

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	Data extraction	8
4.3	Reference format	9
4.4	Languages	11
4.5	Dictionaries	12
4.6	Options	18
5	Configuration	30
5.1	\zcsetup	30
5.2	\zcRefTypeSetup	31
5.3	\zcLanguageSetup	32
6	User interface	34
6.1	\zceref	34
6.2	\zcpageref	36
7	Sorting	36
8	Typesetting	44
9	Compatibility	69
9.1	\footnote	70
9.2	\appendix	70
9.3	appendix package	71
9.4	amsmath package	72
9.5	listings package	74
9.6	enumitem package	75

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

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

10	Dictionaries	76
10.1	English	76
10.2	German	80
10.3	French	83
10.4	Portuguese	86
10.5	Spanish	90
Index		94

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_value:n`

Recursively generate a *sequence* of “enclosing counters” 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_value:n {⟨counter⟩}

41 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
42 {
43   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
44   {
45     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
46     \__zrefclever_get_enclosing_counters_value:e
47     { \__zrefclever_counter_reset_by:n {#1} }
48   }
49 }

```

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’).

```
50 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }
```

(End definition for `__zrefclever_get_enclosing_counters_value:n`.)

`__zrefclever_counter_reset_by:n`

Auxiliary function for `__zrefclever_get_enclosing_counters_value:n`, and useful on its own standing. It is 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⟩}

51 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
52 {
53   \bool_if:nTF
54   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
55   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
56   {
57     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
58     { \__zrefclever_counter_reset_by_aux:nn {#1} }
59   }
60 }

```

```

61 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
62 {
63   \cs_if_exist:cT { c@ #2 }
64   {
65     \tl_if_empty:cF { cl@ #2 }
66     {
67       \tl_map_tokens:cn { cl@ #2 }
68       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
69     }
70   }
71 }
72 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
73 {
74   \str_if_eq:nnT {#2} {#3}
75   { \tl_map_break:n { \seq_map_break:n {#1} } }
76 }

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

77 \zref@newprop { zc@enclval }
78 {
79   \__zrefclever_get_enclosing_counters_value:e
80   \l__zrefclever_current_counter_tl
81 }
82 \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}`.

```

83 \tl_new:N \g__zrefclever_page_format_tl
84 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
85 \AddToHook { shipout / before }
86 {
87   \group_begin:
88   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
89   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
90   \group_end:

```

```

91 }
92 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
93 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

Still some other properties which we don't need to handle at the data provision side, but need to cater for at the retrieval side, are the ones from the `zref-xr` module, which are added to the labels imported from external documents, and needed to construct hyperlinks to them and to distinguish them from the current document ones at sorting and compressing: `urluse`, `url` and `externaldocument`.

4 Plumbing

4.1 Messages

```

94 \msg_new:nnn { zref-clever } { option-not-type-specific }
95 {
96   Option~'#1'~is-not-type-specific~\msg_line_context:~
97   Set~it~in~'\iow_char:N\zcLanguageSetup'~before-first~'type'
98   ~switch-or-as-package-option.
99 }
100 \msg_new:nnn { zref-clever } { option-only-type-specific }
101 {
102   No~type~specified~for~option~'#1'~\msg_line_context:~
103   Set~it~after~'type'~switch-or-in~'\iow_char:N\zcRefTypeSetup'.
104 }
105 \msg_new:nnn { zref-clever } { key-requires-value }
106 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:. }
107 \msg_new:nnn { zref-clever } { language-declared }
108 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing-to-do. }
109 \msg_new:nnn { zref-clever } { unknown-language-alias }
110 {
111   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
112   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
113   '\iow_char:N\zcDeclareLanguageAlias'.
114 }
115 \msg_new:nnn { zref-clever } { unknown-language-setup }
116 {
117   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
118   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
119   '\iow_char:N\zcDeclareLanguageAlias'.
120 }
121 \msg_new:nnn { zref-clever } { unknown-language-opt }
122 {
123   Language~'#1'~is~unknown~\msg_line_context:~Using~default.~
124   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
125   '\iow_char:N\zcDeclareLanguageAlias'.
126 }
127 \msg_new:nnn { zref-clever } { dict-loaded }
128 { Loaded~'#1'~dictionary. }
129 \msg_new:nnn { zref-clever } { dict-not-available }
130 { Dictionary~for~'#1'~not~available~\msg_line_context:. }
131 \msg_new:nnn { zref-clever } { unknown-language-load }
132 {

```

```

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

```

4.2 Data extraction

`_zrefclever_def_extract_default:Nnnn`

Extract property $\langle prop \rangle$ from $\langle label \rangle$ and sets variable $\langle tl var \rangle$ with extracted value. Ensure `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. In case the property is not found, set $\langle tl var \rangle$ with $\langle default \rangle$.

```

\__zrefclever_def_extract_default:Nnnn {\langle tl var \rangle}
{\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

174 \cs_new_protected:Npn \__zrefclever_def_extract_default:Nnnn #1#2#3#4
175 {
176   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
177   { \zref@extractdefault {#2} {#3} {#4} }

```



```

178 }
179 \cs_generate_variant:Nn \__zrefclever_def_extract_default:Nnnn { NVnn }

```

(End definition for __zrefclever_def_extract_default:Nnnn.)

__zrefclever_extract_default_unexp:nnn Extract property $\langle prop \rangle$ from $\langle label \rangle$. Ensure that, in the context of an x expansion, $\backslash\text{zref@extractdefault}$ is expanded exactly twice, but no further to retrieve the proper value. Thus, this is meant to be use in an x expansion context, not in other situations. In case the property is not found, leave $\langle default \rangle$ in the stream.

```

\__zrefclever_extract_default_unexp:nnn{\label}{\prop}{\default}

180 \cs_new:Npn \__zrefclever_extract_default_unexp:nnn #1#2#3
181 {
182   \exp_args:NNo \exp_args:No
183   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
184 }
185 \cs_generate_variant:Nn
186 \__zrefclever_extract_default_unexp:nnn { Vnn , nvnn , Vnn }

```

(End definition for __zrefclever_extract_default_unexp:nnn.)

__zrefclever_extract_default:nnn An internal version for $\backslash\text{zref@extractdefault}$.

```

\__zrefclever_extract_default:nnn{\label}{\prop}{\default}

187 \cs_new:Npn \__zrefclever_extract_default:nnn #1#2#3
188 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for __zrefclever_extract_default:nnn.)

4.3 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 $\backslash\text{__zrefclever_get_ref_string:nN}$, $\backslash\text{__zrefclever_get_ref_font:nN}$, and $\backslash\text{__zrefclever_type_name_setup}$: which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in $\backslash\text{g___zrefclever_fallback_dict_prop}$.

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

```

189 \tl_new:N \l__zrefclever_setup_type_tl
190 \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

191 \seq_const_from_clist:Nn
192 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
193 {

```

```

194     tpairsep ,
195     tlistsep ,
196     tlastsep ,
197     notesep ,
198 }
199 \seq_const_from_clist:Nn
200 \c__zrefclever_ref_options_possibly_type_specific_seq
201 {
202     namesep ,
203     pairsep ,
204     listsep ,
205     lastsep ,
206     rangesep ,
207     refpre ,
208     refpos ,
209     refpre-in ,
210     refpos-in ,
211 }

```

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:`.

```

212 \seq_const_from_clist:Nn
213 \c__zrefclever_ref_options_necessarily_type_specific_seq
214 {
215     Name-sg ,
216     name-sg ,
217     Name-pl ,
218     name-pl ,
219     Name-sg-ab ,
220     name-sg-ab ,
221     Name-pl-ab ,
222     name-pl-ab ,
223 }

```

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

```

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

```

```

242 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
243 \c__zrefclever_ref_options_reference_seq
244 \c__zrefclever_ref_options_font_seq

```

(End definition for \c__zrefclever_ref_options_necessarily_not_type_specific_seq and others.)

4.4 Languages

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

```

245 \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}

246 \NewDocumentCommand \zcDeclareLanguage { m }
247 {
248   \tl_if_empty:nF {#1}
249   {
250     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
251     { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
252     { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
253   }
254 }
255 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

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

```

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

256 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
257 {
258   \tl_if_empty:nF {#1}
259   {
260     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
261     {
262       \exp_args:NNnx
263       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
264       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
265     }
266     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
267   }
268 }
269 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for `\zcDeclareLanguageAlias`.)

4.5 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

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

270 `\seq_new:N \g__zrefclever_loaded_dictionaries_seq`

(End definition for \g__zrefclever_loaded_dictionaries_seq.)

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

271 \bool_new:N \l__zrefclever_load_dict_verbose_bool

(End definition for \l__zrefclever_load_dict_verbose_bool.)

__zrefclever_provide_dictionary:n Load dictionary for known $\langle language \rangle$ if it is available and if it has not already been loaded.

```

\__zrefclever_provide_dictionary:n {\langle language \rangle}
272 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
273 {
274   \group_begin:
275   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
276   \l__zrefclever_dict_language_tl
277   {
278     \seq_if_in:NVF
279     \g__zrefclever_loaded_dictionaries_seq
280     \l__zrefclever_dict_language_tl
281     {
282       \exp_args:Nx \file_get:nnNTF
283       { zref-clever- \l__zrefclever_dict_language_tl .dict }
284       { \ExplSyntaxOn }
285       \l_tmpa_tl
286       {
287         \prop_if_exist:cF
288         {
289           g__zrefclever_dict_
290           \l__zrefclever_dict_language_tl _prop
291         }
292         {
293           \prop_new:c
294           {
295             g__zrefclever_dict_
296             \l__zrefclever_dict_language_tl _prop
297           }
298         }
299         \tl_clear:N \l__zrefclever_setup_type_tl
300         \exp_args:NnV
301         \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
302         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
303         \l__zrefclever_dict_language_tl
304         \msg_note:nnx { zref-clever } { dict-loaded }
305         { \l__zrefclever_dict_language_tl }
306       }
307     }
308     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
309     {
310       \msg_warning:nnx { zref-clever } { dict-not-available }
311       { \l__zrefclever_dict_language_tl }
312     }

```

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.

```

313             \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
314             \l__zrefclever_dict_language_tl
315         }
316     }
317 }
318 {
319     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
320     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
321 }
322 \group_end:
323 }
324 \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>}
325 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
326 {
327     \group_begin:
328     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
329     \__zrefclever_provide_dictionary:n {#1}
330     \group_end:
331 }
332 \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 `<key>` and `<translation>` 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>}
333 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
334 {
335     \exp_args:Nnx \prop_gput_if_new:cnn
336     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
337     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
338 }
339 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2

```

```

340 {
341   \prop_gput_if_new:cnn
342   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
343   { default- #1 } {#2}
344 }

```

(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.

```

345 \keys_define:nn { zref-clever / dictionary }
346 {
347   type .code:n =
348   {
349     \tl_if_empty:NTF {#1}
350     { \tl_clear:N \l__zrefclever_setup_type_tl }
351     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
352   } ,
353 }
354 \seq_map_inline:Nn
355 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
356 {
357   \keys_define:nn { zref-clever / dictionary }
358   {
359     #1 .value_required:n = true ,
360     #1 .code:n =
361     {
362       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
363       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
364       {
365         \msg_info:nnn { zref-clever }
366         { option-not-type-specific } {#1}
367       }
368     } ,
369   }
370 }
371 \seq_map_inline:Nn
372 \c__zrefclever_ref_options_possibly_type_specific_seq
373 {
374   \keys_define:nn { zref-clever / dictionary }
375   {
376     #1 .value_required:n = true ,
377     #1 .code:n =
378     {
379       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
380       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
381       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
382     } ,
383   }
384 }
385 \seq_map_inline:Nn

```

```

386 \c__zrefclever_ref_options_necessarily_type_specific_seq
387 {
388   \keys_define:nn { zref-clever / dictionary }
389   {
390     #1 .value_required:n = true ,
391     #1 .code:n =
392     {
393       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
394       {
395         \msg_info:nnn { zref-clever }
396         { option-only-type-specific } {#1}
397       }
398       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
399     } ,
400   }
401 }

```

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.

```

402 \prop_new:N \g__zrefclever_fallback_dict_prop
403 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
404 {
405   tpairsep = {,~} ,
406   tlistsep = {,~} ,
407   tlastsep = {,~} ,
408   notesep = {~} ,
409   namesep = {\nobreakspace} ,
410   pairsep = {,~} ,
411   listsep = {,~} ,
412   lastsep = {,~} ,
413   rangesep = {\textendash} ,
414   refpre = {} ,
415   refpos = {} ,
416   refpre-in = {} ,
417   refpos-in = {} ,
418 }

```

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 should not be relied upon.

```

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

419 \prg_new_protected_conditional:Npnn
420 \_zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
421 {
422   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
423   \l__zrefclever_dict_language_tl
424   {
425     \prop_get:cnNTF
426     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
427     { type- #2 - #3 } #4
428     { \prg_return_true: }
429     { \prg_return_false: }
430   }
431   { \prg_return_false: }
432 }
433 \prg_generate_conditional_variant:Nnn
434 \_zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }

```

(End definition for _zrefclever_get_type_transl:nnnNF.)

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

```

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

435 \prg_new_protected_conditional:Npnn
436 \_zrefclever_get_default_transl:nnN #1#2#3 { F }
437 {
438   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
439   \l__zrefclever_dict_language_tl
440   {
441     \prop_get:cnNTF
442     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
443     { default- #2 } #3
444     { \prg_return_true: }
445     { \prg_return_false: }
446   }
447   { \prg_return_false: }
448 }
449 \prg_generate_conditional_variant:Nnn
450 \_zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for _zrefclever_get_default_transl:nnNF.)

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

```

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

```

```

451 % {<key>><tl var to set>
452 \prg_new_protected_conditional:Npnn
453 \__zrefclever_get_fallback_transl:nN #1#2 { F }
454 {
455   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
456     { #1 } #2
457     { \prg_return_true: }
458     { \prg_return_false: }
459 }

```

(End definition for `__zrefclever_get_fallback_transl:nNF`.)

4.6 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>}

460 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
461 {
462   \tl_if_empty:nTF {#3}
463     { \prop_remove:Nn #1 {#2} }
464     { \prop_put:Nnn #1 {#2} {#3} }
465 }

```

(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.

```

466 \tl_new:N \l__zrefclever_ref_property_tl
467 \keys_define:nn { zref-clever / reference }
468 {
469   ref .choice: ,
470   ref / default .code:n =
471     { \tl_set:Nn \l__zrefclever_ref_property_tl { default } } ,
472   ref / zc@thecnt .code:n =
473     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
474   ref / page .code:n =
475     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
476   ref / title .code:n =

```

```

477 {
478   \AddToHook { begindocument }
479   {
480     \@ifpackageloaded { zref-titleref }
481     { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
482     {
483       \msg_warning:nn { zref-clever } { missing-zref-titleref }
484       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
485     }
486   }
487 },
488 ref .initial:n = default ,
489 ref .default:n = default ,
490 page .meta:n = { ref = page },
491 page .value_forbidden:n = true ,
492 }
493 \AddToHook { begindocument }
494 {
495   \@ifpackageloaded { zref-titleref }
496   {
497     \keys_define:nn { zref-clever / reference }
498     {
499       ref / title .code:n =
500       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
501     }
502   }
503   {
504     \keys_define:nn { zref-clever / reference }
505     {
506       ref / title .code:n =
507       {
508         \msg_warning:nn { zref-clever } { missing-zref-titleref }
509         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
510       }
511     }
512   }
513 }

```

typeset option

```

514 \bool_new:N \l__zrefclever_typeset_ref_bool
515 \bool_new:N \l__zrefclever_typeset_name_bool
516 \keys_define:nn { zref-clever / reference }
517 {
518   typeset .choice: ,
519   typeset / both .code:n =
520   {
521     \bool_set_true:N \l__zrefclever_typeset_ref_bool
522     \bool_set_true:N \l__zrefclever_typeset_name_bool
523   } ,
524   typeset / ref .code:n =
525   {
526     \bool_set_true:N \l__zrefclever_typeset_ref_bool
527     \bool_set_false:N \l__zrefclever_typeset_name_bool

```

```

528     } ,
529     typeset / name .code:n =
530     {
531         \bool_set_false:N \l__zrefclever_typeset_ref_bool
532         \bool_set_true:N \l__zrefclever_typeset_name_bool
533     } ,
534     typeset .initial:n = both ,
535     typeset .value_required:n = true ,
536
537     noname .meta:n = { typeset = ref } ,
538     noname .value_forbidden:n = true ,
539 }

```

sort option

```

540 \bool_new:N \l__zrefclever_typeset_sort_bool
541 \keys_define:nn { zref-clever / reference }
542 {
543     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
544     sort .initial:n = true ,
545     sort .default:n = true ,
546     nosort .meta:n = { sort = false } ,
547     nosort .value_forbidden:n = true ,
548 }

```

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.

```

549 \seq_new:N \l__zrefclever_typesort_seq
550 \keys_define:nn { zref-clever / reference }
551 {
552     typesort .code:n =
553     {
554         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
555         \seq_reverse:N \l__zrefclever_typesort_seq
556     } ,
557     typesort .initial:n =
558     { part , chapter , section , paragraph } ,
559     typesort .value_required:n = true ,
560     notypesort .code:n =
561     { \seq_clear:N \l__zrefclever_typesort_seq } ,
562     notypesort .value_forbidden:n = true ,
563 }

```

comp option

```

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

```

```

570     nocomp .meta:n = { comp = false },
571     nocomp .value_forbidden:n = true ,
572 }

```

range option

```

573 \bool_new:N \l__zrefclever_typeset_range_bool
574 \keys_define:nn { zref-clever / reference }
575 {
576     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
577     range .initial:n = false ,
578     range .default:n = true ,
579 }

```

cap and capfirst options

```

580 \bool_new:N \l__zrefclever_capitalize_bool
581 \bool_new:N \l__zrefclever_capitalize_first_bool
582 \keys_define:nn { zref-clever / reference }
583 {
584     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
585     cap .initial:n = false ,
586     cap .default:n = true ,
587     nocap .meta:n = { cap = false },
588     nocap .value_forbidden:n = true ,
589
590     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
591     capfirst .initial:n = false ,
592     capfirst .default:n = true ,
593 }

```

abbrev and noabbrevfirst options

```

594 \bool_new:N \l__zrefclever_abbrev_bool
595 \bool_new:N \l__zrefclever_noabbrev_first_bool
596 \keys_define:nn { zref-clever / reference }
597 {
598     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
599     abbrev .initial:n = false ,
600     abbrev .default:n = true ,
601     noabbrev .meta:n = { abbrev = false },
602     noabbrev .value_forbidden:n = true ,
603
604     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
605     noabbrevfirst .initial:n = false ,
606     noabbrevfirst .default:n = true ,
607 }

```

S option

```

608 \keys_define:nn { zref-clever / reference }
609 {
610     S .meta:n =
611         { capfirst = true , noabbrevfirst = true },
612     S .value_forbidden:n = true ,
613 }

```

hyperref option

```

614 \bool_new:N \l__zrefclever_use_hyperref_bool
615 \bool_new:N \l__zrefclever_warn_hyperref_bool
616 \keys_define:nn { zref-clever / reference }
617 {
618   hyperref .choice: ,
619   hyperref / auto .code:n =
620   {
621     \bool_set_true:N \l__zrefclever_use_hyperref_bool
622     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
623   } ,
624   hyperref / true .code:n =
625   {
626     \bool_set_true:N \l__zrefclever_use_hyperref_bool
627     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
628   } ,
629   hyperref / false .code:n =
630   {
631     \bool_set_false:N \l__zrefclever_use_hyperref_bool
632     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
633   } ,
634   hyperref .initial:n = auto ,
635   hyperref .default:n = auto
636 }
637 \AddToHook { begindocument }
638 {
639   \@ifpackageloaded { hyperref }
640   {
641     \bool_if:NT \l__zrefclever_use_hyperref_bool
642     { \RequirePackage { zref-hyperref } }
643   }
644   {
645     \bool_if:NT \l__zrefclever_warn_hyperref_bool
646     { \msg_warning:nn { zref-clever } { missing-hyperref } }
647     \bool_set_false:N \l__zrefclever_use_hyperref_bool
648   }
649   \keys_define:nn { zref-clever / reference }
650   {
651     hyperref .code:n =
652     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
653   }
654 }

```

nameinlink option

```

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

```

```

666     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
667     nameinlink .initial:n = tsingle ,
668     nameinlink .default:n = true ,
669 }

```

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`.

