéndices ,
3530
3531 type = page ,
3532     Name-sg = Página ,
3533     name-sg = página ,
3534     Name-pl = Páginas ,
3535     name-pl = páginas ,
3536
3537 type = line ,
3538     Name-sg = Línea ,
3539     name-sg = línea ,
3540     Name-pl = Líneas ,
3541     name-pl = líneas ,
3542
3543 type = figure ,
3544     Name-sg = Figura ,
3545     name-sg = figura ,
3546     Name-pl = Figuras ,
3547     name-pl = figuras ,
3548

```



```

3549 type = table ,
3550     Name-sg = Cuadro ,
3551     name-sg = cuadro ,
3552     Name-pl = Cuadros ,
3553     name-pl = cuadros ,
3554
3555 type = item ,
3556     Name-sg = Punto ,
3557     name-sg = punto ,
3558     Name-pl = Puntos ,
3559     name-pl = puntos ,
3560
3561 type = footnote ,
3562     Name-sg = Nota ,
3563     name-sg = nota ,
3564     Name-pl = Notas ,
3565     name-pl = notas ,
3566
3567 type = note ,
3568     Name-sg = Nota ,
3569     name-sg = nota ,
3570     Name-pl = Notas ,
3571     name-pl = notas ,
3572
3573 type = equation ,
3574     Name-sg = Ecuación ,
3575     name-sg = ecuación ,
3576     Name-pl = Ecuaciones ,
3577     name-pl = ecuaciones ,
3578     refpre-in = {(} ,
3579     refpos-in = {)} ,
3580
3581 type = theorem ,
3582     Name-sg = Teorema ,
3583     name-sg = teorema ,
3584     Name-pl = Teoremas ,
3585     name-pl = teoremas ,
3586
3587 type = lemma ,
3588     Name-sg = Lema ,
3589     name-sg = lema ,
3590     Name-pl = Lemas ,
3591     name-pl = lemas ,
3592
3593 type = corollary ,
3594     Name-sg = Corolario ,
3595     name-sg = corolario ,
3596     Name-pl = Corolarios ,
3597     name-pl = corolarios ,
3598
3599 type = proposition ,
3600     Name-sg = Proposición ,
3601     name-sg = proposición ,
3602     Name-pl = Proposiciones ,

```

```

3603     name-pl = proposiciones ,
3604
3605 type = definition ,
3606     Name-sg = Definición ,
3607     name-sg = definición ,
3608     Name-pl = Definiciones ,
3609     name-pl = definiciones ,
3610
3611 type = proof ,
3612     Name-sg = Demostración ,
3613     name-sg = demostración ,
3614     Name-pl = Demostraciones ,
3615     name-pl = demostraciones ,
3616
3617 type = result ,
3618     Name-sg = Resultado ,
3619     name-sg = resultado ,
3620     Name-pl = Resultados ,
3621     name-pl = resultados ,
3622
3623 type = remark ,
3624     Name-sg = Observación ,
3625     name-sg = observación ,
3626     Name-pl = Observaciones ,
3627     name-pl = observaciones ,
3628
3629 type = example ,
3630     Name-sg = Ejemplo ,
3631     name-sg = ejemplo ,
3632     Name-pl = Ejemplos ,
3633     name-pl = ejemplos ,
3634
3635 type = algorithm ,
3636     Name-sg = Algoritmo ,
3637     name-sg = algoritmo ,
3638     Name-pl = Algoritmos ,
3639     name-pl = algoritmos ,
3640
3641 type = listing ,
3642     Name-sg = Listado ,
3643     name-sg = listado ,
3644     Name-pl = Listados ,
3645     name-pl = listados ,
3646
3647 type = exercise ,
3648     Name-sg = Ejercicio ,
3649     name-sg = ejercicio ,
3650     Name-pl = Ejercicios ,
3651     name-pl = ejercicios ,
3652
3653 type = solution ,
3654     Name-sg = Solución ,
3655     name-sg = solución ,
3656     Name-pl = Soluciones ,

```

3657 name-pl = soluciones ,
 3658 </dict-spanish>

Index

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

Symbols	
\\	107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155
† internal commands:	
\l_zrefclever_current_counter_tl	4
A	
\AddToHook	95, 475, 490, 634, 670, 695, 733, 735, 787, 808, 2621, 2623, 2628, 2641, 2645, 2658, 2675, 2698
\appendix	70
\appendixname	70
B	
\babelname	680
\babelprovide	12, 23
bool commands:	
\bool_case_true:	2
\bool_if:N	305, 316, 638, 642, 1383, 1425, 1633, 1728, 1858, 1880, 1911, 1957, 1998, 2021, 2025, 2031, 2041, 2047, 2203
\bool_if:nTF	63, 1209, 1218, 1227, 1293, 1303, 1327, 1355, 1392, 1499, 1507, 1647, 1655, 1892, 1899, 1906, 2156, 2277
\bool_lazy_all:nTF	2541
\bool_lazy_and:nnTF	1101, 1119, 2352, 2598
\bool_lazy_any:nTF	2430, 2439
\bool_lazy_or:nnTF	1105, 2340
\bool_new:N	268, 511, 512, 537, 561, 570, 577, 578, 591, 592, 611, 612, 801, 802, 1130, 1145, 1539, 1540, 1550, 1556, 1557
\bool_set:Nn	1099
\bool_set_false:N	524, 528, 619, 628, 629, 644, 824, 1285, 1596, 1639, 1653, 1667, 1870, 1996, 1997, 2437, 2454
\bool_set_true:N	325, 518, 519, 523, 529, 618, 623, 624, 812, 817, 1299, 1309, 1313, 1339, 1350, 1379, 1387, 1420, 1429, 1457, 1469, 1604, 1634, 1640, 1644, 1671, 1677, 2453, 2489, 2496, 2498, 2517, 2524, 2526
\bool_until_do:Nn	1325, 1597
\bool_while_do:nn	2703
C	
clist commands:	
\clist_map_inline:nn	867
\counterwithin	4
cs commands:	
\cs_generate_variant:Nn	59, 60, 321, 329, 1010, 1016, 1261, 2197, 2467
\cs_if_exist:N	43, 52, 73
\cs_if_exist_p:N	2705
\cs_new:Npn	41, 50, 61, 71, 82, 2152, 2198, 2457
\cs_new_protected:Npn	269, 322, 330, 336, 457, 1005, 1011, 1094, 1149, 1191, 1202, 1262, 1477, 1529, 1573, 1735, 1992, 2148, 2150, 2330, 2468, 2535, 2592
\cs_new_protected:Npx	94
\cs_set_eq:NN	98
E	
\endinput	12
exp commands:	
\exp_args:NNe	27, 30
\exp_args:NNnx	259
\exp_args:NnV	297
\exp_args:NNx	99
\exp_args:Nnx	332, 1365, 1402
\exp_args:Nx	279, 2625, 2709
\exp_args:Nxx	1245, 1335, 1438, 2481, 2505, 2509
\exp_not:N	59, 1913, 1916, 1927, 1930, 1933, 2162, 2165, 2168, 2177, 2179, 2182, 2185, 2190, 2192, 2210, 2221, 2224, 2226, 2229, 2236, 2243, 2245, 2249, 2252, 2255, 2267, 2270, 2283, 2286, 2289, 2306, 2308, 2311, 2314, 2321, 2323