```

670 \tl_new:N \l__zrefclever_ref_language_tl
671 \tl_new:N \l__zrefclever_main_language_tl
672 \tl_new:N \l__zrefclever_current_language_tl
673 \AddToHook { begindocument }
674 {
675   \@ifpackageloaded { babel }
676   {
677     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
678     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
679   }
680   {
681     \@ifpackageloaded { polyglossia }
682     {
683       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
684       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
685     }
686     {
687       \tl_set:Nn \l__zrefclever_current_language_tl { english }
688       \tl_set:Nn \l__zrefclever_main_language_tl { english }
689     }
690   }

```

690 }

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.

```

691        \tl_set:Nn \l__zrefclever_ref_language_tl
692            { \l__zrefclever_main_language_tl }
693        }

694 \keys_define:nn { zref-clever / reference }
695 {
696    lang .code:n =
697    {
698      \AddToHook { begindocument }
699      {
700        \str_case:nnF {#1}
701        {
702          { main }
703          {
704            \tl_set:Nn \l__zrefclever_ref_language_tl
705                { \l__zrefclever_main_language_tl }
706            \__zrefclever_provide_dictionary_verbosely:x
707                { \l__zrefclever_ref_language_tl }
708          }
709          { current }
710          {
711            \tl_set:Nn \l__zrefclever_ref_language_tl
712                { \l__zrefclever_current_language_tl }
713            \__zrefclever_provide_dictionary_verbosely:x
714                { \l__zrefclever_ref_language_tl }
715          }
716        }
717      }
718      {
719        \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
720        {
721          \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
722        }
723        {
724          \msg_warning:nnn { zref-clever }
725            { unknown-language-opt } {#1}
726          \tl_set:Nn \l__zrefclever_ref_language_tl
727            { \l__zrefclever_main_language_tl }
728        }
729        \__zrefclever_provide_dictionary_verbosely:x
730        { \l__zrefclever_ref_language_tl }
731      }
732    }
733    ,
734    lang .value_required:n = true ,
735    }

736 \AddToHook { begindocument / before }

```



```

737 {
738   \AddToHook { begindocument }
739   {

```

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.

```

740     \__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.

```

741     \keys_define:nn { zref-clever / reference }
742     {
743       lang .code:n =
744       {
745         \str_case:nnF {#1}
746         {
747           { main }
748           {
749             \tl_set:Nn \l__zrefclever_ref_language_tl
750             { \l__zrefclever_main_language_tl }
751             \__zrefclever_provide_dictionary:x
752             { \l__zrefclever_ref_language_tl }
753           }
754
755           { current }
756           {
757             \tl_set:Nn \l__zrefclever_ref_language_tl
758             { \l__zrefclever_current_language_tl }
759             \__zrefclever_provide_dictionary:x
760             { \l__zrefclever_ref_language_tl }
761           }
762         }
763       {
764         \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
765         {
766           \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
767         }
768         {
769           \msg_warning:nnn { zref-clever }
770           { unknown-language-opt } {#1}
771           \tl_set:Nn \l__zrefclever_ref_language_tl
772           { \l__zrefclever_main_language_tl }
773         }
774         \__zrefclever_provide_dictionary:x
775         { \l__zrefclever_ref_language_tl }
776       }
777     } ,
778     lang .value_required:n = true ,
779   }
780 }
781 }

```

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`.

```
782 \tl_new:N \l__zrefclever_ref_typeset_font_tl
783 \keys_define:nn { zref-clever / reference }
784 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```
785 \keys_define:nn { zref-clever / reference }
786 {
787     titleref .code:n = { \RequirePackage { zref-titleref } } ,
788     titleref .value_forbidden:n = true ,
789 }
790 \AddToHook { begindocument }
791 {
792     \keys_define:nn { zref-clever / reference }
793     {
794         titleref .code:n =
795         { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
796     }
797 }
```

note option

```
798 \tl_new:N \l__zrefclever_zcref_note_tl
799 \keys_define:nn { zref-clever / reference }
800 {
801     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
802     note .value_required:n = true ,
803 }
```

check option

Integration with `zref-check`.

```
804 \bool_new:N \l__zrefclever_zrefcheck_available_bool
805 \bool_new:N \l__zrefclever_zcref_with_check_bool
806 \keys_define:nn { zref-clever / reference }
807 {
808     check .code:n = { \RequirePackage { zref-check } } ,
809     check .value_forbidden:n = true ,
810 }
811 \AddToHook { begindocument }
812 {
813     \@ifpackageloaded { zref-check }
814     {
815         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
816         \keys_define:nn { zref-clever / reference }
817         {
818             check .code:n =
819             {
820                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
821                 \keys_set:nn { zref-check / zcheck } {#1}

```

```

822         } ,
823         check .value_required:n = true ,
824     }
825 }
826 {
827     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
828     \keys_define:nn { zref-clever / reference }
829     {
830         check .value_forbidden:n = false ,
831         check .code:n =
832             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
833     }
834 }
835 }

```

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`.

```

836 \prop_new:N \l__zrefclever_counter_type_prop
837 \keys_define:nn { zref-clever / label }
838 {
839     countertype .code:n =
840     {
841         \keyval_parse:nnn
842         {
843             \msg_warning:nnnn { zref-clever }
844             { key-requires-value } { countertype }
845         }
846         {
847             \__zrefclever_prop_put_non_empty:Nnn
848             \l__zrefclever_counter_type_prop
849         }
850         {#1}
851     } ,
852     countertype .value_required:n = true ,
853     countertype .initial:n =
854     {
855         subsection      = section ,
856         subsubsection    = section ,
857         subparagraph     = paragraph ,
858         enumi            = item ,
859         enumii           = item ,
860         enumiii          = item ,
861         enumiv           = item ,
862         mpfootnote       = footnote ,
863     } ,
864 }

```

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclval` property, 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.

```
865 \seq_new:N \l__zrefclever_counter_resetters_seq
866 \keys_define:nn { zref-clever / label }
867 {
868   counterresetters .code:n =
869   {
870     \clist_map_inline:nn {##1}
871     {
872       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
873       {
874         \seq_put_right:Nn
875         \l__zrefclever_counter_resetters_seq {##1}
876       }
877     }
878   } ,
879   counterresetters .initial:n =
880   {
881     part ,
882     chapter ,
883     section ,
884     subsection ,
885     subsubsection ,
886     paragraph ,
887     subparagraph ,
888   },
889   counterresetters .value_required:n = true ,
890 }
```

counterresetby option

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

```
891 \prop_new:N \l__zrefclever_counter_resetby_prop
892 \keys_define:nn { zref-clever / label }
893 {
894   counterresetby .code:n =
895   {
896     \keyval_parse:nnn
897     {
898       \msg_warning:nnn { zref-clever }
899       { key-requires-value } { counterresetby }
900     }
901   }
```

```

901         {
902             \_zrefclever_prop_put_non_empty:Nnn
903             \l_zrefclever_counter_resetby_prop
904         }
905         {#1}
906     } ,
907     counterresetby .value_required:n = true ,
908     counterresetby .initial:n =
909     {

```

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.

```

910         enumii = enumi ,
911         enumiii = enumii ,
912         enumiv = enumiii ,
913     } ,
914 }

```

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.

```

915 \tl_new:N \l_zrefclever_current_counter_tl
916 \keys_define:nn { zref-clever / label }
917 {
918     currentcounter .tl_set:N = \l_zrefclever_current_counter_tl ,
919     currentcounter .value_required:n = true ,
920     currentcounter .initial:n = \@currentcounter ,
921 }

```

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.

```

922 \prop_new:N \l_zrefclever_ref_options_prop
923 \seq_map_inline:Nn
924     \c_zrefclever_ref_options_reference_seq
925     {
926         \keys_define:nn { zref-clever / reference }
927         {

```

```

928     #1 .default:V = \c_novalue_tl ,
929     #1 .code:n =
930     {
931         \tl_if_novalue:nTF {##1}
932         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
933         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
934     } ,
935 }
936 }

```

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.

```

937 \keys_define:nn { }
938 {
939     zref-clever / zcsetup .inherit:n =
940     {
941         zref-clever / label ,
942         zref-clever / reference ,
943     }
944 }

```

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

```

945 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{\options}

```

```

946 \NewDocumentCommand \zcsetup { m }
947 { \__zrefclever_zcsetup:n {#1} }

```

(End definition for `\zcsetup`.)

`__zrefclever_zcsetup:n` A version of `\zcsetup` for internal use with variant.

```

\__zrefclever_zcsetup:n{\options}

```

```

948 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
949 { \keys_set:nn { zref-clever / zcsetup } {#1} }
950 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End definition for `__zrefclever_zcsetup:n`.)

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⟩}
951 \NewDocumentCommand \zcRefTypeSetup { m m }
952 {
953   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
954   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
955   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
956   \keys_set:nn { zref-clever / typesetup } {#2}
957 }
```

(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 4.6), 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>.

```
958 \seq_map_inline:Nn
959   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
960   {
961     \keys_define:nn { zref-clever / typesetup }
962     {
963       #1 .code:n =
964       {
965         \msg_warning:nnn { zref-clever }
966         { option-not-type-specific } {#1}
967       } ,
968     }
969   }
970 \seq_map_inline:Nn
971   \c__zrefclever_ref_options_typesetup_seq
972   {
973     \keys_define:nn { zref-clever / typesetup }
974     {
975       #1 .default:V = \c_novaluel_tl ,
```

```

976     #1 .code:n =
977     {
978         \tl_if_novalue:nTF {##1}
979         {
980             \prop_remove:cn
981             {
982                 l__zrefclever_type_
983                 \l__zrefclever_setup_type_tl _options_prop
984             }
985             {#1}
986         }
987         {
988             \prop_put:cnn
989             {
990                 l__zrefclever_type_
991                 \l__zrefclever_setup_type_tl _options_prop
992             }
993             {#1} {##1}
994         }
995     } ,
996 }
997 }

```

5.3 \zcLanguageSetup

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

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
998 \NewDocumentCommand \zcLanguageSetup { m m }
999 {
1000     \group_begin:
1001     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1002     \l__zrefclever_dict_language_tl
1003     {
1004         \tl_clear:N \l__zrefclever_setup_type_tl
1005         \keys_set:nn { zref-clever / langsetup } {#2}
1006     }
1007     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1008     \group_end:
1009 }
1010 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

```

\__zrefclever_declare_type_transl:nnnn
\__zrefclever_declare_default_transl:nnn

```

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


```

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

1011 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1012 {
1013   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1014     { type- #2 - #3 } {#4}
1015 }
1016 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
1017 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
1018 {
1019   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1020     { default- #2 } {#3}
1021 }
1022 \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.

```

1023 \keys_define:nn { zref-clever / langsetup }
1024 {
1025   type .code:n =
1026   {
1027     \tl_if_empty:NTF {#1}
1028       { \tl_clear:N \l__zrefclever_setup_type_tl }
1029       { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1030   } ,
1031 }
1032 \seq_map_inline:Nn
1033   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1034   {
1035     \keys_define:nn { zref-clever / langsetup }
1036     {
1037       #1 .value_required:n = true ,
1038       #1 .code:n =
1039       {
1040         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1041         {
1042           \_zrefclever_declare_default_transl:Vnn
1043             \l__zrefclever_dict_language_tl
1044             {#1} {##1}
1045         }
1046         {
1047           \msg_warning:nnn { zref-clever }
1048             { option-not-type-specific } {#1}
1049         }
1050       } ,
1051     }
1052   }
1053 \seq_map_inline:Nn
1054   \c__zrefclever_ref_options_possibly_type_specific_seq

```

```

1055 {
1056   \keys_define:nn { zref-clever / langsetup }
1057   {
1058     #1 .value_required:n = true ,
1059     #1 .code:n =
1060     {
1061       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1062       {
1063         \__zrefclever_declare_default_transl:Vnn
1064         \l__zrefclever_dict_language_tl
1065         {#1} {##1}
1066       }
1067       {
1068         \__zrefclever_declare_type_transl:Vnn
1069         \l__zrefclever_dict_language_tl
1070         \l__zrefclever_setup_type_tl
1071         {#1} {##1}
1072       }
1073     } ,
1074   }
1075 }
1076 \seq_map_inline:Nn
1077 \c__zrefclever_ref_options_necessarily_type_specific_seq
1078 {
1079   \keys_define:nn { zref-clever / langsetup }
1080   {
1081     #1 .value_required:n = true ,
1082     #1 .code:n =
1083     {
1084       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1085       {
1086         \msg_warning:nnn { zref-clever }
1087         { option-only-type-specific } {#1}
1088       }
1089       {
1090         \__zrefclever_declare_type_transl:Vnn
1091         \l__zrefclever_dict_language_tl
1092         \l__zrefclever_setup_type_tl
1093         {#1} {##1}
1094       }
1095     } ,
1096   }
1097 }

```

6 User interface

6.1 \zcref

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

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

```

1098 \NewDocumentCommand \zcref { s O { } m }
1099 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for \zcref.)

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

```
\__zrefclever_zcref:nnnn {\langle labels \rangle} {\langle * \rangle} {\langle options \rangle}
```

```
1100 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1101 {
1102   \group_begin:
```

Set options.

```
1103   \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```
1104   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1105   \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.

```
1106   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Integration with zref-check.

```
1107   \bool_lazy_and:nnT
1108     { \l__zrefclever_zrefcheck_available_bool }
1109     { \l__zrefclever_zcref_with_check_bool }
1110     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1111   \bool_lazy_or:nnT
1112     { \l__zrefclever_typeset_sort_bool }
1113     { \l__zrefclever_typeset_range_bool }
1114     { \__zrefclever_sort_labels: }
```

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

```
1115   \group_begin:
1116   \l__zrefclever_ref_typeset_font_tl
1117   \__zrefclever_typeset_refs:
1118   \group_end:
```

Typeset note.

```
1119   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1120   {
1121     \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1122     \l_tmpa_tl
1123     \l__zrefclever_zcref_note_tl
1124   }
```

Integration with zref-check.

```
1125   \bool_lazy_and:nnT
1126     { \l__zrefclever_zrefcheck_available_bool }
1127     { \l__zrefclever_zcref_with_check_bool }
1128   {
```

```

1129         \zrefcheck_zcref_end_label_maybe:
1130         \zrefcheck_zcref_run_checks_on_labels:n
1131         { \l__zrefclever_zcref_labels_seq }
1132     }
1133     \group_end:
1134 }

```

(End definition for `__zrefclever_zcref:n`.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1135 \seq_new:N \l__zrefclever_zcref_labels_seq
1136 \bool_new:N \l__zrefclever_link_star_bool

```

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

6.2 `\zcpageref`

`\zcpageref` A `\pageref` equivalent of `\zcref`.

```

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

```

```

1137 \NewDocumentCommand \zcpageref { s O { } m }
1138 {
1139     \IfBooleanTF {#1}
1140     { \zcref*[#2, ref = page] {#3} }
1141     { \zcref [ #2, ref = page] {#3} }
1142 }

```

(End definition for `\zcpageref`.)

7 Sorting

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

```

\l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl
\l__zrefclever_label_extdoc_a_tl
\l__zrefclever_label_extdoc_b_tl

```

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.

```

1143 \tl_new:N \l__zrefclever_label_type_a_tl
1144 \tl_new:N \l__zrefclever_label_type_b_tl
1145 \tl_new:N \l__zrefclever_label_enclval_a_tl
1146 \tl_new:N \l__zrefclever_label_enclval_b_tl
1147 \tl_new:N \l__zrefclever_label_extdoc_a_tl
1148 \tl_new:N \l__zrefclever_label_extdoc_b_tl

```

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

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

```
1149 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for `\l_zrefclever_sort_decided_bool`.)

`\l_zrefclever_sort_prior_a_int` Auxiliary variables for `__zrefclever_sort_default_different_types:nn`. Store the sort priority of the “current” and “next” labels.

`\l_zrefclever_sort_prior_b_int`

```
1150 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
1151 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for `\l_zrefclever_sort_prior_a_int` and `\l_zrefclever_sort_prior_b_int`.)

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

```
1152 \seq_new:N \l__zrefclever_label_types_seq
```

(End definition for `\l_zrefclever_label_types_seq`.)

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

```
1153 \cs_new_protected:Npn \__zrefclever_sort_labels:
```

```
1154 {
```

Store label types sequence.

```
1155 \seq_clear:N \l__zrefclever_label_types_seq
```

```
1156 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
```

```
1157 {
```

```
1158 \seq_map_function:NN \l__zrefclever_zcref_labels_seq
```

```
1159 \__zrefclever_label_type_put_new_right:n
```

```
1160 }
```

Sort.