<code>\exp_not:n</code> . 1757, 1773, 1785, 1790, 1813, 1827, 1831, 1843, 1847, 1881, 1882, 1914, 1926, 1931, 1932, 2061, 2074, 2081, 2105, 2117, 2121, 2131, 2135, 2163, 2164, 2166, 2172, 2175, 2178, 2183, 2184, 2186, 2187, 2189, 2191, 2222, 2223, 2225, 2227, 2228, 2230, 2231, 2235, 2247, 2248, 2253, 2254, 2256, 2264, 2268, 2269, 2271, 2284, 2285, 2287, 2300, 2304, 2307, 2312, 2313, 2315, 2316, 2320, 2322 <code>\ExplSyntaxOn</code> 12, 281	<code>\int_zero:N</code> 1361, 1398, 1479, 1480, 1582, 1583, 1584, 1585, 1986, 1988, 1989, 2141, 2142 <code>\l_tmpa_int</code> 1398, 1401, 1411, 2702, 2706, 2713, 2714, 2717, 2719, 2721 <code>\l_tmpb_int</code> 1361, 1364, 1375 iow commands: <code>\iow_char:N</code> 107, 113, 122, 123, 128, 129, 134, 135, 144, 145, 155
F file commands: <code>\file_get:nnNTF</code> 279 <code>\fmtversion</code> 3 <code>\footnote</code> 69	K keys commands: <code>\keys_define:nn</code> 30, 342, 354, 371, 385, 464, 494, 501, 513, 538, 547, 562, 571, 579, 593, 605, 613, 646, 653, 691, 738, 780, 782, 789, 796, 803, 813, 825, 834, 863, 889, 913, 923, 934, 955, 967, 1017, 1029, 1050, 1073 <code>\keys_set:nn</code> 12, 30, 34, 298, 818, 944, 950, 999, 1097 keyval commands: <code>\keyval_parse:nnn</code> 838, 893
G group commands: <code>\group_begin:</code> . . 97, 271, 324, 994, 1096, 1109, 1913, 1930, 2162, 2165, 2182, 2185, 2221, 2226, 2229, 2245, 2252, 2267, 2283, 2286, 2311, 2314 <code>\group_end:</code> . . . 100, 319, 327, 1002, 1112, 1127, 1927, 1933, 2177, 2179, 2190, 2192, 2224, 2236, 2243, 2249, 2255, 2270, 2306, 2308, 2321, 2323	L <code>\labelformat</code> 3 <code>\language name</code> 22, 674
I <code>\IfBooleanTF</code> 1133 <code>\IfFormatAtLeastTF</code> 3, 4 <code>\input</code> 12 int commands: <code>\int_case:nnTF</code> 1738, 1766, 1798, 1960, 2054, 2093 <code>\int_compare:nNnTF</code> 1249, 1340, 1370, 1407, 1442, 1458, 1485, 1487, 1531, 1699, 1753, 1787, 1949, 1951, 2009, 2034, 2078, 2485, 2491, 2513, 2519, 2721 <code>\int_compare_p:nNn</code> 1501, 1509, 2344, 2355, 2450 <code>\int_eval:n</code> 94 <code>\int_incr:N</code> 1364, 1401, 1987, 2024, 2026, 2040, 2042, 2046, 2048, 2146, 2719 <code>\int_new:N</code> 1146, 1147, 1541, 1542, 1553, 1554 <code>\int_set:Nn</code> 1486, 1488, 1492, 1495, 2702 <code>\int_to_roman:n</code> 2706, 2713, 2714, 2717 <code>\int_use:N</code> 37, 39, 54	M <code>\mainbabelname</code> 22, 681 <code>\MessageBreak</code> 10 msg commands: <code>\msg_info:nnn</code> 362, 392, 2671, 2694, 2722 <code>\msg_line_context:</code> . 106, 112, 116, 118, 121, 127, 133, 140, 143, 149, 154, 161, 166, 170, 172, 174, 177, 181 <code>\msg_new:nnn</code> 104, 110, 115, 117, 119, 125, 131, 137, 139, 141, 147, 152, 157, 159, 164, 169, 171, 173, 175, 180, 182, 184 <code>\msg_note:nnn</code> 301 <code>\msg_warning:nn</code> 480, 505, 643, 649, 792, 829 <code>\msg_warning:nnn</code> . . . 248, 263, 307, 317, 721, 766, 895, 959, 1001, 1041, 1080, 1692, 1865, 2383, 2418, 2584 <code>\msg_warning:nnnn</code> 840, 1348, 1385, 1427, 1467