```
1161 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
```

```
1162 {
```

```
1163 \zref@ifrefundefined {##1}
```

```
1164 {
```

```
1165 \zref@ifrefundefined {##2}
```

```
1166 {
```

```
1167 % Neither label is defined.
```

```
1168 \sort_return_same:
```

```
1169 }
```

```
1170 {
```

```
1171 % The second label is defined, but the first isn't, leave the
```

```
1172 % undefined first (to be more visible).
```

```
1173 \sort_return_same:
```

```
1174 }
```

```
1175 }
```

```

1176     {
1177         \zref@ifrefundefined {##2}
1178         {
1179             % The first label is defined, but the second isn't, bring the
1180             % second forward.
1181             \sort_return_swapped:
1182         }
1183         {
1184             % The interesting case: both labels are defined. References
1185             % to the "default" property or to the "page" are quite
1186             % different with regard to sorting, so we branch them here to
1187             % specialized functions.
1188             \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1189             { \__zrefclever_sort_page:nn {##1} {##2} }
1190             { \__zrefclever_sort_default:nn {##1} {##2} }
1191         }
1192     }
1193 }
1194 }

```

(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>}

1195 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1196 {
1197     \__zrefclever_def_extract_default:Nnnn
1198     \l__zrefclever_label_type_a_tl {#1} {zc@type} { \c_empty_tl }
1199     \seq_if_in:NVF \l__zrefclever_label_types_seq
1200     \l__zrefclever_label_type_a_tl
1201     {
1202         \seq_put_right:NV \l__zrefclever_label_types_seq
1203         \l__zrefclever_label_type_a_tl
1204     }
1205 }

```

(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>}

```

```

1206 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1207 {
1208   \__zrefclever_def_extract_default:Nnnn
1209   \l__zrefclever_label_type_a_tl {#1} {zc@type} { \c_empty_tl }
1210   \__zrefclever_def_extract_default:Nnnn
1211   \l__zrefclever_label_type_b_tl {#2} {zc@type} { \c_empty_tl }
1212
1213   \bool_if:nTF
1214   {
1215     % The second label has a type, but the first doesn't, leave the
1216     % undefined first (to be more visible).
1217     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1218     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1219   }
1220   { \sort_return_same: }
1221   {
1222     \bool_if:nTF
1223     {
1224       % The first label has a type, but the second doesn't, bring the
1225       % second forward.
1226       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1227       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1228     }
1229     { \sort_return_swapped: }
1230     {
1231       \bool_if:nTF
1232       {
1233         % The interesting case: both labels have a type...
1234         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1235         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1236       }
1237       {
1238         \tl_if_eq:NNTF
1239         \l__zrefclever_label_type_a_tl
1240         \l__zrefclever_label_type_b_tl
1241         % ...and it's the same type.
1242         { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1243         % ...and they are different types.
1244         { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1245       }
1246     }
1247     % Neither label has a type. We can't do much of meaningful
1248     % here, but if it's the same counter, compare it.
1249     \exp_args:Nxx \tl_if_eq:nnTF
1250     {
1251       \__zrefclever_extract_default_unexp:nnn
1252       {#1} {zc@counter} { }
1253     }
1254     {
1255       \__zrefclever_extract_default_unexp:nnn
1256       {#2} {zc@counter} { }
1257     }
1258     {
1259       \int_compare:nNnTF

```

```

1260         {
1261             \_zrefclever_extract_default:nnn
1262             {#1} { zc@cntval } { -1 }
1263         }
1264         >
1265         {
1266             \_zrefclever_extract_default:nnn
1267             {#2} { zc@cntval } { -1 }
1268         }
1269         { \sort_return_swapped: }
1270         { \sort_return_same: }
1271     }
1272     { \sort_return_same: }
1273 }
1274 }
1275 }
1276 }

```

(End definition for _zrefclever_sort_default:nn.)

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

```

1277 \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>}
1278 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1279 {
1280     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
1281     {#1} { zc@enclval } { \c_empty_tl }
1282     \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1283     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
1284     {#2} { zc@enclval } { \c_empty_tl }
1285     \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1286     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
1287     {#1} { externaldocument } { \c_empty_tl }
1288     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
1289     {#2} { externaldocument } { \c_empty_tl }
1290
1291     \bool_set_false:N \l__zrefclever_sort_decided_bool
1292
1293     % First we check if there's any "external document" difference (coming
1294     % from 'zref-xr') and, if so, sort based on that.
1295     \tl_if_eq:NNF
1296     \l__zrefclever_label_extdoc_a_tl
1297     \l__zrefclever_label_extdoc_b_tl
1298     {
1299         \bool_if:nTF
1300         {
1301             \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1302             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1303         }
1304         {
1305             \bool_set_true:N \l__zrefclever_sort_decided_bool
1306             \sort_return_same:
1307         }

```



```

1308     {
1309         \bool_if:nTF
1310         {
1311             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1312             \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1313         }
1314         {
1315             \bool_set_true:N \l__zrefclever_sort_decided_bool
1316             \sort_return_swapped:
1317         }
1318         {
1319             \bool_set_true:N \l__zrefclever_sort_decided_bool
1320             % Two different "external documents": last resort, sort by the
1321             % document name itself.
1322             \str_compare:eNeTF
1323             { \l__zrefclever_label_extdoc_b_tl } <
1324             { \l__zrefclever_label_extdoc_a_tl }
1325             { \sort_return_swapped: }
1326             { \sort_return_same: }
1327         }
1328     }
1329 }
1330
1331 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1332 {
1333     \bool_if:nTF
1334     {
1335         % Both are empty: neither label has any (further) "enclosing
1336         % counters" (left).
1337         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1338         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1339     }
1340     {
1341         \bool_set_true:N \l__zrefclever_sort_decided_bool
1342         \int_compare:nNnTF
1343         { \__zrefclever_extract_default:nnn {#1} { zc@cntval } { -1 } }
1344         >
1345         { \__zrefclever_extract_default:nnn {#2} { zc@cntval } { -1 } }
1346         { \sort_return_swapped: }
1347         { \sort_return_same: }
1348     }
1349     {
1350         \bool_if:nTF
1351         {
1352             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1353             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1354         }
1355         {
1356             \bool_set_true:N \l__zrefclever_sort_decided_bool
1357             \int_compare:nNnTF
1358             { \__zrefclever_extract_default:nnn {#1} { zc@cntval } { } }
1359             >
1360             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1361             { \sort_return_swapped: }

```

```

1362         { \sort_return_same:      }
1363     }
1364     {
1365     \bool_if:nTF
1366     {
1367         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1368         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1369     }
1370     {
1371         \bool_set_true:N \l__zrefclever_sort_decided_bool
1372         \int_compare:nNnTF
1373         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1374         <
1375         {
1376             \__zrefclever_extract_default:nnn
1377             {#2} { zc@cntval } { }
1378         }
1379         { \sort_return_same:      }
1380         { \sort_return_swapped: }
1381     }
1382     {
1383         % Neither is empty: we can compare the values of the
1384         % current enclosing counter in the loop, if they are
1385         % equal, we are still in the loop, if they are not, a
1386         % sorting decision can be made directly.
1387         \int_compare:nNnTF
1388         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1389         =
1390         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1391         {
1392             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1393             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1394             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1395             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1396         }
1397         {
1398             \bool_set_true:N \l__zrefclever_sort_decided_bool
1399             \int_compare:nNnTF
1400             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1401             >
1402             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1403             { \sort_return_swapped: }
1404             { \sort_return_same:      }
1405         }
1406     }
1407 }
1408 }
1409 }
1410 }

```

(End definition for __zrefclever_sort_default_same_type:nn.)

```

__zrefclever_sort_default_different_types:nn      \__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
1411 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2

```

```

1412 {
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”.
1413 \int_zero:N \l__zrefclever_sort_prior_a_int
1414 \int_zero:N \l__zrefclever_sort_prior_b_int
1415 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1416 {
1417   \tl_if_eq:nnTF {##2} {{othertypes}}
1418   {
1419     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1420     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1421     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1422     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1423   }
1424   {
1425     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1426     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1427     {
1428       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1429       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1430     }
1431   }
1432 }

```

Then do the actual sorting.

```

1433 \bool_if:nTF
1434 {
1435   \int_compare_p:nNn
1436   { \l__zrefclever_sort_prior_a_int } <
1437   { \l__zrefclever_sort_prior_b_int }
1438 }
1439 { \sort_return_same: }
1440 {
1441   \bool_if:nTF
1442   {
1443     \int_compare_p:nNn
1444     { \l__zrefclever_sort_prior_a_int } >
1445     { \l__zrefclever_sort_prior_b_int }
1446   }
1447   { \sort_return_swapped: }
1448   {
1449     % Sort priorities are equal: the type that occurs first in
1450     % ‘labels’, as given by the user, is kept (or brought) forward.
1451     \seq_map_inline:Nn \l__zrefclever_label_types_seq
1452     {
1453       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1454       { \seq_map_break:n { \sort_return_same: } }
1455       {
1456         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1457         { \seq_map_break:n { \sort_return_swapped: } }
1458       }
1459     }
1460   }

```

```

1461     }
1462 }

```

(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}

1463 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1464 {
1465   \int_compare:nNnTF
1466     { \__zrefclever_extract_default:nnn {#1} { abspage } { -1 } }
1467     >
1468     { \__zrefclever_extract_default:nnn {#2} { abspage } { -1 } }
1469     { \sort_return_swapped: }
1470     { \sort_return_same: }
1471 }

```

(End definition for `_zrefclever_sort_page:nn`.)

8 Typesetting

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

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

arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

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

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

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

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

<code>\l__zrefclever_typeset_labels_seq</code>	
<code>\l__zrefclever_typeset_last_bool</code>	1472 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l__zrefclever_last_of_type_bool</code>	1473 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	1474 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>

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

`\l_zrefclever_type_count_int`
`\l__zrefclever_label_count_int`

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

1475 `\int_new:N \l__zrefclever_type_count_int`
1476 `\int_new:N \l__zrefclever_label_count_int`

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

`\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`

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

1477 `\tl_new:N \l__zrefclever_label_a_tl`
1478 `\tl_new:N \l__zrefclever_label_b_tl`
1479 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
1480 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
1481 `\tl_new:N \l__zrefclever_type_first_label_tl`
1482 `\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.

1483 `\tl_new:N \l__zrefclever_type_name_tl`
1484 `\bool_new:N \l__zrefclever_name_in_link_bool`
1485 `\tl_new:N \l__zrefclever_name_format_tl`
1486 `\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.

1487 `\int_new:N \l__zrefclever_range_count_int`
1488 `\int_new:N \l__zrefclever_range_same_count_int`
1489 `\tl_new:N \l__zrefclever_range_beg_label_tl`
1490 `\bool_new:N \l__zrefclever_next_maybe_range_bool`
1491 `\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.

1492 `\tl_new:N \l__zrefclever_tpairsep_tl`
1493 `\tl_new:N \l__zrefclever_tlistsep_tl`
1494 `\tl_new:N \l__zrefclever_tlastsep_tl`
1495 `\tl_new:N \l__zrefclever_namesep_tl`
1496 `\tl_new:N \l__zrefclever_pairsep_tl`
1497 `\tl_new:N \l__zrefclever_listsep_tl`
1498 `\tl_new:N \l__zrefclever_lastsep_tl`
1499 `\tl_new:N \l__zrefclever_rangesep_tl`
1500 `\tl_new:N \l__zrefclever_refpre_out_tl`
1501 `\tl_new:N \l__zrefclever_refpos_out_tl`
1502 `\tl_new:N \l__zrefclever_refpre_in_tl`
1503 `\tl_new:N \l__zrefclever_refpos_in_tl`
1504 `\tl_new:N \l__zrefclever_namefont_tl`
1505 `\tl_new:N \l__zrefclever_reffont_out_tl`
1506 `\tl_new:N \l__zrefclever_reffont_in_tl`

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

Main functions

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

```

1507 \cs_new_protected:Npn \_zrefclever_typeset_refs:
1508 {
1509   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1510   \l__zrefclever_zcref_labels_seq
1511   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1512   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1513   \tl_clear:N \l__zrefclever_type_first_label_tl
1514   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1515   \tl_clear:N \l__zrefclever_range_beg_label_tl
1516   \int_zero:N \l__zrefclever_label_count_int
1517   \int_zero:N \l__zrefclever_type_count_int
1518   \int_zero:N \l__zrefclever_range_count_int
1519   \int_zero:N \l__zrefclever_range_same_count_int
1520
1521   % Get type block options (not type-specific).
1522   \_zrefclever_get_ref_string:nN { tpairsep }
1523   \l__zrefclever_tpairsep_tl
1524   \_zrefclever_get_ref_string:nN { tlistsep }
1525   \l__zrefclever_tlistsep_tl
1526   \_zrefclever_get_ref_string:nN { tlastsep }
1527   \l__zrefclever_tlastsep_tl
1528
1529   % Process label stack.
1530   \bool_set_false:N \l__zrefclever_typeset_last_bool
1531   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1532   {
1533     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1534     \l__zrefclever_label_a_tl
1535     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1536     {
1537       \tl_clear:N \l__zrefclever_label_b_tl
1538       \bool_set_true:N \l__zrefclever_typeset_last_bool
1539     }
1540     {
1541       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1542       \l__zrefclever_label_b_tl
1543     }
1544
1545     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1546     {
1547       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1548       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1549     }
1550     {
1551       \_zrefclever_def_extract_default:Nvnn
1552       \l__zrefclever_label_type_a_tl \l__zrefclever_label_a_tl
1553       { zc@type } { \c_empty_tl }
1554       \_zrefclever_def_extract_default:Nvnn
1555       \l__zrefclever_label_type_b_tl \l__zrefclever_label_b_tl
1556       { zc@type } { \c_empty_tl }
1557     }

```

```

1558
1559 % First, we establish whether the "current label" (i.e. 'a') is the
1560 % last one of its type. This can happen because the "next label"
1561 % (i.e. 'b') is of a different type (or different definition status),
1562 % or because we are at the end of the list.
1563 \bool_if:NTF \l__zrefclever_typeset_last_bool
1564 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1565 {
1566   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1567   {
1568     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1569     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1570     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1571   }
1572   {
1573     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1574     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1575     {
1576       % Neither is undefined, we must check the types.
1577       \bool_if:NTF
1578       {
1579         % Both empty: same "type".
1580         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1581         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1582       }
1583       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1584       {
1585         \bool_if:nTF
1586         {
1587           % Neither empty: compare types.
1588           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1589           &&
1590           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1591         }
1592         {
1593           \tl_if_eq:NNTF
1594           \l__zrefclever_label_type_a_tl
1595           \l__zrefclever_label_type_b_tl
1596           {
1597             \bool_set_false:N
1598             \l__zrefclever_last_of_type_bool
1599           }
1600           {
1601             \bool_set_true:N
1602             \l__zrefclever_last_of_type_bool
1603           }
1604         }
1605         % One empty, the other not: different "types".
1606         {
1607           \bool_set_true:N
1608           \l__zrefclever_last_of_type_bool
1609         }
1610       }
1611     }
1612   }

```



```

1612     }
1613   }
1614
1615   % Handle warnings in case of reference or type undefined.
1616   \zref@refused { \l__zrefclever_label_a_tl }
1617   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1618   {}
1619   {
1620     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1621     {
1622       \msg_warning:nxx { zref-clever } { missing-type }
1623       { \l__zrefclever_label_a_tl }
1624     }
1625   }
1626
1627   % Get type-specific separators, refpre/pos and font options, once per
1628   % type.
1629   \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1630   {
1631     \__zrefclever_get_ref_string:nN { namesep      }
1632     \l__zrefclever_namesep_tl
1633     \__zrefclever_get_ref_string:nN { rangesep     }
1634     \l__zrefclever_rangesep_tl
1635     \__zrefclever_get_ref_string:nN { pairsep      }
1636     \l__zrefclever_pairsep_tl
1637     \__zrefclever_get_ref_string:nN { listsep      }
1638     \l__zrefclever_listsep_tl
1639     \__zrefclever_get_ref_string:nN { lastsep      }
1640     \l__zrefclever_lastsep_tl
1641     \__zrefclever_get_ref_string:nN { refpre       }
1642     \l__zrefclever_refpre_out_tl
1643     \__zrefclever_get_ref_string:nN { refpos       }
1644     \l__zrefclever_refpos_out_tl
1645     \__zrefclever_get_ref_string:nN { refpre-in    }
1646     \l__zrefclever_refpre_in_tl
1647     \__zrefclever_get_ref_string:nN { refpos-in    }
1648     \l__zrefclever_refpos_in_tl
1649     \__zrefclever_get_ref_font:nN   { namefont     }
1650     \l__zrefclever_namefont_tl
1651     \__zrefclever_get_ref_font:nN   { reffont      }
1652     \l__zrefclever_reffont_out_tl
1653     \__zrefclever_get_ref_font:nN   { reffont-in   }
1654     \l__zrefclever_reffont_in_tl
1655   }
1656
1657   % Here we send this to a couple of auxiliary functions.
1658   \bool_if:NTF \l__zrefclever_last_of_type_bool
1659   { There exists no next label of the same type as the current.
1660     { \__zrefclever_typeset_refs_last_of_type: }
1661     % There exists a next label of the same type as the current.
1662     { \__zrefclever_typeset_refs_not_last_of_type: }
1663   }
1664 }

```

(End definition for __zrefclever_typeset_refs:.)

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

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

```

1665 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1666 {
1667   % Process the current label to the current queue.
1668   \int_case:nnF { \l__zrefclever_label_count_int }
1669   {
1670     % It is the last label of its type, but also the first one, and that's
1671     % what matters here: just store it.
1672     { 0 }
1673     {
1674       \tl_set:NV \l__zrefclever_type_first_label_tl
1675       \l__zrefclever_label_a_tl
1676       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1677       \l__zrefclever_label_type_a_tl
1678     }
1679
1680     % The last is the second: we have a pair (if not repeated).
1681     { 1 }
1682     {
1683       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1684       {
1685         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1686         {
1687           \exp_not:V \l__zrefclever_pairsep_tl
1688           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1689         }
1690       }
1691     }
1692   }
1693   % Last is third or more of its type: without repetition, we'd have the
1694   % last element on a list, but control for possible repetition.
1695   {
1696     \int_case:nnF { \l__zrefclever_range_count_int }
1697     {
1698       % There was no range going on.
1699       { 0 }
1700       {
1701         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1702         {
1703           \exp_not:V \l__zrefclever_lastsep_tl
1704           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1705         }
1706       }

```

```

1707 % Last in the range is also the second in it.
1708 { 1 }
1709 {
1710   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1711   {
1712     % We know 'range_beg_label' is not empty, since this is the
1713     % second element in the range, but the third or more in the
1714     % type list.
1715     \exp_not:V \l__zrefclever_listsep_tl
1716     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1717     \int_compare:nNnF
1718       { \l__zrefclever_range_same_count_int } = { 1 }
1719       {
1720         \exp_not:V \l__zrefclever_lastsep_tl
1721         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1722       }
1723   }
1724 }
1725 }
1726 % Last in the range is third or more in it.
1727 {
1728   \int_case:nnF
1729   {
1730     \l__zrefclever_range_count_int -
1731     \l__zrefclever_range_same_count_int
1732   }
1733   {
1734     % Repetition, not a range.
1735     { 0 }
1736     {
1737       % If 'range_beg_label' is empty, it means it was also the
1738       % first of the type, and hence was already handled.
1739       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1740       {
1741         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1742         {
1743           \exp_not:V \l__zrefclever_lastsep_tl
1744           \__zrefclever_get_ref:V
1745             \l__zrefclever_range_beg_label_tl
1746         }
1747       }
1748     }
1749     % A 'range', but with no skipped value, treat as list.
1750     { 1 }
1751     {
1752       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1753       {
1754         % Ditto.
1755         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1756         {
1757           \exp_not:V \l__zrefclever_listsep_tl
1758           \__zrefclever_get_ref:V
1759             \l__zrefclever_range_beg_label_tl
1760         }
1761       }
1762     }
1763   }
1764 }

```

```

1761         \exp_not:V \l__zrefclever_lastsep_tl
1762         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1763     }
1764 }
1765 }
1766 {
1767     % An actual range.
1768     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1769     {
1770         % Ditto.
1771         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1772         {
1773             \exp_not:V \l__zrefclever_lastsep_tl
1774             \__zrefclever_get_ref:V
1775             \l__zrefclever_range_beg_label_tl
1776         }
1777         \exp_not:V \l__zrefclever_rangesep_tl
1778         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1779     }
1780 }
1781 }
1782 }
1783
1784 % Handle "range" option. The idea is simple: if the queue is not empty,
1785 % we replace it with the end of the range (or pair). We can still
1786 % retrieve the end of the range from 'label_a' since we know to be
1787 % processing the last label of its type at this point.
1788 \bool_if:NT \l__zrefclever_typeset_range_bool
1789 {
1790     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1791     {
1792         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1793         { }
1794         {
1795             \msg_warning:nxx { zref-clever } { single-element-range }
1796             { \l__zrefclever_type_first_label_type_tl }
1797         }
1798     }
1799     {
1800         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1801         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1802         { }
1803         {
1804             \__zrefclever_labels_in_sequence:nn
1805             { \l__zrefclever_type_first_label_tl }
1806             { \l__zrefclever_label_a_tl }
1807         }
1808         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1809         {
1810             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1811             { \exp_not:V \l__zrefclever_pairsep_tl }
1812             { \exp_not:V \l__zrefclever_rangesep_tl }
1813             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1814         }

```

```

1815     }
1816   }
1817
1818   % Now that the type block is finished, we can add the name and the first
1819   % ref to the queue. Also, if "typeset" option is not "both", handle it
1820   % here as well.
1821   \__zrefclever_type_name_setup:
1822   \bool_if:nTF
1823     { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1824     {
1825       \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1826       { \__zrefclever_get_ref_first: }
1827     }
1828     {
1829       \bool_if:nTF
1830       { \l__zrefclever_typeset_ref_bool }
1831       {
1832         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1833         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1834       }
1835       {
1836         \bool_if:nTF
1837         { \l__zrefclever_typeset_name_bool }
1838         {
1839           \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1840           {
1841             \bool_if:NTF \l__zrefclever_name_in_link_bool
1842             {
1843               \exp_not:N \group_begin:
1844               \exp_not:V \l__zrefclever_namefont_tl
1845               % It's two '@s', but escaped for DocStrip.
1846               \exp_not:N \hyper@@link
1847               {
1848                 \__zrefclever_extract_url_unexp:V
1849                 \l__zrefclever_type_first_label_tl
1850               }
1851               {
1852                 \__zrefclever_extract_default_unexp:Vnn
1853                 \l__zrefclever_type_first_label_tl
1854                 { anchor } { }
1855               }
1856               { \exp_not:V \l__zrefclever_type_name_tl }
1857               \exp_not:N \group_end:
1858             }
1859             {
1860               \exp_not:N \group_begin:
1861               \exp_not:V \l__zrefclever_namefont_tl
1862               \exp_not:V \l__zrefclever_type_name_tl
1863               \exp_not:N \group_end:
1864             }
1865           }
1866         }
1867       }
1868       % Logically, this case would correspond to "typeset=none", but

```

```

1869         % it should not occur, given that the options are set up to
1870         % typeset either "ref" or "name". Still, leave here a
1871         % sensible fallback, equal to the behavior of "both".
1872         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1873         { \__zrefclever_get_ref_first: }
1874     }
1875 }
1876 }
1877
1878 % Typeset the previous type, if there is one.
1879 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1880 {
1881     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1882     { \l__zrefclever_tlistsep_tl }
1883     \l__zrefclever_typeset_queue_prev_tl
1884 }
1885
1886 % Wrap up loop, or prepare for next iteration.
1887 \bool_if:NTF \l__zrefclever_typeset_last_bool
1888 {
1889     % We are finishing, typeset the current queue.
1890     \int_case:nnF { \l__zrefclever_type_count_int }
1891     {
1892         % Single type.
1893         { 0 }
1894         { \l__zrefclever_typeset_queue_curr_tl }
1895         % Pair of types.
1896         { 1 }
1897         {
1898             \l__zrefclever_tpairsep_tl
1899             \l__zrefclever_typeset_queue_curr_tl
1900         }
1901     }
1902     {
1903         % Last in list of types.
1904         \l__zrefclever_tlastsep_tl
1905         \l__zrefclever_typeset_queue_curr_tl
1906     }
1907 }
1908 {
1909     % There are further labels, set variables for next iteration.
1910     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1911     \l__zrefclever_typeset_queue_curr_tl
1912     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1913     \tl_clear:N \l__zrefclever_type_first_label_tl
1914     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1915     \tl_clear:N \l__zrefclever_range_beg_label_tl
1916     \int_zero:N \l__zrefclever_label_count_int
1917     \int_incr:N \l__zrefclever_type_count_int
1918     \int_zero:N \l__zrefclever_range_count_int
1919     \int_zero:N \l__zrefclever_range_same_count_int
1920 }
1921 }

```

(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.

```

1922 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1923 {
1924   % Signal if next label may form a range with the current one (only
1925   % considered if compression is enabled in the first place).
1926   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1927   \bool_set_false:N \l__zrefclever_next_is_same_bool
1928   \bool_if:NT \l__zrefclever_typeset_compress_bool
1929   {
1930     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1931     { }
1932     {
1933       \__zrefclever_labels_in_sequence:nn
1934       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1935     }
1936   }
1937
1938   % Process the current label to the current queue.
1939   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1940   {
1941     % Current label is the first of its type (also not the last, but it
1942     % doesn't matter here): just store the label.
1943     \tl_set:NV \l__zrefclever_type_first_label_tl
1944     \l__zrefclever_label_a_tl
1945     \tl_set:NV \l__zrefclever_type_first_label_type_tl
1946     \l__zrefclever_label_type_a_tl
1947
1948     % If the next label may be part of a range, we set 'range_beg_label'
1949     % to "empty" (we deal with it as the "first", and must do it there, to
1950     % handle hyperlinking), but also step the range counters.
1951     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1952     {
1953       \tl_clear:N \l__zrefclever_range_beg_label_tl
1954       \int_incr:N \l__zrefclever_range_count_int
1955       \bool_if:NT \l__zrefclever_next_is_same_bool
1956       { \int_incr:N \l__zrefclever_range_same_count_int }
1957     }
1958   }
1959   {
1960     % Current label is neither the first (nor the last) of its type.
1961     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1962     {
1963       % Starting, or continuing a range.
1964       \int_compare:nNnTF
1965       { \l__zrefclever_range_count_int } = { 0 }
1966       {
1967         % There was no range going, we are starting one.
1968         \tl_set:NV \l__zrefclever_range_beg_label_tl
1969         \l__zrefclever_label_a_tl
1970         \int_incr:N \l__zrefclever_range_count_int
1971         \bool_if:NT \l__zrefclever_next_is_same_bool
1972         { \int_incr:N \l__zrefclever_range_same_count_int }
1973       }
1974       {

```

```

1975         % Second or more in the range, but not the last.
1976         \int_incr:N \l__zrefclever_range_count_int
1977         \bool_if:NT \l__zrefclever_next_is_same_bool
1978         { \int_incr:N \l__zrefclever_range_same_count_int }
1979     }
1980 }
1981 {
1982     % Next element is not in sequence: there was no range, or we are
1983     % closing one.
1984     \int_case:nnF { \l__zrefclever_range_count_int }
1985     {
1986         % There was no range going on.
1987         { 0 }
1988         {
1989             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1990             {
1991                 \exp_not:V \l__zrefclever_listsep_tl
1992                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1993             }
1994         }
1995         % Last is second in the range: if 'range_same_count' is also
1996         % '1', it's a repetition (drop it), otherwise, it's a "pair
1997         % within a list", treat as list.
1998         { 1 }
1999         {
2000             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2001             {
2002                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2003                 {
2004                     \exp_not:V \l__zrefclever_listsep_tl
2005                     \__zrefclever_get_ref:V
2006                     \l__zrefclever_range_beg_label_tl
2007                 }
2008                 \int_compare:nNnF
2009                 { \l__zrefclever_range_same_count_int } = { 1 }
2010                 {
2011                     \exp_not:V \l__zrefclever_listsep_tl
2012                     \__zrefclever_get_ref:V
2013                     \l__zrefclever_label_a_tl
2014                 }
2015             }
2016         }
2017     }
2018 }
2019 % Last is third or more in the range: if 'range_count' and
2020 % 'range_same_count' are the same, its a repetition (drop it),
2021 % if they differ by '1', its a list, if they differ by more,
2022 % it is a real range.
2023 \int_case:nnF
2024 {
2025     \l__zrefclever_range_count_int -
2026     \l__zrefclever_range_same_count_int
2027 }
2028 {

```



```

2029         { 0 }
2030     {
2031         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2032         {
2033             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2034             {
2035                 \exp_not:V \l__zrefclever_listsep_tl
2036                 \__zrefclever_get_ref:V
2037                 \l__zrefclever_range_beg_label_tl
2038             }
2039         }
2040     }
2041     { 1 }
2042     {
2043         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2044         {
2045             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2046             {
2047                 \exp_not:V \l__zrefclever_listsep_tl
2048                 \__zrefclever_get_ref:V
2049                 \l__zrefclever_range_beg_label_tl
2050             }
2051             \exp_not:V \l__zrefclever_listsep_tl
2052             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2053         }
2054     }
2055 }
2056 {
2057     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2058     {
2059         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2060         {
2061             \exp_not:V \l__zrefclever_listsep_tl
2062             \__zrefclever_get_ref:V
2063             \l__zrefclever_range_beg_label_tl
2064         }
2065         \exp_not:V \l__zrefclever_rangesep_tl
2066         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2067     }
2068 }
2069 }
2070 % Reset counters.
2071 \int_zero:N \l__zrefclever_range_count_int
2072 \int_zero:N \l__zrefclever_range_same_count_int
2073 }
2074 }
2075 % Step label counter for next iteration.
2076 \int_incr:N \l__zrefclever_label_count_int
2077 }

```

(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.

```

2078 \cs_new_protected:Npn \__zrefclever_ref_default:
2079 { \zref@default }
2080 \cs_new_protected:Npn \__zrefclever_name_default:
2081 { \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>}

2082 \cs_new:Npn \__zrefclever_get_ref:n #1
2083 {
2084   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2085   {
2086     \bool_if:nTF
2087     {
2088       \l__zrefclever_use_hyperref_bool &&
2089       ! \l__zrefclever_link_star_bool
2090     }
2091     {
2092       \exp_not:N \group_begin:
2093       \exp_not:N \l__zrefclever_reffont_out_tl
2094       \exp_not:N \l__zrefclever_refpre_out_tl

```

```

2095 \exp_not:N \group_begin:
2096 \exp_not:V \l__zrefclever_reffont_in_tl
2097 % It's two '@s', but escaped for DocStrip.
2098 \exp_not:N \hyper@@link
2099 { \__zrefclever_extract_url_unexp:n {#1} }
2100 { \__zrefclever_extract_default_unexp:nnn {#1} { anchor } { } }
2101 {
2102 \exp_not:V \l__zrefclever_refpre_in_tl
2103 \__zrefclever_extract_default_unexp:nvn {#1}
2104 { l__zrefclever_ref_property_tl } { }
2105 \exp_not:V \l__zrefclever_refpos_in_tl
2106 }
2107 \exp_not:N \group_end:
2108 \exp_not:V \l__zrefclever_refpos_out_tl
2109 \exp_not:N \group_end:
2110 }
2111 {
2112 \exp_not:N \group_begin:
2113 \exp_not:V \l__zrefclever_reffont_out_tl
2114 \exp_not:V \l__zrefclever_refpre_out_tl
2115 \exp_not:N \group_begin:
2116 \exp_not:V \l__zrefclever_reffont_in_tl
2117 \exp_not:V \l__zrefclever_refpre_in_tl
2118 \__zrefclever_extract_default_unexp:nvn {#1}
2119 { l__zrefclever_ref_property_tl } { }
2120 \exp_not:V \l__zrefclever_refpos_in_tl
2121 \exp_not:N \group_end:
2122 \exp_not:V \l__zrefclever_refpos_out_tl
2123 \exp_not:N \group_end:
2124 }
2125 }
2126 { \__zrefclever_ref_default: }
2127 }
2128 \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.

```

2129 \cs_new:Npn \__zrefclever_get_ref_first:
2130 {
2131 \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2132 { \__zrefclever_ref_default: }
2133 {
2134 \bool_if:NTF \l__zrefclever_name_in_link_bool
2135 {
2136 \zref@ifrefcontainsprop
2137 { \l__zrefclever_type_first_label_tl }

```

```

2138 { \l__zrefclever_ref_property_tl }
2139 {
2140   % It's two '@s', but escaped for DocStrip.
2141   \exp_not:N \hyper@@link
2142   {
2143     \__zrefclever_extract_url_unexp:V
2144     \l__zrefclever_type_first_label_tl
2145   }
2146   {
2147     \__zrefclever_extract_default_unexp:Vnn
2148     \l__zrefclever_type_first_label_tl
2149     { anchor } { }
2150   }
2151   {
2152     \exp_not:N \group_begin:
2153     \exp_not:V \l__zrefclever_namefont_tl
2154     \exp_not:V \l__zrefclever_type_name_tl
2155     \exp_not:N \group_end:
2156     \exp_not:V \l__zrefclever_namesep_tl
2157     \exp_not:N \group_begin:
2158     \exp_not:V \l__zrefclever_reffont_out_tl
2159     \exp_not:V \l__zrefclever_refpre_out_tl
2160     \exp_not:N \group_begin:
2161     \exp_not:V \l__zrefclever_reffont_in_tl
2162     \exp_not:V \l__zrefclever_refpre_in_tl
2163     \__zrefclever_extract_default_unexp:Vnn
2164     \l__zrefclever_type_first_label_tl
2165     { \l__zrefclever_ref_property_tl } { }
2166     \exp_not:V \l__zrefclever_refpos_in_tl
2167     \exp_not:N \group_end:
2168     % hyperlink makes it's own group, we'd like to close the
2169     % 'refpre-out' group after 'refpos-out', but... we close
2170     % it here, and give the trailing 'refpos-out' its own
2171     % group. This will result that formatting given to
2172     % 'refpre-out' will not reach 'refpos-out', but I see no
2173     % alternative, and this has to be handled specially.
2174     \exp_not:N \group_end:
2175   }
2176   \exp_not:N \group_begin:
2177   % Ditto: special treatment.
2178   \exp_not:V \l__zrefclever_reffont_out_tl
2179   \exp_not:V \l__zrefclever_refpos_out_tl
2180   \exp_not:N \group_end:
2181 }
2182 {
2183   \exp_not:N \group_begin:
2184   \exp_not:V \l__zrefclever_namefont_tl
2185   \exp_not:V \l__zrefclever_type_name_tl
2186   \exp_not:N \group_end:
2187   \exp_not:V \l__zrefclever_namesep_tl
2188   \__zrefclever_ref_default:
2189 }
2190 }
2191 {

```

```

2192 \tl_if_empty:NTF \l__zrefclever_type_name_tl
2193 {
2194   \__zrefclever_name_default:
2195   \exp_not:V \l__zrefclever_namesep_tl
2196 }
2197 {
2198   \exp_not:N \group_begin:
2199   \exp_not:V \l__zrefclever_namefont_tl
2200   \exp_not:V \l__zrefclever_type_name_tl
2201   \exp_not:N \group_end:
2202   \exp_not:V \l__zrefclever_namesep_tl
2203 }
2204 \zref@ifrefcontainsprop
2205 { \l__zrefclever_type_first_label_tl }
2206 { \l__zrefclever_ref_property_tl }
2207 {
2208   \bool_if:nTF
2209   {
2210     \l__zrefclever_use_hyperref_bool &&
2211     ! \l__zrefclever_link_star_bool
2212   }
2213   {
2214     \exp_not:N \group_begin:
2215     \exp_not:V \l__zrefclever_reffont_out_tl
2216     \exp_not:V \l__zrefclever_refpre_out_tl
2217     \exp_not:N \group_begin:
2218     \exp_not:V \l__zrefclever_reffont_in_tl
2219     % It's two 's', but escaped for DocStrip.
2220     \exp_not:N \hyper@@link
2221     {
2222       \__zrefclever_extract_url_unexp:V
2223       \l__zrefclever_type_first_label_tl
2224     }
2225     {
2226       \__zrefclever_extract_default_unexp:Vnn
2227       \l__zrefclever_type_first_label_tl
2228       { anchor } { }
2229     }
2230     {
2231       \exp_not:V \l__zrefclever_refpre_in_tl
2232       \__zrefclever_extract_default_unexp:Vnn
2233       \l__zrefclever_type_first_label_tl
2234       { \l__zrefclever_ref_property_tl } { }
2235       \exp_not:V \l__zrefclever_refpos_in_tl
2236     }
2237     \exp_not:N \group_end:
2238     \exp_not:V \l__zrefclever_refpos_out_tl
2239     \exp_not:N \group_end:
2240   }
2241   {
2242     \exp_not:N \group_begin:
2243     \exp_not:V \l__zrefclever_reffont_out_tl
2244     \exp_not:V \l__zrefclever_refpre_out_tl
2245     \exp_not:N \group_begin:

```

```

2246         \exp_not:V \l__zrefclever_reffont_in_tl
2247         \exp_not:V \l__zrefclever_refpre_in_tl
2248         \__zrefclever_extract_default_unexp:Vvn
2249         \l__zrefclever_type_first_label_tl
2250         { \l__zrefclever_ref_property_tl } { }
2251         \exp_not:V \l__zrefclever_refpos_in_tl
2252         \exp_not:N \group_end:
2253         \exp_not:V \l__zrefclever_refpos_out_tl
2254         \exp_not:N \group_end:
2255     }
2256 }
2257 { \__zrefclever_ref_default: }
2258 }
2259 }
2260 }

```

(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.