N <code>\newcounter</code> 4 <code>\NewDocumentCommand</code> 243, 253, 943, 945, 992, 1092, 1131 <code>\nobreakspace</code> 406, 2736, 2737, 2739, 2740, 2742, 2744,	

<code>\zcRefTypeSetup</code>	9, 30, 945	<code>__zrefclever_get_enclosing_</code>	
<code>\zcsetup</code>	22 , 25 , 29 , 30 , 943 , 2625 ,	<code>counters:n</code>	5, 41 , 46, 88
	2630 , 2647 , 2660 , 2679 , 2692 , 2709	<code>__zrefclever_get_enclosing_</code>	
<code>\zlabel</code>	72 , 2691	<code>counters_value:n</code>	5, 41 , 55, 90
zrefcheck commands:		<code>__zrefclever_get_fallback_</code>	
<code>\zrefcheck_zcref_beg_label:</code> . .	1104	<code>transl:nN</code>	450
<code>\zrefcheck_zcref_end_label_</code>		<code>__zrefclever_get_fallback_</code>	
<code>maybe:</code>	1123	<code>transl:nNTF</code>	17 , 448 , 2581
<code>\zrefcheck_zcref_run_checks_on_</code>		<code>__zrefclever_get_ref:n</code>	
<code>labels:n</code>	1124	58 , 59 , 1758 , 1774 ,
zrefclever internal commands:		1786 , 1791 , 1814 , 1828 , 1832 , 1844 ,
<code>\l__zrefclever_abbrev_bool</code>	591 , 595 , 2353	1848 , 1883 , 1903 , 2062 , 2075 , 2082 ,
<code>\l__zrefclever_capitalize_bool</code> . .	577 , 581 , 2341	<code>__zrefclever_get_ref_first:</code> . . .	2106 , 2118 , 2122 , 2132 , 2136 , 2152
<code>\l__zrefclever_capitalize_first_</code>		58 , 59 , 63 , 1896 , 1943 , 2198
<code>bool</code>	578 , 587 , 2343	<code>__zrefclever_get_ref_font:nN</code> .	9 ,
<code>__zrefclever_counter_reset_by:n</code>	5 , 6 , 27 , 28 , 43 , 45 , 47 , 52 , 54 , 56 , 61	16 , 29 , 67 , 68 , 1719 , 1721 , 1723 , 2592	
<code>__zrefclever_counter_reset_by_</code>		<code>__zrefclever_get_ref_string:nN</code> .	
<code>aux:nn</code>	68 , 71	9 , 15 , 29 , 67 , 1115 , 1588 ,
<code>__zrefclever_counter_reset_by_</code>		1590 , 1592 , 1701 , 1703 , 1705 , 1707 ,
<code>auxi:nnn</code>	78 , 82	1709 , 1711 , 1713 , 1715 , 1717 , 2535
<code>\l__zrefclever_counter_resetby_</code>		<code>__zrefclever_get_type_transl:nnnN</code>
<code>prop</code>	5 , 28 , 64 , 65 , 888 , 900	9 , 417 , 431
<code>\l__zrefclever_counter_resettters_</code>		<code>__zrefclever_get_type_transl:nnnNTF</code>
<code>seq</code>	4 , 5 , 27 , 28 , 67 , 862 , 869 , 872	16 , 416 , 2376 , 2405 , 2411 , 2570
<code>\l__zrefclever_counter_type_prop</code>	3 , 26 , 27 , 30 , 833 , 845	<code>\l__zrefclever_label_a_tl</code>
<code>\l__zrefclever_current_counter_</code>		45 , 1543 , 1600 , 1620 , 1636 , 1686 ,
<code>tl</code>	3 , 28 , 20 , 23 , 28 , 31 , 33 , 37 , 88 , 90 , 912 , 915	1687 , 1693 , 1745 , 1758 , 1774 , 1791 ,
<code>\l__zrefclever_current_language_</code>		1832 , 1848 , 1876 , 1883 , 2000 , 2004 ,
<code>tl</code>	22 , 669 , 674 , 680 , 684 , 710 , 755	2014 , 2039 , 2062 , 2083 , 2122 , 2136
<code>__zrefclever_declare_default_</code>		<code>\l__zrefclever_label_b_tl</code>
<code>transl:nnn</code>	32 , 1005 , 1036 , 1057	45 , 1543 ,
<code>__zrefclever_declare_type_</code>		1603 , 1608 , 1625 , 1638 , 1643 , 2004