```

2261 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2262 {
2263   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2264   { \tl_clear:N \l__zrefclever_type_name_tl }
2265   {
2266     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2267     { \tl_clear:N \l__zrefclever_type_name_tl }
2268     {
2269       % Determine whether we should use capitalization, abbreviation,
2270       % and plural.
2271       \bool_lazy_or:nnTF
2272       { \l__zrefclever_capitalize_bool }
2273       {
2274         \l__zrefclever_capitalize_first_bool &&
2275         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2276       }
2277       { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2278       { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2279       % If the queue is empty, we have a singular, otherwise, plural.
2280       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2281       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2282       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2283       \bool_lazy_and:nnTF
2284       { \l__zrefclever_abbrev_bool }

```

```

2285 {
2286   ! \int_compare_p:nNn
2287     { \l__zrefclever_type_count_int } = { 0 } ||
2288   ! \l__zrefclever_noabbrev_first_bool
2289 }
2290 {
2291   \tl_set:NV \l__zrefclever_name_format_fallback_tl
2292     \l__zrefclever_name_format_tl
2293   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2294 }
2295 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2296
2297 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2298 {
2299   \prop_get:cVNF
2300   {
2301     \l__zrefclever_type_
2302     \l__zrefclever_type_first_label_type_tl _options_prop
2303   }
2304   \l__zrefclever_name_format_tl
2305   \l__zrefclever_type_name_tl
2306   {
2307     \__zrefclever_get_type_transl:xxxNF
2308     { \l__zrefclever_ref_language_tl }
2309     { \l__zrefclever_type_first_label_type_tl }
2310     { \l__zrefclever_name_format_tl }
2311     \l__zrefclever_type_name_tl
2312     {
2313       \tl_clear:N \l__zrefclever_type_name_tl
2314       \msg_warning:nxx { zref-clever } { missing-name }
2315       { \l__zrefclever_type_first_label_type_tl }
2316     }
2317   }
2318 }
2319 {
2320   \prop_get:cVNF
2321   {
2322     \l__zrefclever_type_
2323     \l__zrefclever_type_first_label_type_tl _options_prop
2324   }
2325   \l__zrefclever_name_format_tl
2326   \l__zrefclever_type_name_tl
2327   {
2328     \prop_get:cVNF
2329     {
2330       \l__zrefclever_type_
2331       \l__zrefclever_type_first_label_type_tl _options_prop
2332     }
2333     \l__zrefclever_name_format_fallback_tl
2334     \l__zrefclever_type_name_tl
2335     {
2336       \__zrefclever_get_type_transl:xxxNF
2337       { \l__zrefclever_ref_language_tl }
2338       { \l__zrefclever_type_first_label_type_tl }

```

```

2339         { \l__zrefclever_name_format_tl }
2340         \l__zrefclever_type_name_tl
2341         {
2342             \__zrefclever_get_type_transl:xxxNF
2343             { \l__zrefclever_ref_language_tl }
2344             { \l__zrefclever_type_first_label_type_tl }
2345             { \l__zrefclever_name_format_fallback_tl }
2346             \l__zrefclever_type_name_tl
2347             {
2348                 \tl_clear:N \l__zrefclever_type_name_tl
2349                 \msg_warning:nxx { zref-clever }
2350                 { missing-name }
2351                 { \l__zrefclever_type_first_label_type_tl }
2352             }
2353         }
2354     }
2355 }
2356 }
2357 }
2358 }
2359
2360 % Signal whether the type name is to be included in the hyperlink or not.
2361 \bool_lazy_any:nTF
2362 {
2363     { ! \l__zrefclever_use_hyperref_bool }
2364     { \l__zrefclever_link_star_bool }
2365     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2366     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2367 }
2368 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2369 {
2370     \bool_lazy_any:nTF
2371     {
2372         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2373         {
2374             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2375             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2376         }
2377         {
2378             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2379             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2380             \l__zrefclever_typeset_last_bool &&
2381             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2382         }
2383     }
2384     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2385     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2386 }
2387 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_extract_url_unexp:n A convenience auxiliary function for extraction of the url / urluse property, provided by the zref-xr module. Ensure that, in the context of an x expansion, \zref@extractdefault

is expanded exactly twice, but no further to retrieve the proper value. See documentation for `__zrefclever_extract_default_unexp:nnn`.

```

2388 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2389 {
2390   \zref@ifpropundefined { urluse }
2391   {
2392     \__zrefclever_extract_default_unexp:nnn
2393     {#1} { url } { \c_empty_tl }
2394   }
2395   {
2396     \zref@ifrefcontainsprop {#1} { urluse }
2397     {
2398       \__zrefclever_extract_default_unexp:nnn
2399       {#1} { urluse } { \c_empty_tl }
2400     }
2401     {
2402       \__zrefclever_extract_default_unexp:nnn
2403       {#1} { url } { \c_empty_tl }
2404     }
2405   }
2406 }
2407 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for `__zrefclever_extract_url_unexp: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 {\langle label\ a \rangle} {\langle label\ b \rangle}

2408 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2409 {
2410   \__zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
2411   {#1} { externaldocument } { \c_empty_tl }
2412   \__zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
2413   {#2} { externaldocument } { \c_empty_tl }
2414
2415   \tl_if_eq:NNT
2416   \l__zrefclever_label_extdoc_a_tl
2417   \l__zrefclever_label_extdoc_b_tl
2418   {
2419     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2420     {
2421       \exp_args:Nxx \tl_if_eq:nnT
2422       {
2423         \__zrefclever_extract_default_unexp:nnn
2424         {#1} { zc@pgfmt } { }
2425       }
2426     }

```

```

2427     \__zrefclever_extract_default_unexp:nnn
2428     {#2} { zc@pgfmt } { }
2429   }
2430   {
2431     \int_compare:nNnTF
2432     {
2433       \__zrefclever_extract_default:nnn
2434       {#1} { zc@pgval } { -2 } + 1
2435     }
2436     =
2437     {
2438       \__zrefclever_extract_default:nnn
2439       {#2} { zc@pgval } { -1 }
2440     }
2441     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2442     {
2443       \int_compare:nNnT
2444       {
2445         \__zrefclever_extract_default:nnn
2446         {#1} { zc@pgval } { -1 }
2447       }
2448       =
2449       {
2450         \__zrefclever_extract_default:nnn
2451         {#2} { zc@pgval } { -1 }
2452       }
2453       {
2454         \bool_set_true:N
2455         \l__zrefclever_next_maybe_range_bool
2456         \bool_set_true:N
2457         \l__zrefclever_next_is_same_bool
2458       }
2459     }
2460   }
2461 }
2462 {
2463   \exp_args:Nxx \tl_if_eq:nnT
2464   {
2465     \__zrefclever_extract_default_unexp:nnn
2466     {#1} { zc@counter } { }
2467   }
2468   {
2469     \__zrefclever_extract_default_unexp:nnn
2470     {#2} { zc@counter } { }
2471   }
2472   {
2473     \exp_args:Nxx \tl_if_eq:nnT
2474     {
2475       \__zrefclever_extract_default_unexp:nnn
2476       {#1} { zc@enclval } { }
2477     }
2478     {
2479       \__zrefclever_extract_default_unexp:nnn
2480       {#2} { zc@enclval } { }

```

```

2481     }
2482     {
2483         \int_compare:nNnTF
2484         {
2485             \__zrefclever_extract_default:nnn
2486             {#1} { zc@cntval } { -2 } + 1
2487         }
2488         =
2489         {
2490             \__zrefclever_extract_default:nnn
2491             {#2} { zc@cntval } { -1 }
2492         }
2493         {
2494             \bool_set_true:N
2495             \l__zrefclever_next_maybe_range_bool
2496         }
2497         {
2498             \int_compare:nNnT
2499             {
2500                 \__zrefclever_extract_default:nnn
2501                 {#1} { zc@cntval } { -1 }
2502             }
2503             =
2504             {
2505                 \__zrefclever_extract_default:nnn
2506                 {#2} { zc@cntval } { -1 }
2507             }
2508             {
2509                 \bool_set_true:N
2510                 \l__zrefclever_next_maybe_range_bool
2511                 \exp_args:Nxx \tl_if_eq:nnT
2512                 {
2513                     \__zrefclever_extract_default_unexp:nvn {#1}
2514                     { l__zrefclever_ref_property_tl } { }
2515                 }
2516                 {
2517                     \__zrefclever_extract_default_unexp:nvn {#2}
2518                     { l__zrefclever_ref_property_tl } { }
2519                 }
2520                 {
2521                     \bool_set_true:N
2522                     \l__zrefclever_next_is_same_bool
2523                 }
2524             }
2525         }
2526     }
2527 }
2528 }
2529 }
2530 }

```

(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 $\langle tl\ variable \rangle$. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of $\backslash l_zrefclever_label_type_a_tl$, 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, $\backslash l_zrefclever_label_type_a_tl$ is indeed what we want in all practical cases. The difference between $\backslash_zrefclever_get_ref_string:nN$ and $\backslash_zrefclever_get_ref_font:nN$ is the kind of option each should be used for. $\backslash_zrefclever_get_ref_string:nN$ is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fallback”). $\backslash_zrefclever_get_ref_font:nN$ 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>}
2531 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2532 {
2533   % First attempt: general options.
2534   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2535   {
2536     % If not found, try type specific options.
2537     \bool_lazy_all:nTF
2538     {
2539       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2540       {
2541         \prop_if_exist_p:c
2542         {
2543           l__zrefclever_type_
2544           \l__zrefclever_label_type_a_tl _options_prop
2545         }
2546       }
2547       {
2548         \prop_if_in_p:cn
2549         {
2550           l__zrefclever_type_
2551           \l__zrefclever_label_type_a_tl _options_prop
2552         }
2553         {#1}
2554       }
2555     }
2556     {
2557       \prop_get:cnN
2558       {
2559         l__zrefclever_type_
2560         \l__zrefclever_label_type_a_tl _options_prop
2561       }
2562       {#1} #2
2563     }
2564     {
2565       % If not found, try type specific translations.
2566       \__zrefclever_get_type_transl:xxnNF
2567       { \l__zrefclever_ref_language_tl }
2568       { \l__zrefclever_label_type_a_tl }
2569       {#1} #2
2570       {
2571         % If not found, try default translations.

```

```

2572         \__zrefclever_get_default_transl:xnNF
2573         { \l__zrefclever_ref_language_tl }
2574         {#1} #2
2575         {
2576             % If not found, try fallback.
2577             \__zrefclever_get_fallback_transl:nNF {#1} #2
2578             {
2579                 \tl_clear:N #2
2580                 \msg_warning:nnn { zref-clever }
2581                 { missing-string } {#1}
2582             }
2583         }
2584     }
2585 }
2586 }
2587 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

\__zrefclever_get_ref_font:nN \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2588 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2589 {
2590     % First attempt: general options.
2591     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2592     {
2593         % If not found, try type specific options.
2594         \bool_lazy_and:nnTF
2595         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2596         {
2597             \prop_if_exist_p:c
2598             {
2599                 l__zrefclever_type_
2600                 \l__zrefclever_label_type_a_tl _options_prop
2601             }
2602         }
2603         {
2604             \prop_get:cnNF
2605             {
2606                 l__zrefclever_type_
2607                 \l__zrefclever_label_type_a_tl _options_prop
2608             }
2609             {#1} #2
2610             { \tl_clear:N #2 }
2611         }
2612         { \tl_clear:N #2 }
2613     }
2614 }

```

(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.

Auxiliary

`_zrefclever_ride_on_label:n` An auxiliary function to “get a ride” on the standard `\label`, so that it issues a `\zlabel` too, to be used locally in selected environments for compatibility support of packages/features for which there’s really no other way to do it.

```
2615 \AddToHook { begindocument }
2616 {
2617   \cs_set_eq:NN \_zrefclever_orig_label:n \label
2618 }
2619 \cs_new_nopar:Npn \_zrefclever_ride_on_label:n #1
2620 {
2621   \_zrefclever_orig_label:n {#1}
2622   \zlabel {#1}
2623 }
```

(End definition for `_zrefclever_ride_on_label:n`.)

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.

```
2624 \tl_new:N \l__zrefclever_footnote_type_tl
2625 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
2626 \AddToHook { env / minipage / begin }
2627 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
2628 \AddToHook { cmd / @makefntext / before }
2629 {
2630   \_zrefclever_zcsetup:x
2631   { currentcounter = \l__zrefclever_footnote_type_tl }
2632 }
```

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.

```

2633 \AddToHook { cmd / appendix / before }
2634 {
2635   \__zrefclever_zcsetup:n
2636   {
2637     countertype =
2638     {
2639       chapter      = appendix ,
2640       section      = appendix ,
2641       subsection   = appendix ,
2642       subsubsection = appendix ,
2643     }
2644   }
2645 }

```

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.

```

2646 \AddToHook { begindocument }
2647 {
2648   \ifpackageloaded { appendix }
2649   {
2650     \newcounter { zc@appendix }
2651     \newcounter { zc@save@appendix }
2652     \setcounter { zc@appendix } { 0 }
2653     \setcounter { zc@save@appendix } { 0 }
2654     \cs_if_exist:cTF { chapter }
2655     {

```

```

2656         \cs_if_exist:cT { section }
2657         {
2658             \__zrefclever_zcsetup:n
2659             { counterresetby = { section = zc@appendix } }
2660         }
2661     }
2662     {
2663         \__zrefclever_zcsetup:n
2664         { counterresetby = { chapter = zc@appendix } }
2665     }
2666     \AddToHook { env / appendices / begin }
2667     {
2668         \stepcounter { zc@save@appendix }
2669         \setcounter { zc@appendix } { \value { zc@save@appendix } }
2670         \__zrefclever_zcsetup:n
2671         {
2672             countertype =
2673             {
2674                 chapter      = appendix ,
2675                 section      = appendix ,
2676                 subsection   = appendix ,
2677                 subsubsection = appendix ,
2678             }
2679         }
2680     }
2681     \AddToHook { env / appendices / end }
2682     { \setcounter { zc@appendix } { 0 } }
2683     \AddToHook { cmd / appendix / before }
2684     {
2685         \stepcounter { zc@save@appendix }
2686         \setcounter { zc@appendix } { \value { zc@save@appendix } }
2687     }
2688     \AddToHook { env / subappendices / begin }
2689     {
2690         \__zrefclever_zcsetup:n
2691         {
2692             countertype =
2693             {
2694                 section      = appendix ,
2695                 subsection   = appendix ,
2696                 subsubsection = appendix ,
2697             } ,
2698         }
2699     }
2700     \msg_info:nnn { zref-clever } { compat-package } { appendix }
2701 }
2702 {}
2703 }

```

9.4 amsmath package

```

2704 \AddToHook { begindocument }
2705 {
2706     \@ifpackageloaded { amsmath }

```



```

2707 {
2708   \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
2709   {
2710     \__zrefclever_orig_ltxlabel:n {#1}
2711     \zlabel {#1}
2712   }
2713   % We must handle 'hyperref' here, which comes very late in the
2714   % preamble, and which loads 'nameref' with a 'atbegindocument' hook,
2715   % which in turn, lets '\ltx@label' be '\label'. This has to come
2716   % after 'nameref'. 'cleveref' also redefines it, and comes even
2717   % later, but this is not compatible with it.
2718   \IfFormatAtLeastTF { 2021-11-15 }
2719   {
2720     \@ifpackageloaded { hyperref }
2721     {
2722       \AddToHook { package / nameref / after }
2723       {
2724         \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2725         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2726       }
2727     }
2728     {
2729       \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2730       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2731     }
2732   }
2733   {
2734     \@ifpackageloaded { hyperref }
2735     {
2736       \@ifpackageloaded { nameref }
2737       {
2738         \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2739         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2740       }
2741       {
2742         \AddToHook { package / after / nameref }
2743         {
2744           \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2745           \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2746         }
2747       }
2748     }
2749     {
2750       \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2751       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2752     }
2753   }
2754
2755   \clist_map_inline:nn
2756   {
2757     equation ,
2758     equation* ,
2759     align ,
2760     align* ,

```

```

2761         alignnat ,
2762         alignat* ,
2763         flalign ,
2764         flalign* ,
2765         xalignnat ,
2766         xalignat* ,
2767         xxalignnat ,
2768         gather ,
2769         gather* ,
2770         multiline ,
2771         multiline* ,
2772     }
2773     {
2774         \AddToHook { env / #1 / begin }
2775         {
2776             % Needed for '\tag', but also for subequations, since we have
2777             % to manually set currentcounter to 'parentequation' in them,
2778             % we also have to manually set it to 'equation' in the
2779             % environments within it.
2780             \__zrefclever_zcsetup:n { currentcounter = equation }
2781         }
2782     }
2783     \AddToHook { env / subequations / begin }
2784     {
2785         \__zrefclever_zcsetup:x
2786         {
2787             counterresetby =
2788             {
2789                 parentequation =
2790                 \__zrefclever_counter_reset_by:n { equation } ,
2791                 equation = parentequation ,
2792             } ,
2793             currentcounter = parentequation ,
2794             countertype = { parentequation = equation } ,
2795         }
2796     }
2797     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
2798 }
2799 {}
2800 }

```

9.5 listings package

```

2801 \AddToHook { begindocument }
2802 {
2803     \@ifpackageloaded { listings }
2804     {
2805         \__zrefclever_zcsetup:n
2806         {
2807             countertype =
2808             {
2809                 lstlisting = listing ,
2810                 lstnumber = line ,
2811             } ,
2812             counterresetby = { lstnumber = lstlisting } ,

```

```

2813     }
2814     \lst@AddToHook { Init }
2815     {

```

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

```

2816         \tl_if_empty:NF \lst@label
2817         { \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.

```

2818         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
2819     }
2820     \msg_info:nnn { zref-clever } { compat-package } { listings }
2821 }
2822 {}
2823 }

```

9.6 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.

```

2824 \AddToHook { begindocument }
2825 {
2826     \@ifpackageloaded { enumitem }
2827     {
2828         \int_set:Nn \l_tmpa_int { 5 }
2829         \bool_while_do:nn
2830         {
2831             \cs_if_exist_p:c
2832             { c@ enum \int_to_roman:n { \l_tmpa_int } }
2833         }
2834         {
2835             \__zrefclever_zcsetup:x
2836             {
2837                 counterresetby =
2838                 {

```

```

2839         enum \int_to_roman:n { \l_tmpa_int } =
2840         enum \int_to_roman:n { \l_tmpa_int - 1 }
2841     } ,
2842     countertype =
2843     { enum \int_to_roman:n { \l_tmpa_int } = item } ,
2844 }
2845 \int_incr:N \l_tmpa_int
2846 }
2847 \int_compare:nNnT { \l_tmpa_int } > { 5 }
2848 { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
2849 }
2850 {}
2851 }
2852 </package>

```

10 Dictionaries

10.1 English

```

2853 <package>\zcDeclareLanguage { english }
2854 <package>\zcDeclareLanguageAlias { american } { english }
2855 <package>\zcDeclareLanguageAlias { australian } { english }
2856 <package>\zcDeclareLanguageAlias { british } { english }
2857 <package>\zcDeclareLanguageAlias { canadian } { english }
2858 <package>\zcDeclareLanguageAlias { newzealand } { english }
2859 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2860 <package>\zcDeclareLanguageAlias { USenglish } { english }
2861 <*dict-english>
2862 namesep = {\nobreakspace} ,
2863 pairsep = {\~and\nobreakspace} ,
2864 listsep = {,\~} ,
2865 lastsep = {\~and\nobreakspace} ,
2866 tpairsep = {\~and\nobreakspace} ,
2867 tlistsep = {,\~} ,
2868 tlastsep = {,\~and\nobreakspace} ,
2869 notesep = {\~} ,
2870 rangesep = {\~to\nobreakspace} ,
2871
2872 type = part ,
2873   Name-sg = Part ,
2874   name-sg = part ,
2875   Name-pl = Parts ,
2876   name-pl = parts ,
2877
2878 type = chapter ,
2879   Name-sg = Chapter ,
2880   name-sg = chapter ,
2881   Name-pl = Chapters ,
2882   name-pl = chapters ,
2883
2884 type = section ,
2885   Name-sg = Section ,