<code>transl:nnnn</code>	32 , 1005 , 1062 , 1084	<code>\l__zrefclever_label_count_int</code>
<code>\g__zrefclever_dict_⟨language⟩_prop</code>	12	46 , 1541 ,
<code>\l__zrefclever_dict_language_tl</code> .	186 , 273 , 277 , 280 , 287 , 293 , 300 ,	1582 , 1699 , 1738 , 1986 , 2009 , 2146
.	302 , 308 , 311 , 333 , 339 , 420 , 423 ,	<code>\l__zrefclever_label_enclcnt_a_</code>	
.	436 , 439 , 996 , 1037 , 1058 , 1063 , 1085	<code>tl</code>	1137 , 1264 , 1266 , 1267 ,
<code>__zrefclever_extract_url:n</code>	1918 , 2169 , 2212 , 2291 , 2457	1331 , 1358 , 1399 , 1439 , 1447 , 1448
<code>\g__zrefclever_fallback_dict_</code>		<code>\l__zrefclever_label_enclcnt_b_</code>	
<code>prop</code>	9 , 399 , 400 , 452	<code>tl</code>	1137 , 1268 , 1270 , 1271 ,
<code>\l__zrefclever_footnote_type_tl</code> .	2619 , 2620 , 2622 , 2626	1332 , 1362 , 1395 , 1440 , 1449 , 1450
<code>__zrefclever_get_default_</code>		<code>\l__zrefclever_label_enclval_a_</code>	
<code>transl:nnN</code>	9 , 433 , 447	<code>tl</code>	1137 , 1272 , 1274 ,
<code>__zrefclever_get_default_</code>		1275 , 1410 , 1443 , 1451 , 1452 , 1459
<code>transl:nnNTF</code>	17 , 432 , 2576	<code>\l__zrefclever_label_enclval_b_</code>	
		<code>tl</code>	1137 , 1276 , 1278 ,
		1279 , 1374 , 1445 , 1453 , 1454 , 1461
		<code>\l__zrefclever_label_extdoc_a_tl</code>
		1137 , 1280 ,
		1290 , 1295 , 1305 , 1318 , 2470 , 2476
		<code>\l__zrefclever_label_extdoc_b_tl</code>
		1137 , 1282 ,
		1291 , 1296 , 1306 , 1317 , 2472 , 2477

`\l_zrefclever_label_type_a_tl` 67, 1137, 1193, 1196,
1199, 1204, 1213, 1222, 1230, 1235,
1491, 1519, 1613, 1617, 1650, 1658,
1664, 1690, 1747, 2016, 2543, 2548,
2555, 2564, 2572, 2599, 2604, 2611
`\l_zrefclever_label_type_b_tl` 1137,
1206, 1214, 1223, 1231, 1236, 1494,
1522, 1614, 1622, 1651, 1660, 1665
`_zrefclever_label_type_put_-`
`new_right:n` 36, 38, 1155, 1191
`\l_zrefclever_label_types_seq` 37, 1148, 1151, 1195, 1198, 1517
`_zrefclever_labels_in_sequence:nn`
. 46, 66, 1874, 2003, 2468
`\g_zrefclever_languages_prop` 11, 242, 247, 249, 257,
260, 261, 272, 419, 435, 716, 761, 995
`\l_zrefclever_last_of_type_bool`
. 45, 1538, 1634, 1639, 1640,
1644, 1653, 1668, 1672, 1678, 1728
`\l_zrefclever_lastsep_tl` 1558,
1710, 1773, 1790, 1813, 1831, 1843
`\l_zrefclever_link_star_bool` 1099, 1129, 2159, 2280, 2433
`\l_zrefclever_listsep_tl` 1558, 1708, 1785, 1827, 2061,
2074, 2081, 2105, 2117, 2121, 2131
`\l_zrefclever_load_dict_-`
`verbose_bool` 268, 305, 316, 325
`\g_zrefclever_loaded_dictionaries_-`
`seq` 267, 276, 299, 310
`\l_zrefclever_main_language_tl` 22, 668,
675, 681, 685, 689, 702, 724, 747, 769
`_zrefclever_name_default:` 2148, 2263
`\l_zrefclever_name_format_-`
`fallback_tl` 1549, 2360, 2364, 2366, 2402, 2414
`\l_zrefclever_name_format_tl` 1549, 2346, 2347, 2350, 2351,