```

```

2886     name-sg = section ,
2887     Name-pl = Sections ,
2888     name-pl = sections ,
2889
2890 type = paragraph ,
2891     Name-sg = Paragraph ,
2892     name-sg = paragraph ,
2893     Name-pl = Paragraphs ,
2894     name-pl = paragraphs ,
2895     Name-sg-ab = Par. ,
2896     name-sg-ab = par. ,
2897     Name-pl-ab = Par. ,
2898     name-pl-ab = par. ,
2899
2900 type = appendix ,
2901     Name-sg = Appendix ,
2902     name-sg = appendix ,
2903     Name-pl = Appendices ,
2904     name-pl = appendices ,
2905
2906 type = subappendix ,
2907     Name-sg = Appendix ,
2908     name-sg = appendix ,
2909     Name-pl = Appendices ,
2910     name-pl = appendices ,
2911
2912 type = page ,
2913     Name-sg = Page ,
2914     name-sg = page ,
2915     Name-pl = Pages ,
2916     name-pl = pages ,
2917     name-sg-ab = p. ,
2918     name-pl-ab = pp. ,
2919
2920 type = line ,
2921     Name-sg = Line ,
2922     name-sg = line ,
2923     Name-pl = Lines ,
2924     name-pl = lines ,
2925
2926 type = figure ,
2927     Name-sg = Figure ,
2928     name-sg = figure ,
2929     Name-pl = Figures ,
2930     name-pl = figures ,
2931     Name-sg-ab = Fig. ,
2932     name-sg-ab = fig. ,
2933     Name-pl-ab = Figs. ,
2934     name-pl-ab = figs. ,
2935
2936 type = table ,
2937     Name-sg = Table ,
2938     name-sg = table ,
2939     Name-pl = Tables ,

```

```

2940     name-pl = tables ,
2941
2942 type = item ,
2943     Name-sg = Item ,
2944     name-sg = item ,
2945     Name-pl = Items ,
2946     name-pl = items ,
2947
2948 type = footnote ,
2949     Name-sg = Footnote ,
2950     name-sg = footnote ,
2951     Name-pl = Footnotes ,
2952     name-pl = footnotes ,
2953
2954 type = note ,
2955     Name-sg = Note ,
2956     name-sg = note ,
2957     Name-pl = Notes ,
2958     name-pl = notes ,
2959
2960 type = equation ,
2961     Name-sg = Equation ,
2962     name-sg = equation ,
2963     Name-pl = Equations ,
2964     name-pl = equations ,
2965     Name-sg-ab = Eq. ,
2966     name-sg-ab = eq. ,
2967     Name-pl-ab = Eqs. ,
2968     name-pl-ab = eqs. ,
2969     refpre-in = {(} ,
2970     refpos-in = {)} ,
2971
2972 type = theorem ,
2973     Name-sg = Theorem ,
2974     name-sg = theorem ,
2975     Name-pl = Theorems ,
2976     name-pl = theorems ,
2977
2978 type = lemma ,
2979     Name-sg = Lemma ,
2980     name-sg = lemma ,
2981     Name-pl = Lemmas ,
2982     name-pl = lemmas ,
2983
2984 type = corollary ,
2985     Name-sg = Corollary ,
2986     name-sg = corollary ,
2987     Name-pl = Corollaries ,
2988     name-pl = corollaries ,
2989
2990 type = proposition ,
2991     Name-sg = Proposition ,
2992     name-sg = proposition ,
2993     Name-pl = Propositions ,

```

```

2994     name-pl = propositions ,
2995
2996 type = definition ,
2997     Name-sg = Definition ,
2998     name-sg = definition ,
2999     Name-pl = Definitions ,
3000     name-pl = definitions ,
3001
3002 type = proof ,
3003     Name-sg = Proof ,
3004     name-sg = proof ,
3005     Name-pl = Proofs ,
3006     name-pl = proofs ,
3007
3008 type = result ,
3009     Name-sg = Result ,
3010     name-sg = result ,
3011     Name-pl = Results ,
3012     name-pl = results ,
3013
3014 type = remark ,
3015     Name-sg = Remark ,
3016     name-sg = remark ,
3017     Name-pl = Remarks ,
3018     name-pl = remarks ,
3019
3020 type = example ,
3021     Name-sg = Example ,
3022     name-sg = example ,
3023     Name-pl = Examples ,
3024     name-pl = examples ,
3025
3026 type = algorithm ,
3027     Name-sg = Algorithm ,
3028     name-sg = algorithm ,
3029     Name-pl = Algorithms ,
3030     name-pl = algorithms ,
3031
3032 type = listing ,
3033     Name-sg = Listing ,
3034     name-sg = listing ,
3035     Name-pl = Listings ,
3036     name-pl = listings ,
3037
3038 type = exercise ,
3039     Name-sg = Exercise ,
3040     name-sg = exercise ,
3041     Name-pl = Exercises ,
3042     name-pl = exercises ,
3043
3044 type = solution ,
3045     Name-sg = Solution ,
3046     name-sg = solution ,
3047     Name-pl = Solutions ,

```

```

3048   name-pl = solutions ,
3049 </dict-english>

```

10.2 German

```

3050 <package>\zcDeclareLanguage { german }
3051 <package>\zcDeclareLanguageAlias { austrian      } { german }
3052 <package>\zcDeclareLanguageAlias { germanb       } { german }
3053 <package>\zcDeclareLanguageAlias { ngerman       } { german }
3054 <package>\zcDeclareLanguageAlias { naustrian     } { german }
3055 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
3056 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
3057 <*dict-german>

3058 namesep = {\nobreakspace} ,
3059 pairsep  = {\und\nobreakspace} ,
3060 listsep  = {,~} ,
3061 lastsep  = {\und\nobreakspace} ,
3062 tpairsep = {\und\nobreakspace} ,
3063 tlistsep = {,~} ,
3064 tlastsep = {\und\nobreakspace} ,
3065 notesep  = {~} ,
3066 rangesep = {\bis\nobreakspace} ,
3067
3068 type = part ,
3069   Name-sg = Teil ,
3070   name-sg = Teil ,
3071   Name-pl = Teile ,
3072   name-pl = Teile ,
3073
3074 type = chapter ,
3075   Name-sg = Kapitel ,
3076   name-sg = Kapitel ,
3077   Name-pl = Kapitel ,
3078   name-pl = Kapitel ,
3079
3080 type = section ,
3081   Name-sg = Abschnitt ,
3082   name-sg = Abschnitt ,
3083   Name-pl = Abschnitte ,
3084   name-pl = Abschnitte ,
3085
3086 type = paragraph ,
3087   Name-sg = Absatz ,
3088   name-sg = Absatz ,
3089   Name-pl = Absätze ,
3090   name-pl = Absätze ,
3091
3092 type = appendix ,
3093   Name-sg = Anhang ,
3094   name-sg = Anhang ,
3095   Name-pl = Anhänge ,
3096   name-pl = Anhänge ,
3097
3098 type = subappendix ,

```



```

3099     Name-sg = Anhang ,
3100     name-sg = Anhang ,
3101     Name-pl = Anhänge ,
3102     name-pl = Anhänge ,
3103
3104     type = page ,
3105     Name-sg = Seite ,
3106     name-sg = Seite ,
3107     Name-pl = Seiten ,
3108     name-pl = Seiten ,
3109
3110     type = line ,
3111     Name-sg = Zeile ,
3112     name-sg = Zeile ,
3113     Name-pl = Zeilen ,
3114     name-pl = Zeilen ,
3115
3116     type = figure ,
3117     Name-sg = Abbildung ,
3118     name-sg = Abbildung ,
3119     Name-pl = Abbildungen ,
3120     name-pl = Abbildungen ,
3121     Name-sg-ab = Abb. ,
3122     name-sg-ab = Abb. ,
3123     Name-pl-ab = Abb. ,
3124     name-pl-ab = Abb. ,
3125
3126     type = table ,
3127     Name-sg = Tabelle ,
3128     name-sg = Tabelle ,
3129     Name-pl = Tabellen ,
3130     name-pl = Tabellen ,
3131
3132     type = item ,
3133     Name-sg = Punkt ,
3134     name-sg = Punkt ,
3135     Name-pl = Punkte ,
3136     name-pl = Punkte ,
3137
3138     type = footnote ,
3139     Name-sg = Fußnote ,
3140     name-sg = Fußnote ,
3141     Name-pl = Fußnoten ,
3142     name-pl = Fußnoten ,
3143
3144     type = note ,
3145     Name-sg = Anmerkung ,
3146     name-sg = Anmerkung ,
3147     Name-pl = Anmerkungen ,
3148     name-pl = Anmerkungen ,
3149
3150     type = equation ,
3151     Name-sg = Gleichung ,
3152     name-sg = Gleichung ,

```

```

3153 Name-pl = Gleichungen ,
3154 name-pl = Gleichungen ,
3155 refpre-in = {()} ,
3156 refpos-in = {} } ,
3157
3158 type = theorem ,
3159 Name-sg = Theorem ,
3160 name-sg = Theorem ,
3161 Name-pl = Theoreme ,
3162 name-pl = Theoreme ,
3163
3164 type = lemma ,
3165 Name-sg = Lemma ,
3166 name-sg = Lemma ,
3167 Name-pl = Lemmata ,
3168 name-pl = Lemmata ,
3169
3170 type = corollary ,
3171 Name-sg = Korollar ,
3172 name-sg = Korollar ,
3173 Name-pl = Korollare ,
3174 name-pl = Korollare ,
3175
3176 type = proposition ,
3177 Name-sg = Satz ,
3178 name-sg = Satz ,
3179 Name-pl = Sätze ,
3180 name-pl = Sätze ,
3181
3182 type = definition ,
3183 Name-sg = Definition ,
3184 name-sg = Definition ,
3185 Name-pl = Definitionen ,
3186 name-pl = Definitionen ,
3187
3188 type = proof ,
3189 Name-sg = Beweis ,
3190 name-sg = Beweis ,
3191 Name-pl = Beweise ,
3192 name-pl = Beweise ,
3193
3194 type = result ,
3195 Name-sg = Ergebnis ,
3196 name-sg = Ergebnis ,
3197 Name-pl = Ergebnisse ,
3198 name-pl = Ergebnisse ,
3199
3200 type = remark ,
3201 Name-sg = Bemerkung ,
3202 name-sg = Bemerkung ,
3203 Name-pl = Bemerkungen ,
3204 name-pl = Bemerkungen ,
3205
3206 type = example ,

```

```

3207   Name-sg = Beispiel ,
3208   name-sg = Beispiel ,
3209   Name-pl = Beispiele ,
3210   name-pl = Beispiele ,
3211
3212   type = algorithm ,
3213   Name-sg = Algorithmus ,
3214   name-sg = Algorithmus ,
3215   Name-pl = Algorithmen ,
3216   name-pl = Algorithmen ,
3217
3218   type = listing ,
3219   Name-sg = Listing ,
3220   name-sg = Listing ,
3221   Name-pl = Listings ,
3222   name-pl = Listings ,
3223
3224   type = exercise ,
3225   Name-sg = Übungsaufgabe ,
3226   name-sg = Übungsaufgabe ,
3227   Name-pl = Übungsaufgaben ,
3228   name-pl = Übungsaufgaben ,
3229
3230   type = solution ,
3231   Name-sg = Lösung ,
3232   name-sg = Lösung ,
3233   Name-pl = Lösungen ,
3234   name-pl = Lösungen ,
3235 </dict-german>

```

10.3 French

```

3236 <package>\zcDeclareLanguage { french }
3237 <package>\zcDeclareLanguageAlias { acadian } { french }
3238 <package>\zcDeclareLanguageAlias { canadien } { french }
3239 <package>\zcDeclareLanguageAlias { francais } { french }
3240 <package>\zcDeclareLanguageAlias { frenchb } { french }
3241 <*dict-french>
3242 namesep = {\nobreakspace} ,
3243 pairsep = {\~et\nobreakspace} ,
3244 listsep = {,~} ,
3245 lastsep = {\~et\nobreakspace} ,
3246 tpairsep = {\~et\nobreakspace} ,
3247 tlistsep = {,~} ,
3248 tlastsep = {\~et\nobreakspace} ,
3249 notesep = {\~} ,
3250 rangesep = {\~à\nobreakspace} ,
3251
3252 type = part ,
3253   Name-sg = Partie ,
3254   name-sg = partie ,
3255   Name-pl = Parties ,
3256   name-pl = parties ,
3257

```

```

3258 type = chapter ,
3259     Name-sg = Chapitre ,
3260     name-sg = chapitre ,
3261     Name-pl = Chapitres ,
3262     name-pl = chapitres ,
3263
3264 type = section ,
3265     Name-sg = Section ,
3266     name-sg = section ,
3267     Name-pl = Sections ,
3268     name-pl = sections ,
3269
3270 type = paragraph ,
3271     Name-sg = Paragraphe ,
3272     name-sg = paragraphe ,
3273     Name-pl = Paragraphes ,
3274     name-pl = paragraphes ,
3275
3276 type = appendix ,
3277     Name-sg = Annexe ,
3278     name-sg = annexe ,
3279     Name-pl = Annexes ,
3280     name-pl = annexes ,
3281
3282 type = subappendix ,
3283     Name-sg = Annexe ,
3284     name-sg = annexe ,
3285     Name-pl = Annexes ,
3286     name-pl = annexes ,
3287
3288 type = page ,
3289     Name-sg = Page ,
3290     name-sg = page ,
3291     Name-pl = Pages ,
3292     name-pl = pages ,
3293
3294 type = line ,
3295     Name-sg = Ligne ,
3296     name-sg = ligne ,
3297     Name-pl = Lignes ,
3298     name-pl = lignes ,
3299
3300 type = figure ,
3301     Name-sg = Figure ,
3302     name-sg = figure ,
3303     Name-pl = Figures ,
3304     name-pl = figures ,
3305
3306 type = table ,
3307     Name-sg = Table ,
3308     name-sg = table ,
3309     Name-pl = Tables ,
3310     name-pl = tables ,
3311

```

```

3312 type = item ,
3313     Name-sg = Point ,
3314     name-sg = point ,
3315     Name-pl = Points ,
3316     name-pl = points ,
3317
3318 type = footnote ,
3319     Name-sg = Note ,
3320     name-sg = note ,
3321     Name-pl = Notes ,
3322     name-pl = notes ,
3323
3324 type = note ,
3325     Name-sg = Note ,
3326     name-sg = note ,
3327     Name-pl = Notes ,
3328     name-pl = notes ,
3329
3330 type = equation ,
3331     Name-sg = Équation ,
3332     name-sg = équation ,
3333     Name-pl = Équations ,
3334     name-pl = équations ,
3335     refpre-in = {()} ,
3336     refpos-in = {} ,
3337
3338 type = theorem ,
3339     Name-sg = Théorème ,
3340     name-sg = théorème ,
3341     Name-pl = Théorèmes ,
3342     name-pl = théorèmes ,
3343
3344 type = lemma ,
3345     Name-sg = Lemme ,
3346     name-sg = lemme ,
3347     Name-pl = Lemmes ,
3348     name-pl = lemmes ,
3349
3350 type = corollary ,
3351     Name-sg = Corollaire ,
3352     name-sg = corollaire ,
3353     Name-pl = Corollaires ,
3354     name-pl = corollaires ,
3355
3356 type = proposition ,
3357     Name-sg = Proposition ,
3358     name-sg = proposition ,
3359     Name-pl = Propositions ,
3360     name-pl = propositions ,
3361
3362 type = definition ,
3363     Name-sg = Définition ,
3364     name-sg = définition ,
3365     Name-pl = Définitions ,

```

```

3366     name-pl = définitions ,
3367
3368 type = proof ,
3369     Name-sg = Démonstration ,
3370     name-sg = démonstration ,
3371     Name-pl = Démonstrations ,
3372     name-pl = démonstrations ,
3373
3374 type = result ,
3375     Name-sg = Résultat ,
3376     name-sg = résultat ,
3377     Name-pl = Résultats ,
3378     name-pl = résultats ,
3379
3380 type = remark ,
3381     Name-sg = Remarque ,
3382     name-sg = remarque ,
3383     Name-pl = Remarques ,
3384     name-pl = remarques ,
3385
3386 type = example ,
3387     Name-sg = Exemple ,
3388     name-sg = exemple ,
3389     Name-pl = Exemples ,
3390     name-pl = exemples ,
3391
3392 type = algorithm ,
3393     Name-sg = Algorithme ,
3394     name-sg = algorithme ,
3395     Name-pl = Algorithmes ,
3396     name-pl = algorithmes ,
3397
3398 type = listing ,
3399     Name-sg = Liste ,
3400     name-sg = liste ,
3401     Name-pl = Listes ,
3402     name-pl = listes ,
3403
3404 type = exercise ,
3405     Name-sg = Exercice ,
3406     name-sg = exercice ,
3407     Name-pl = Exercices ,
3408     name-pl = exercices ,
3409
3410 type = solution ,
3411     Name-sg = Solution ,
3412     name-sg = solution ,
3413     Name-pl = Solutions ,
3414     name-pl = solutions ,
3415 </dict-french>

```

10.4 Portuguese

```

3416 <package>\zcDeclareLanguage { portuguese }
3417 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }

```

```

3418 <package>\zcDeclareLanguageAlias { brazil    } { portuguese }
3419 <package>\zcDeclareLanguageAlias { portuges  } { portuguese }
3420 <*dict-portuguese>

3421 namesep = {\nobreakspace} ,
3422 pairsep = {\~e\nobreakspace} ,
3423 listsep = {,~} ,
3424 lastsep = {\~e\nobreakspace} ,
3425 tpairsep = {\~e\nobreakspace} ,
3426 tlistsep = {,~} ,
3427 tlastsep = {\~e\nobreakspace} ,
3428 notesep = {\~} ,
3429 rangesep = {\~a\nobreakspace} ,
3430
3431 type = part ,
3432   Name-sg = Parte ,
3433   name-sg = parte ,
3434   Name-pl = Partes ,
3435   name-pl = partes ,
3436
3437 type = chapter ,
3438   Name-sg = Capítulo ,
3439   name-sg = capítulo ,
3440   Name-pl = Capítulos ,
3441   name-pl = capítulos ,
3442
3443 type = section ,
3444   Name-sg = Seção ,
3445   name-sg = seção ,
3446   Name-pl = Seções ,
3447   name-pl = seções ,
3448
3449 type = paragraph ,
3450   Name-sg = Parágrafo ,
3451   name-sg = parágrafo ,
3452   Name-pl = Parágrafos ,
3453   name-pl = parágrafos ,
3454   Name-sg-ab = Par. ,
3455   name-sg-ab = par. ,
3456   Name-pl-ab = Par. ,
3457   name-pl-ab = par. ,
3458
3459 type = appendix ,
3460   Name-sg = Apêndice ,
3461   name-sg = apêndice ,
3462   Name-pl = Apêndices ,
3463   name-pl = apêndices ,
3464
3465 type = subappendix ,
3466   Name-sg = Apêndice ,
3467   name-sg = apêndice ,
3468   Name-pl = Apêndices ,
3469   name-pl = apêndices ,
3470