2361, 2362, 2373, 2379, 2394, 2408
`\l_zrefclever_name_in_link_bool`
. 60,
63, 1549, 1911, 2203, 2437, 2453, 2454
`\l_zrefclever_namefont_tl` 1558,
1720, 1914, 1931, 2222, 2253, 2268
`\l_zrefclever_nameinlink_str` 652, 657,
659, 661, 663, 2435, 2441, 2443, 2447
`\l_zrefclever_namesep_tl` 1558, 1702, 2225, 2256, 2264, 2271
`\l_zrefclever_next_is_same_bool`
. 46, 66, 1553,
1997, 2025, 2041, 2047, 2499, 2527
`\l_zrefclever_next_maybe_range_-`
`bool` 46, 66, 1553, 1870, 1880, 1996,
2021, 2031, 2489, 2497, 2517, 2525
`\l_zrefclever_noabbrev_first_-`
`bool` 592, 601, 2357
`_zrefclever_page_format_aux:` 94, 98
`\g_zrefclever_page_format_tl` 7, 93, 99, 102
`\l_zrefclever_pairsep_tl` 1558, 1706, 1757, 1881
`_zrefclever_prop_put_non_-`
`empty:Nnn` 17, 457, 844, 899
`_zrefclever_provide_dict_-`
`default_transl:nn` 14, 330, 360, 377
`_zrefclever_provide_dict_type_-`
`transl:nn` 14, 330, 378, 395
`_zrefclever_provide_dictionary:n`
. 9, 12-14,
34, 269, 326, 737, 748, 756, 771, 1100
`_zrefclever_provide_dictionary_-`
`verbose:n` 13, 322, 703, 711, 726
`\l_zrefclever_range_beg_label_-`
`tl` 46, 1553, 1581,
1786, 1809, 1815, 1825, 1829, 1841,
1845, 1985, 2023, 2038, 2072, 2076,
2103, 2107, 2115, 2119, 2129, 2133
`\l_zrefclever_range_count_int` 46,
1553, 1584, 1766, 1800, 1988, 2024,
2035, 2040, 2046, 2054, 2095, 2141
`\l_zrefclever_range_same_count_-`
`int` 46,
1553, 1585, 1753, 1788, 1801, 1989,
2026, 2042, 2048, 2079, 2096, 2142
`\l_zrefclever_rangesep_tl` 1558, 1704, 1847, 1882, 2135
`_zrefclever_ref_default:` 2148, 2195, 2201, 2257, 2326
`\l_zrefclever_ref_language_tl` 22, 23, 667, 688,
701, 704, 709, 712, 718, 723, 727,
737, 746, 749, 754, 757, 763, 768,
772, 1100, 2377, 2406, 2412, 2571, 2577
`\c_zrefclever_ref_options_font_-`
`seq` 10, 16, 188
`\c_zrefclever_ref_options_-`
`necessarily_not_type_specific_-`
`seq` 15, 188, 352, 953, 1027

\c_zrefclever_ref_options_- necessarily_type_specific_seq	188, 383, 1071	\l_zrefclever_sort_prior_b_int	1146, 1480, 1487, 1488, 1495, 1503, 1511
\c_zrefclever_ref_options_- possibly_type_specific_seq	15, 188, 369, 1048	\l_zrefclever_tlastsep_tl	1558, 1593, 1974
\l_zrefclever_ref_options_prop	29, 30, 919, 929, 930, 2538, 2595	\l_zrefclever_tlistsep_tl	1558, 1591, 1952
\c_zrefclever_ref_options_- reference_seq	188, 921	\l_zrefclever_tpairsep_tl	1558, 1589, 1968
\c_zrefclever_ref_options_- typesetup_seq	188, 965	\l_zrefclever_type_<type>- options_prop	30
\l_zrefclever_ref_property_tl	18, 463, 468, 470, 472, 478, 481, 497, 506, 1152, 1184, 1611, 2154, 2174, 2188, 2207, 2234, 2275, 2303, 2319, 2479	\l_zrefclever_type_count_int	46, 63, 1541, 1583, 1949, 1951, 1960, 1987, 2344, 2356, 2450
\l_zrefclever_ref_typeset_font_- tl	779, 781, 1110	\l_zrefclever_type_first_label_- tl	45, 60, 1543, 1579, 1744, 1862, 1871, 1875, 1903, 1919, 1923, 1983, 2013, 2200, 2206, 2213, 2217, 2233, 2274, 2292, 2296, 2302, 2318, 2332