```

```

3471 type = page ,
3472     Name-sg = Página ,
3473     name-sg = página ,
3474     Name-pl = Páginas ,
3475     name-pl = páginas ,
3476     name-sg-ab = p. ,
3477     name-pl-ab = pp. ,
3478
3479 type = line ,
3480     Name-sg = Linha ,
3481     name-sg = linha ,
3482     Name-pl = Linhas ,
3483     name-pl = linhas ,
3484
3485 type = figure ,
3486     Name-sg = Figura ,
3487     name-sg = figura ,
3488     Name-pl = Figuras ,
3489     name-pl = figuras ,
3490     Name-sg-ab = Fig. ,
3491     name-sg-ab = fig. ,
3492     Name-pl-ab = Figs. ,
3493     name-pl-ab = figs. ,
3494
3495 type = table ,
3496     Name-sg = Tabela ,
3497     name-sg = tabela ,
3498     Name-pl = Tabelas ,
3499     name-pl = tabelas ,
3500
3501 type = item ,
3502     Name-sg = Item ,
3503     name-sg = item ,
3504     Name-pl = Itens ,
3505     name-pl = itens ,
3506
3507 type = footnote ,
3508     Name-sg = Nota ,
3509     name-sg = nota ,
3510     Name-pl = Notas ,
3511     name-pl = notas ,
3512
3513 type = note ,
3514     Name-sg = Nota ,
3515     name-sg = nota ,
3516     Name-pl = Notas ,
3517     name-pl = notas ,
3518
3519 type = equation ,
3520     Name-sg = Equação ,
3521     name-sg = equação ,
3522     Name-pl = Equações ,
3523     name-pl = equações ,
3524     Name-sg-ab = Eq. ,

```



```

3525 name-sg-ab = eq. ,
3526 Name-pl-ab = Eqs. ,
3527 name-pl-ab = eqs. ,
3528 refpre-in = {} ,
3529 refpos-in = {} ,
3530
3531 type = theorem ,
3532 Name-sg = Teorema ,
3533 name-sg = teorema ,
3534 Name-pl = Teoremas ,
3535 name-pl = teoremas ,
3536
3537 type = lemma ,
3538 Name-sg = Lema ,
3539 name-sg = lema ,
3540 Name-pl = Lemas ,
3541 name-pl = lemas ,
3542
3543 type = corollary ,
3544 Name-sg = Corolário ,
3545 name-sg = corolário ,
3546 Name-pl = Corolários ,
3547 name-pl = corolários ,
3548
3549 type = proposition ,
3550 Name-sg = Proposição ,
3551 name-sg = proposição ,
3552 Name-pl = Proposições ,
3553 name-pl = proposições ,
3554
3555 type = definition ,
3556 Name-sg = Definição ,
3557 name-sg = definição ,
3558 Name-pl = Definições ,
3559 name-pl = definições ,
3560
3561 type = proof ,
3562 Name-sg = Demonstração ,
3563 name-sg = demonstração ,
3564 Name-pl = Demonstrações ,
3565 name-pl = demonstrações ,
3566
3567 type = result ,
3568 Name-sg = Resultado ,
3569 name-sg = resultado ,
3570 Name-pl = Resultados ,
3571 name-pl = resultados ,
3572
3573 type = remark ,
3574 Name-sg = Observação ,
3575 name-sg = observação ,
3576 Name-pl = Observações ,
3577 name-pl = observações ,
3578

```

```

3579 type = example ,
3580   Name-sg = Exemplo ,
3581   name-sg = exemplo ,
3582   Name-pl = Exemplos ,
3583   name-pl = exemplos ,
3584
3585 type = algorithm ,
3586   Name-sg = Algoritmo ,
3587   name-sg = algoritmo ,
3588   Name-pl = Algoritmos ,
3589   name-pl = algoritmos ,
3590
3591 type = listing ,
3592   Name-sg = Listagem ,
3593   name-sg = listagem ,
3594   Name-pl = Listagens ,
3595   name-pl = listagens ,
3596
3597 type = exercise ,
3598   Name-sg = Exercício ,
3599   name-sg = exercício ,
3600   Name-pl = Exercícios ,
3601   name-pl = exercícios ,
3602
3603 type = solution ,
3604   Name-sg = Solução ,
3605   name-sg = solução ,
3606   Name-pl = Soluções ,
3607   name-pl = soluções ,
3608 </dict-portuguese>

```

10.5 Spanish

```

3609 <package>\zcDeclareLanguage { spanish }
3610 <*dict-spanish>
3611 namesep = {\nobreakspace} ,
3612 pairsep = {\~y\nobreakspace} ,
3613 listsep = {,~} ,
3614 lastsep = {\~y\nobreakspace} ,
3615 tpairsep = {\~y\nobreakspace} ,
3616 tlistsep = {,~} ,
3617 tlastsep = {\~y\nobreakspace} ,
3618 notesep = {\~} ,
3619 rangesep = {\~a\nobreakspace} ,
3620
3621 type = part ,
3622   Name-sg = Parte ,
3623   name-sg = parte ,
3624   Name-pl = Partes ,
3625   name-pl = partes ,
3626
3627 type = chapter ,
3628   Name-sg = Capítulo ,
3629   name-sg = capítulo ,

```

```

3630     Name-pl = Capítulos ,
3631     name-pl = capítulos ,
3632
3633     type = section ,
3634     Name-sg = Sección ,
3635     name-sg = sección ,
3636     Name-pl = Secciones ,
3637     name-pl = secciones ,
3638
3639     type = paragraph ,
3640     Name-sg = Párrafo ,
3641     name-sg = párrafo ,
3642     Name-pl = Párrafos ,
3643     name-pl = párrafos ,
3644
3645     type = appendix ,
3646     Name-sg = Apéndice ,
3647     name-sg = apéndice ,
3648     Name-pl = Apéndices ,
3649     name-pl = apéndices ,
3650
3651     type = subappendix ,
3652     Name-sg = Apéndice ,
3653     name-sg = apéndice ,
3654     Name-pl = Apéndices ,
3655     name-pl = apéndices ,
3656
3657     type = page ,
3658     Name-sg = Página ,
3659     name-sg = página ,
3660     Name-pl = Páginas ,
3661     name-pl = páginas ,
3662
3663     type = line ,
3664     Name-sg = Línea ,
3665     name-sg = línea ,
3666     Name-pl = Líneas ,
3667     name-pl = líneas ,
3668
3669     type = figure ,
3670     Name-sg = Figura ,
3671     name-sg = figura ,
3672     Name-pl = Figuras ,
3673     name-pl = figuras ,
3674
3675     type = table ,
3676     Name-sg = Cuadro ,
3677     name-sg = cuadro ,
3678     Name-pl = Cuadros ,
3679     name-pl = cuadros ,
3680
3681     type = item ,
3682     Name-sg = Punto ,
3683     name-sg = punto ,

```

```

3684     Name-pl = Puntos ,
3685     name-pl = puntos ,
3686
3687 type = footnote ,
3688     Name-sg = Nota ,
3689     name-sg = nota ,
3690     Name-pl = Notas ,
3691     name-pl = notas ,
3692
3693 type = note ,
3694     Name-sg = Nota ,
3695     name-sg = nota ,
3696     Name-pl = Notas ,
3697     name-pl = notas ,
3698
3699 type = equation ,
3700     Name-sg = Ecuación ,
3701     name-sg = ecuación ,
3702     Name-pl = Ecuaciones ,
3703     name-pl = ecuaciones ,
3704     refpre-in = {() ,
3705     refpos-in = {)} ,
3706
3707 type = theorem ,
3708     Name-sg = Teorema ,
3709     name-sg = teorema ,
3710     Name-pl = Teoremas ,
3711     name-pl = teoremas ,
3712
3713 type = lemma ,
3714     Name-sg = Lema ,
3715     name-sg = lema ,
3716     Name-pl = Lemas ,
3717     name-pl = lemas ,
3718
3719 type = corollary ,
3720     Name-sg = Corolario ,
3721     name-sg = corolario ,
3722     Name-pl = Corolarios ,
3723     name-pl = corolarios ,
3724
3725 type = proposition ,
3726     Name-sg = Proposición ,
3727     name-sg = proposición ,
3728     Name-pl = Proposiciones ,
3729     name-pl = proposiciones ,
3730
3731 type = definition ,
3732     Name-sg = Definición ,
3733     name-sg = definición ,
3734     Name-pl = Definiciones ,
3735     name-pl = definiciones ,
3736
3737 type = proof ,

```

```

3738     Name-sg = Demostración ,
3739     name-sg = demostración ,
3740     Name-pl = Demostraciones ,
3741     name-pl = demostraciones ,
3742
3743     type = result ,
3744     Name-sg = Resultado ,
3745     name-sg = resultado ,
3746     Name-pl = Resultados ,
3747     name-pl = resultados ,
3748
3749     type = remark ,
3750     Name-sg = Observación ,
3751     name-sg = observación ,
3752     Name-pl = Observaciones ,
3753     name-pl = observaciones ,
3754
3755     type = example ,
3756     Name-sg = Ejemplo ,
3757     name-sg = ejemplo ,
3758     Name-pl = Ejemplos ,
3759     name-pl = ejemplos ,
3760
3761     type = algorithm ,
3762     Name-sg = Algoritmo ,
3763     name-sg = algoritmo ,
3764     Name-pl = Algoritmos ,
3765     name-pl = algoritmos ,
3766
3767     type = listing ,
3768     Name-sg = Listado ,
3769     name-sg = listado ,
3770     Name-pl = Listados ,
3771     name-pl = listados ,
3772
3773     type = exercise ,
3774     Name-sg = Ejercicio ,
3775     name-sg = ejercicio ,
3776     Name-pl = Ejercicios ,
3777     name-pl = ejercicios ,
3778
3779     type = solution ,
3780     Name-sg = Solución ,
3781     name-sg = solución ,
3782     Name-pl = Soluciones ,
3783     name-pl = soluciones ,
3784 </dict-spanish>

```

Index

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

Symbols	
<code>\\</code>	97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145
† internal commands:	
<code>\t_zrefclever_current_counter_tl</code>	4
A	
<code>\AddToHook</code>	85, 478, 493, 637, 673, 698, 736, 738, 790, 811, 2615, 2626, 2628, 2633, 2646, 2666, 2681, 2683, 2688, 2704, 2722, 2742, 2774, 2783, 2801, 2824
<code>\appendix</code>	70, 71
<code>\appendixname</code>	70
B	
<code>\babelname</code>	683
<code>\babelprovide</code>	12, 23
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_if:NTF</code> 308, 319, 641, 645, 1563, 1658, 1788, 1810, 1841, 1887, 1928, 1951, 1955, 1961, 1971, 1977, 2134	
<code>\bool_if:nTF</code>	53, 1213, 1222, 1231, 1299, 1309, 1333, 1350, 1365, 1433, 1441, 1577, 1585, 1822, 1829, 1836, 2086, 2208
<code>\bool_lazy_all:nTF</code>	2537
<code>\bool_lazy_and:nnTF</code>	1107, 1125, 2283, 2594
<code>\bool_lazy_any:nTF</code>	2361, 2370
<code>\bool_lazy_or:nnTF</code>	1111, 2271
<code>\bool_new:N</code>	271, 514, 515, 540, 564, 573, 580, 581, 594, 595, 614, 615, 804, 805, 1136, 1149, 1473, 1474, 1484, 1490, 1491
<code>\bool_set:Nn</code>	1105
<code>\bool_set_false:N</code>	527, 531, 622, 631, 632, 647, 827, 1291, 1530, 1569, 1583, 1597, 1800, 1926, 1927, 2368, 2385
<code>\bool_set_true:N</code>	328, 521, 522, 526, 532, 621, 626, 627, 815, 820, 1305, 1315, 1319, 1341, 1356, 1371, 1398, 1538, 1564, 1570, 1574, 1601, 1607, 2384, 2441, 2454, 2456, 2494, 2509, 2521
<code>\bool_until_do:Nn</code>	1331, 1531
<code>\bool_while_do:nn</code>	2829
C	
clist commands:	
<code>\clist_map_inline:nn</code>	870, 2755
<code>\counterwithin</code>	4
cs commands:	
<code>\cs_generate_variant:Nn</code>	50, 179, 185, 324, 332, 950, 1016, 1022, 1277, 2128, 2407
<code>\cs_if_exist:NTF</code> ...	43, 63, 2654, 2656
<code>\cs_if_exist_p:N</code>	2831
<code>\cs_new:Npn</code>	41, 51, 61, 72, 180, 187, 2082, 2129, 2388
<code>\cs_new_nopar:Npn</code>	2619
<code>\cs_new_protected:Npn</code>	174, 272, 325, 333, 339, 460, 948, 1011, 1017, 1100, 1153, 1195, 1206, 1278, 1411, 1463, 1507, 1665, 1922, 2078, 2080, 2261, 2408, 2531, 2588
<code>\cs_new_protected:Npx</code>	84
<code>\cs_set_eq:NN</code>	88, 2617, 2724, 2725, 2729, 2730, 2738, 2739, 2744, 2745, 2750, 2751
<code>\cs_set_nopar:Npn</code>	2708
E	
<code>\endinput</code>	12
exp commands:	
<code>\exp_args:NNe</code>	27, 30
<code>\exp_args:NNNo</code>	176
<code>\exp_args:NNnx</code>	262
<code>\exp_args:NNo</code>	176, 182
<code>\exp_args:NnV</code>	300
<code>\exp_args:Nnx</code>	335
<code>\exp_args:No</code>	182
<code>\exp_args:Nx</code>	282
<code>\exp_args:Nxx</code>	1249, 2421, 2463, 2473, 2511
<code>\exp_not:N</code>	58, 1843, 1846, 1857, 1860, 1863, 2092, 2095, 2098, 2107, 2109, 2112, 2115, 2121, 2123, 2141, 2152, 2155, 2157, 2160, 2167, 2174, 2176, 2180, 2183, 2186, 2198, 2201, 2214, 2217, 2220, 2237, 2239, 2242, 2245, 2252, 2254
<code>\exp_not:n</code>	183, 1687, 1703, 1715, 1720, 1743, 1757, 1761, 1773, 1777, 1811,

1812, 1844, 1856, 1861, 1862, 1991, 2004, 2011, 2035, 2047, 2051, 2061, 2065, 2093, 2094, 2096, 2102, 2105, 2108, 2113, 2114, 2116, 2117, 2120, 2122, 2153, 2154, 2156, 2158, 2159, 2161, 2162, 2166, 2178, 2179, 2184, 2185, 2187, 2195, 2199, 2200, 2202, 2215, 2216, 2218, 2231, 2235, 2238, 2243, 2244, 2246, 2247, 2251, 2253		iow commands:	
\ExplSyntaxOn 12, 284		\iow_char:N 97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145	
F		K	
file commands:		keys commands:	
\file_get:nnTF 282		\keys_define:nn 31, 345, 357, 374, 388, 467, 497, 504, 516, 541, 550, 565, 574, 582, 596, 608, 616, 649, 656, 694, 741, 783, 785, 792, 799, 806, 816, 828, 837, 866, 892, 916, 926, 937, 961, 973, 1023, 1035, 1056, 1079	
\fmtversion 3		\keys_set:nn 12, 31, 35, 301, 821, 949, 956, 1005, 1103	
\footnote 70		keyval commands:	
G		\keyval_parse:nnn 841, 896	
group commands:		L	
\group_begin: . . 87, 274, 327, 1000, 1102, 1115, 1843, 1860, 2092, 2095, 2112, 2115, 2152, 2157, 2160, 2176, 2183, 2198, 2214, 2217, 2242, 2245		\label 70, 2617, 2715	
\group_end: 90, 322, 330, 1008, 1118, 1133, 1857, 1863, 2107, 2109, 2121, 2123, 2155, 2167, 2174, 2180, 2186, 2201, 2237, 2239, 2252, 2254		\labelformat 3	
		\language name 23, 677	
I		M	
\IfBooleanTF 1139		\mainbabelname 23, 684	
\IfFormatAtLeastTF 3, 4, 2718		\MessageBreak 10	
\input 12		msg commands:	
int commands:		\msg_info:nnn 365, 395, 2700, 2797, 2820, 2848	
\int_case:nnTF 1668, 1696, 1728, 1890, 1984, 2023		\msg_line_context: 96, 102, 106, 108, 111, 117, 123, 130, 133, 139, 144, 151, 156, 160, 162, 165, 169	
\int_compare:nNnTF 1259, 1342, 1357, 1372, 1387, 1399, 1419, 1421, 1465, 1629, 1683, 1717, 1879, 1881, 1939, 1964, 2008, 2431, 2443, 2483, 2498, 2847		\msg_new:nnn 94, 100, 105, 107, 109, 115, 121, 127, 129, 131, 137, 142, 147, 149, 154, 159, 161, 163, 168, 170, 172	
\int_compare_p:nNn 1435, 1443, 2275, 2286, 2381		\msg_note:nnn 304	
\int_eval:n 84		\msg_warning:nn 483, 508, 646, 652, 795, 832	
\int_incr:N 1917, 1954, 1956, 1970, 1972, 1976, 1978, 2076, 2845		\msg_warning:nnn 251, 266, 310, 320, 724, 769, 898, 965, 1007, 1047, 1086, 1622, 1795, 2314, 2349, 2580	
\int_new:N 1150, 1151, 1475, 1476, 1487, 1488		\msg_warning:nnnn 843	
\int_set:Nn 1420, 1422, 1426, 1429, 2828		N	
\int_to_roman:n 2832, 2839, 2840, 2843		\newcounter 4, 2650, 2651	
\int_use:N 37, 39, 45		\NewDocumentCommand 246, 256, 946, 951, 998, 1098, 1137	
\int_zero:N 1413, 1414, 1516, 1517, 1518, 1519, 1916, 1918, 1919, 2071, 2072		\nobreakspace 409, 2862, 2863, 2865, 2866, 2868, 2870, 3058, 3059, 3061, 3062, 3064, 3066, 3242, 3243, 3245, 3246, 3248, 3250, 3421, 3422, 3424, 3425, 3427, 3429, 3611, 3612, 3614, 3615, 3617, 3619	
\l_tmpa_int 2828, 2832, 2839, 2840, 2843, 2845, 2847			

P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	6
<code>\pageref</code>	36
prg commands:	
<code>\prg_generate_conditional_</code> variant:Nnn	433, 449
<code>\prg_new_protected_conditional:Npnn</code>	419, 435, 452
<code>\prg_return_false:</code>	429, 431, 445, 447, 458
<code>\prg_return_true:</code>	428, 444, 457
<code>\ProcessKeysOptions</code>	945
prop commands:	
<code>\prop_get:NnN</code>	2557
<code>\prop_get:NnNTF</code>	275, 422, 425, 438, 441, 455, 1001, 2299, 2320, 2328, 2534, 2591, 2604
<code>\prop_gput:Nnn</code>	252, 263, 1013, 1019
<code>\prop_gput_if_new:Nnn</code>	335, 341
<code>\prop_gset_from_keyval:Nn</code>	403
<code>\prop_if_exist:NTF</code>	287, 953
<code>\prop_if_exist_p:N</code>	2541, 2597
<code>\prop_if_in:NnTF</code>	27, 250, 260, 719, 764
<code>\prop_if_in_p:Nn</code>	54, 2548
<code>\prop_item:Nn</code>	30, 55, 264
<code>\prop_new:N</code>	245, 293, 402, 836, 891, 922, 954
<code>\prop_put:Nnn</code>	464, 933, 988
<code>\prop_remove:Nn</code>	463, 932, 980
<code>\providecommand</code>	3
<code>\ProvidesExplPackage</code>	14
<code>\ProvidesFile</code>	12
R	
<code>\refstepcounter</code>	3, 70, 75
<code>\renewlist</code>	75
<code>\RequirePackage</code>	16, 17, 18, 19, 642, 787, 808
S	
<code>\scantokens</code>	71
seq commands:	
<code>\seq_clear:N</code>	561, 1155
<code>\seq_const_from_clist:Nn</code>	191, 199, 212, 224
<code>\seq_gconcat:NNN</code>	232, 235, 239, 242
<code>\seq_get_left:NN</code>	1541
<code>\seq_gput_right:Nn</code>	302, 313
<code>\seq_if_empty:NTF</code>	1535
<code>\seq_if_in:NnTF</code>	278, 872, 1199
<code>\seq_map_break:n</code>	75, 1454, 1457
<code>\seq_map_function:NN</code>	1158
<code>\seq_map_indexed_inline:Nn</code>	20, 1415
<code>\seq_map_inline:Nn</code>	354, 371, 385, 923, 958, 970, 1032, 1053, 1076, 1451
<code>\seq_map_tokens:Nn</code>	57
<code>\seq_new:N</code>	231, 238, 270, 549, 865, 1135, 1152, 1472
<code>\seq_pop_left:NN</code>	1533
<code>\seq_put_right:Nn</code>	874, 1202
<code>\seq_reverse:N</code>	555
<code>\seq_set_eq:NN</code>	1509
<code>\seq_set_from_clist:Nn</code>	554, 1104
<code>\seq_sort:Nn</code>	38, 1161
<code>\setcounter</code>	2652, 2653, 2669, 2682, 2686
sort commands:	
<code>\sort_return_same:</code>	38, 44, 1168, 1173, 1220, 1270, 1272, 1306, 1326, 1347, 1362, 1379, 1404, 1439, 1454, 1470
<code>\sort_return_swapped:</code>	38, 44, 1181, 1229, 1269, 1316, 1325, 1346, 1361, 1380, 1403, 1447, 1457, 1469
<code>\stepcounter</code>	2668, 2685
str commands:	
<code>\str_case:nnTF</code>	700, 745
<code>\str_compare:nnTF</code>	1322
<code>\str_if_eq:nnTF</code>	74
<code>\str_if_eq_p:nn</code>	2366, 2372, 2374, 2378
<code>\str_new:N</code>	655
<code>\str_set:Nn</code>	660, 662, 664, 666
T	
<code>\tag</code>	2776
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@Alpha</code>	70
<code>\@addtoreset</code>	4
<code>\@chapapp</code>	70
<code>\@currentcounter</code>	3, 4, 29, 70, 75, 920
<code>\@currentlabel</code>	3, 75
<code>\@elt</code>	4
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	480, 495, 639, 675, 681, 813, 2648, 2706, 2720, 2734, 2736, 2803, 2826
<code>\@onlypreamble</code>	255, 269, 1010
<code>\bbl@loaded</code>	23
<code>\bbl@main@language</code>	23, 678
<code>\c@</code>	4
<code>\c@enumN</code>	75
<code>\c@lstnumber</code>	75
<code>\c@page</code>	6, 88
<code>\c@l@</code>	4, 5
<code>\hyper@link</code>	58, 1846, 2098, 2141, 2220
<code>\lst@AddToHook</code>	2814
<code>\lst@label</code>	2816, 2817

zrefclever internal commands:

\l_zrefclever_abbrev_bool
 594, 598, 2284
 \l_zrefclever_capitalize_bool ..
 580, 584, 2272
 \l_zrefclever_capitalize_first_-
 bool 581, 590, 2274
 _zrefclever_counter_reset_by:n
 5, 28, 43, 45, 47, 51, 2790
 _zrefclever_counter_reset_by_-
 aux:nn 58, 61
 _zrefclever_counter_reset_by_-
 auxi:nnn 68, 72
 \l_zrefclever_counter_resetby_-
 prop 5, 28, 54, 55, 891, 903
 \l_zrefclever_counter_resettters_-
 seq 4, 5, 28, 57, 865, 872, 875
 \l_zrefclever_counter_type_prop
 3, 27, 27, 30, 836, 848
 \l_zrefclever_current_counter_-
 tl 3,
 29, 20, 23, 28, 31, 33, 37, 80, 915, 918
 \l_zrefclever_current_language_-
 tl .. 23, 672, 677, 683, 687, 713, 758
 _zrefclever_declare_default_-
 transl:nnn ... 33, 1011, 1042, 1063
 _zrefclever_declare_type_-
 transl:nnnn ... 33, 1011, 1068, 1090
 _zrefclever_def_extract_-
 default:Nnnn 8,
 174, 1197, 1208, 1210, 1280, 1283,
 1286, 1288, 1551, 1554, 2410, 2412
 \g_zrefclever_dict_(language)_prop
 12
 \l_zrefclever_dict_language_tl .
 . 189, 276, 280, 283, 290, 296, 303,
 305, 311, 314, 336, 342, 423, 426,
 439, 442, 1002, 1043, 1064, 1069, 1091
 _zrefclever_extract_default:nnn
 .. 9, 187, 1261, 1266, 1343, 1345,
 1358, 1376, 1466, 1468, 2433, 2438,
 2445, 2450, 2485, 2490, 2500, 2505
 _zrefclever_extract_default_-
 unexp:nnn 9,
 65, 180, 1251, 1255, 1852, 2100,
 2103, 2118, 2147, 2163, 2226, 2232,
 2248, 2392, 2398, 2402, 2423, 2427,
 2465, 2469, 2475, 2479, 2513, 2517
 _zrefclever_extract_url_-
 unexp:n 1848, 2099, 2143, 2222, 2388
 \g_zrefclever_fallback_dict_-
 prop 9, 402, 403, 455
 \l_zrefclever_footnote_type_tl .
 2624, 2625, 2627, 2631

_zrefclever_get_default_-
 transl:nnN 9, 436, 450
 _zrefclever_get_default_-
 transl:nnNTF 17, 435, 2572
 _zrefclever_get_enclosing_-
 counters_value:n ... 5, 41, 46, 79
 _zrefclever_get_fallback_-
 transl:nN 453
 _zrefclever_get_fallback_-
 transl:nNTF 17, 451, 2577
 _zrefclever_get_ref:n
 58, 1688, 1704,
 1716, 1721, 1744, 1758, 1762, 1774,
 1778, 1813, 1833, 1992, 2005, 2012,
 2036, 2048, 2052, 2062, 2066, 2082
 _zrefclever_get_ref_first: ...
 58, 62, 1826, 1873, 2129
 _zrefclever_get_ref_font:nN . 9,
 16, 29, 68, 69, 1649, 1651, 1653, 2588
 _zrefclever_get_ref_string:nN .
 9, 10, 16, 29, 68, 1121, 1522,
 1524, 1526, 1631, 1633, 1635, 1637,
 1639, 1641, 1643, 1645, 1647, 2531
 _zrefclever_get_type_transl:nnnN
 9, 420, 434
 _zrefclever_get_type_transl:nnnNTF
 17, 419, 2307, 2336, 2342, 2566
 \l_zrefclever_label_a_tl
 . 44, 1477, 1534, 1552, 1566, 1616,
 1617, 1623, 1675, 1688, 1704, 1721,
 1762, 1778, 1806, 1813, 1930, 1934,
 1944, 1969, 1992, 2013, 2052, 2066
 \l_zrefclever_label_b_tl
 44, 1477,
 1537, 1542, 1555, 1568, 1573, 1934
 \l_zrefclever_label_count_int ..
 45, 1475,
 1516, 1629, 1668, 1916, 1939, 2076
 \l_zrefclever_label_enclval_a_-
 tl 1143, 1280, 1282, 1337,
 1353, 1373, 1388, 1392, 1393, 1400
 \l_zrefclever_label_enclval_b_-
 tl 1143, 1283, 1285, 1338,
 1360, 1368, 1390, 1394, 1395, 1402
 \l_zrefclever_label_extdoc_a_tl
 1143, 1286,
 1296, 1301, 1311, 1324, 2410, 2416
 \l_zrefclever_label_extdoc_b_tl
 1143, 1288,
 1297, 1302, 1312, 1323, 2412, 2417
 \l_zrefclever_label_type_a_tl ..
 68, 1143, 1198, 1200,
 1203, 1209, 1217, 1226, 1234, 1239,
 1425, 1453, 1547, 1552, 1580, 1588,

1594, 1620, 1677, 1946, 2539, 2544,
 2551, 2560, 2568, 2595, 2600, 2607
 \l_zrefclever_label_type_b_tl ...
 1143,
 1211, 1218, 1227, 1235, 1240, 1428,
 1456, 1548, 1555, 1581, 1590, 1595
 _zrefclever_label_type_put_-
 new_right:n 37, 38, 1159, 1195
 \l_zrefclever_label_types_seq ...
 38, 1152, 1155, 1199, 1202, 1451
 _zrefclever_labels_in_sequence:nn
 45, 65, 1804, 1933, 2408
 \g_zrefclever_languages_prop ...
 11, 245, 250, 252, 260,
 263, 264, 275, 422, 438, 719, 764, 1001
 \l_zrefclever_last_of_type_bool
 45, 1472, 1564, 1569, 1570,
 1574, 1583, 1598, 1602, 1608, 1658
 \l_zrefclever_lastsep_tl . 1492,
 1640, 1703, 1720, 1743, 1761, 1773
 \l_zrefclever_link_star_bool ...
 1105, 1135, 2089, 2211, 2364
 \l_zrefclever_listsep_tl
 ... 1492, 1638, 1715, 1757, 1991,
 2004, 2011, 2035, 2047, 2051, 2061
 \l_zrefclever_load_dict_-
 verbose_bool ... 271, 308, 319, 328
 \g_zrefclever_loaded_dictionaries_-
 seq 270, 279, 302, 313
 _zrefclever_ltxlabel:n
 .. 2708, 2725, 2730, 2739, 2745, 2751
 \l_zrefclever_main_language_tl .
 23, 671,
 678, 684, 688, 692, 705, 727, 750, 772
 _zrefclever_name_default:
 2078, 2194
 \l_zrefclever_name_format_-
 fallback_tl
 .. 1483, 2291, 2295, 2297, 2333, 2345
 \l_zrefclever_name_format_tl ...
 ... 1483, 2277, 2278, 2281, 2282,
 2292, 2293, 2304, 2310, 2325, 2339
 \l_zrefclever_name_in_link_bool
 59,
 62, 1483, 1841, 2134, 2368, 2384, 2385
 \l_zrefclever_namefont_tl 1492,
 1650, 1844, 1861, 2153, 2184, 2199
 \l_zrefclever_nameinlink_str ...
 655, 660,
 662, 664, 666, 2366, 2372, 2374, 2378
 \l_zrefclever_namesep_tl
 .. 1492, 1632, 2156, 2187, 2195, 2202
 \l_zrefclever_next_is_same_bool
 45, 65, 1487,
 1927, 1955, 1971, 1977, 2457, 2522
 \l_zrefclever_next_maybe_range_-
 bool
 .. 45, 65, 1487, 1800, 1810, 1926,
 1951, 1961, 2441, 2455, 2495, 2510
 \l_zrefclever_noabbrev_first_-
 bool 595, 604, 2288
 _zrefclever_orig_label:n 2617, 2621
 _zrefclever_orig_ltxlabel:n ...
 .. 2710, 2724, 2729, 2738, 2744, 2750
 _zrefclever_page_format_aux: ..
 84, 88
 \g_zrefclever_page_format_tl ...
 6, 83, 89, 92
 \l_zrefclever_pairsep_tl
 1492, 1636, 1687, 1811
 _zrefclever_prop_put_non_-
 empty:Nnn 18, 460, 847, 902
 _zrefclever_provide_dict_-
 default_transl:nn 14, 333, 363, 380
 _zrefclever_provide_dict_type_-
 transl:nn 14, 333, 381, 398
 _zrefclever_provide_dictionary:n
 9, 12-15,
 35, 272, 329, 740, 751, 759, 774, 1106
 _zrefclever_provide_dictionary_-
 verbose:n ... 14, 325, 706, 714, 729
 \l_zrefclever_range_beg_label_-
 tl 45, 1487, 1515,
 1716, 1739, 1745, 1755, 1759, 1771,
 1775, 1915, 1953, 1968, 2002, 2006,
 2033, 2037, 2045, 2049, 2059, 2063
 \l_zrefclever_range_count_int ..
 45,
 1487, 1518, 1696, 1730, 1918, 1954,
 1965, 1970, 1976, 1984, 2025, 2071
 \l_zrefclever_range_same_count_-
 int 45,
 1487, 1519, 1683, 1718, 1731, 1919,
 1956, 1972, 1978, 2009, 2026, 2072
 \l_zrefclever_rangesep_tl
 1492, 1634, 1777, 1812, 2065
 _zrefclever_ref_default:
 2078, 2126, 2132, 2188, 2257
 \l_zrefclever_ref_language_tl ..
 23, 24, 670, 691,
 704, 707, 712, 715, 721, 726, 730,
 740, 749, 752, 757, 760, 766, 771,
 775, 1106, 2308, 2337, 2343, 2567, 2573
 \c_zrefclever_ref_options_font_-
 seq 10, 16, 191
 \c_zrefclever_ref_options_-
 necessarily_not_type_specific_-
 seq 16, 191, 355, 959, 1033

\c_zrefclever_ref_options_- necessarily_type_specific_seq	\l_zrefclever_sort_prior_b_int
191, 386, 1077	1150, 1414, 1421, 1422, 1429, 1437, 1445
\c_zrefclever_ref_options_- possibly_type_specific_seq	\l_zrefclever_tlastsep_tl
16, 191, 372, 1054	1492, 1527, 1904
\l_zrefclever_ref_options_prop	\l_zrefclever_tlistsep_tl
29, 31, 922, 932, 933, 2534, 2591	1492, 1525, 1882
\c_zrefclever_ref_options_- reference_seq	\l_zrefclever_tpairsep_tl
191, 924	1492, 1523, 1898
\c_zrefclever_ref_options_- typesetup_seq	\l_zrefclever_type_<type>- options_prop
191, 971	31
\l_zrefclever_ref_property_tl	\l_zrefclever_type_count_int
18, 466, 471, 473, 475, 481, 484, 500, 509, 1156, 1188, 1545, 2084, 2138, 2206, 2419	45, 62, 1475, 1517, 1879, 1881, 1890, 1917, 2275, 2287, 2381
\l_zrefclever_ref_typeset_font_- tl	\l_zrefclever_type_first_label_- tl
782, 784, 1116	45, 59, 1477, 1513, 1674, 1792, 1801, 1805, 1833, 1849, 1853, 1913, 1943, 2131, 2137, 2144, 2148, 2164, 2205, 2223, 2227, 2233, 2249, 2263
\l_zrefclever_reffont_in_tl 1492, 1654, 2096, 2116, 2161, 2218, 2246	\l_zrefclever_type_first_label_- type_tl
\l_zrefclever_reffont_out_tl	45, 62, 1477, 1514, 1676, 1796, 1914, 1945, 2266, 2302, 2309, 2315, 2323, 2331, 2338, 2344, 2351
1492, 1652, 2093, 2113, 2158, 2178, 2215, 2243	_zrefclever_type_name_setup:
\l_zrefclever_refpos_in_tl 1492, 1648, 2105, 2120, 2166, 2235, 2251	9, 10, 59, 1821, 2261
\l_zrefclever_refpos_out_tl 1492, 1644, 2108, 2122, 2179, 2238, 2253	\l_zrefclever_type_name_tl
\l_zrefclever_refpre_in_tl 1492, 1646, 2102, 2117, 2162, 2231, 2247	59, 62, 1483, 1856, 1862, 2154, 2185, 2192, 2200, 2264, 2267, 2305, 2311, 2313, 2326, 2334, 2340, 2346, 2348, 2365
\l_zrefclever_refpre_out_tl 1492, 1642, 2094, 2114, 2159, 2216, 2244	\l_zrefclever_typeset_compress_- bool
_zrefclever_ride_on_label:n . 2615	564, 567, 1928
\l_zrefclever_setup_type_tl	\l_zrefclever_typeset_labels_- seq
14, 189, 299, 337, 350, 351, 362, 379, 393, 955, 983, 991, 1004, 1028, 1029, 1040, 1061, 1070, 1084, 1092	44, 1472, 1509, 1533, 1535, 1541
\l_zrefclever_sort_decided_bool	\l_zrefclever_typeset_last_bool
1149, 1291, 1305, 1315, 1319, 1331, 1341, 1356, 1371, 1398	45, 1472, 1530, 1531, 1538, 1563, 1887, 2380
_zrefclever_sort_default:nn	\l_zrefclever_typeset_name_bool
38, 1190, 1206	515, 522, 527, 532, 1823, 1837
_zrefclever_sort_default_- different_types:nn	\l_zrefclever_typeset_queue_- curr_tl
20, 37, 42, 1244, 1411	45, 58, 62, 1477, 1512, 1685, 1701, 1710, 1741, 1752, 1768, 1790, 1808, 1825, 1832, 1839, 1872, 1894, 1899, 1905, 1911, 1912, 1989, 2000, 2031, 2043, 2057, 2280, 2375, 2379
_zrefclever_sort_default_same_- type:nn	\l_zrefclever_typeset_queue_- prev_tl
37, 40, 1242, 1278	45, 1477, 1511, 1883, 1910
_zrefclever_sort_labels:	\l_zrefclever_typeset_range_- bool
37, 38, 44, 1114, 1153	573, 576, 1113, 1788
_zrefclever_sort_page:nn	\l_zrefclever_typeset_ref_bool
44, 1189, 1463	514, 521, 526, 531, 1823, 1830
\l_zrefclever_sort_prior_a_int	_zrefclever_typeset_refs:
1150, 1413, 1419, 1420, 1426, 1436, 1444	44-46, 1117, 1507

_zrefclever_typeset_refs_last_- of_type: . 50, 58, 59, 62, 1660, 1665	_zrefclever_zceref:nnnn 35, 37, 1100
_zrefclever_typeset_refs_not_- last_of_type: 45, 50, 58, 65, 1662, 1922	\l_zrefclever_zceref_labels_seq . 37, 38, 1104, 1131, 1135, 1158, 1161, 1510
\l_zrefclever_typeset_sort_bool 540, 543, 1112	\l_zrefclever_zceref_note_tl ... 798, 801, 1119, 1123
\l_zrefclever_typesort_seq 20, 43, 549, 554, 555, 561, 1415	\l_zrefclever_zceref_with_check_- bool 805, 820, 1109, 1127
\l_zrefclever_use_hyperref_bool 614, 621, 626, 631, 641, 647, 2088, 2210, 2363	_zrefclever_zcsetup:n . 30, 947, 948, 2630, 2635, 2658, 2663, 2670, 2690, 2780, 2785, 2805, 2818, 2835
\l_zrefclever_warn_hyperref_- bool 615, 622, 627, 632, 645	\l_zrefclever_zrefcheck_- available_bool 804, 815, 827, 1108, 1126
_zrefclever_zceref:nnn .. 1099, 1100	