\l_zrefclever_reffont_in_tl	1558, 1724, 2166, 2186, 2230, 2287, 2315	\l_zrefclever_type_first_label_- type_tl	45, 63, 1543, 1580, 1746, 1866, 1984, 2015, 2335, 2371, 2378, 2384, 2392, 2400, 2407, 2413, 2420
\l_zrefclever_reffont_out_tl	1558, 1722, 2163, 2183, 2227, 2247, 2284, 2312	_zrefclever_type_name_setup:	9, 60, 1891, 2330
\l_zrefclever_refpos_in_tl	1558, 1718, 2175, 2189, 2235, 2304, 2320	\l_zrefclever_type_name_tl	60, 63, 1549, 1926, 1932, 2223, 2254, 2261, 2269, 2333, 2336, 2374, 2380, 2382, 2395, 2403, 2409, 2415, 2417, 2434
\l_zrefclever_refpos_out_tl	1558, 1714, 2178, 2191, 2248, 2307, 2322	\l_zrefclever_typeset_compress_- bool	561, 564, 1998
\l_zrefclever_refpre_in_tl	1558, 1716, 2172, 2187, 2231, 2300, 2316	\l_zrefclever_typeset_labels_- seq	45, 1538, 1575, 1599, 1601, 1607
\l_zrefclever_refpre_out_tl	1558, 1712, 2164, 2184, 2228, 2285, 2313	\l_zrefclever_typeset_last_bool	45, 1538, 1596, 1597, 1604, 1633, 1957, 2449
\l_zrefclever_setup_type_tl	14, 186, 296, 334, 347, 348, 359, 376, 390, 949, 977, 985, 998, 1022, 1023, 1034, 1055, 1064, 1078, 1086	\l_zrefclever_typeset_name_bool	512, 519, 524, 529, 1893, 1907
\l_zrefclever_sort_decided_bool	1145, 1285, 1299, 1309, 1313, 1325, 1339, 1350, 1379, 1383, 1387, 1421, 1425, 1429, 1457, 1469	\l_zrefclever_typeset_queue_- curr_tl	46, 59, 63, 1543, 1578, 1755, 1771, 1780, 1811, 1822, 1838, 1860, 1878, 1895, 1902, 1909, 1942, 1964, 1969, 1975, 1981, 1982, 2059, 2070, 2101, 2113, 2127, 2349, 2444, 2448
_zrefclever_sort_default:nn	38, 1186, 1202	\l_zrefclever_typeset_queue_- prev_tl .	46, 1543, 1577, 1953, 1980
_zrefclever_sort_default_- different_types:nn	19, 36, 43, 1240, 1477	\l_zrefclever_typeset_range_- bool	570, 573, 1107, 1858
_zrefclever_sort_default_same_- type:nn	36, 39, 1238, 1262	\l_zrefclever_typeset_ref_bool	511, 518, 523, 528, 1893, 1900
_zrefclever_sort_labels:	36-38, 44, 1108, 1149	_zrefclever_typeset_refs:	45-47, 1111, 1573
_zrefclever_sort_page:nn	44, 1185, 1529		
\l_zrefclever_sort_prior_a_int	1146, 1479, 1485, 1486, 1492, 1502, 1510		

_zrefclever_typeset_refs_last_-	bool	612, 619, 624, 629, 642
of_type: . 51, 59, 60, 63, 1730, 1735	_zrefclever_zcref:nnn . .	1093, 1094
_zrefclever_typeset_refs_not_-	_zrefclever_zcref:nnnn	34, 36, 1094
last_of_type:	\l_zrefclever_zcref_labels_seq .	
. 46, 51, 59, 66, 1732, 1992	36,
\l_zrefclever_typeset_sort_bool	37, 1098, 1125, 1129, 1154, 1157, 1576	
. 537, 540, 1106	\l_zrefclever_zcref_note_tl . . .	
\l_zrefclever_typesort_seq 795, 798, 1113, 1117	
. . . . 19, 43, 546, 551, 552, 558, 1481	\l_zrefclever_zcref_with_check_-	
\l_zrefclever_use_hyperref_bool	bool	802, 817, 1103, 1121
. 611, 618,	\l_zrefclever_zrefcheck_-	
623, 628, 638, 644, 2158, 2279, 2432	available_bool	
\l_zrefclever_warn_hyperref_-	801, 812, 824, 1102, 1120