

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	70
9.1	\footnote	70
9.2	\appendix	71
9.3	appendix package	72
9.4	amsmath package	73
9.5	mathtools package	75
9.6	listings package	76
9.7	enumitem package	77

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

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

10	Dictionaries	78
10.1	English	78
10.2	German	82
10.3	French	85
10.4	Portuguese	88
10.5	Spanish	92
Index		96

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, \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 \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 __zrefclever_get_ref_string:nN, __zrefclever_get_ref_font:nN, and __zrefclever_type_name_setup: which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in \g__zrefclever_fallback_dict_prop.

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

```

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

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

Provide

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

`\g__zrefclever_loaded_dictionaries_seq`

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` 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     }

```

Integration with mathtools.

```

1133     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1134     {
1135         \__zrefclever_mathtools_showonlyrefs:n
1136         { \l__zrefclever_zcref_labels_seq }
1137     }
1138     \group_end:
1139 }

```

(End definition for `__zrefclever_zcref:nnnn`.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1140 \seq_new:N \l__zrefclever_zcref_labels_seq
1141 \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>}

```

```

1142 \NewDocumentCommand \zcpageref { s O { } m }
1143 {
1144     \IfBooleanTF {#1}
1145     { \zcref*[#2, ref = page] {#3} }
1146     { \zcref [ #2, ref = page] {#3} }
1147 }

```

(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.

$\backslash l_zrefclever_label_type_a_tl$ $\backslash l_zrefclever_label_type_b_tl$ $\backslash l_zrefclever_label_enclval_a_tl$ $\backslash l_zrefclever_label_enclval_b_tl$ $\backslash l_zrefclever_label_extdoc_a_tl$ $\backslash l_zrefclever_label_extdoc_b_tl$	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p> <pre> 1148 \tl_new:N \l__zrefclever_label_type_a_tl 1149 \tl_new:N \l__zrefclever_label_type_b_tl 1150 \tl_new:N \l__zrefclever_label_enclval_a_tl 1151 \tl_new:N \l__zrefclever_label_enclval_b_tl 1152 \tl_new:N \l__zrefclever_label_extdoc_a_tl 1153 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre> <p><i>(End definition for $\backslash l_zrefclever_label_type_a_tl$ and others.)</i></p>
$\backslash l_zrefclever_sort_decided_bool$	<p>Auxiliary variable for $\backslash _zrefclever_sort_default_same_type:nn$, signals if the sorting between two labels has been decided or not.</p> <pre> 1154 \bool_new:N \l__zrefclever_sort_decided_bool </pre> <p><i>(End definition for $\backslash l_zrefclever_sort_decided_bool$.)</i></p>
$\backslash l_zrefclever_sort_prior_a_int$ $\backslash l_zrefclever_sort_prior_b_int$	<p>Auxiliary variables for $\backslash _zrefclever_sort_default_different_types:nn$. Store the sort priority of the “current” and “next” labels.</p> <pre> 1155 \int_new:N \l__zrefclever_sort_prior_a_int 1156 \int_new:N \l__zrefclever_sort_prior_b_int </pre> <p><i>(End definition for $\backslash l_zrefclever_sort_prior_a_int$ and $\backslash l_zrefclever_sort_prior_b_int$.)</i></p>
$\backslash l_zrefclever_label_types_seq$	<p>Stores the order in which reference types appear in the label list supplied by the user in $\backslash zcref$. This variable is populated by $\backslash _zrefclever_label_type_put_new_right:n$ at the start of $\backslash _zrefclever_sort_labels:$. This order is required as a “last resort” sort criterion between the reference types, for use in $\backslash _zrefclever_sort_default_different_types:nn$.</p> <pre> 1157 \seq_new:N \l__zrefclever_label_types_seq </pre> <p><i>(End definition for $\backslash l_zrefclever_label_types_seq$.)</i></p>
$\backslash _zrefclever_sort_labels:$	<p>The main sorting function. It does not receive arguments, but it is expected to be run inside $\backslash _zrefclever_zcref:nnnn$ where a number of environment variables are to be set appropriately. In particular, $\backslash l_zrefclever_zcref_labels_seq$ should contain the labels received as argument to $\backslash zcref$, and the function performs its task by sorting this variable.</p> <pre> 1158 \cs_new_protected:Npn _zrefclever_sort_labels: 1159 { </pre> <p>Store label types sequence.</p> <pre> 1160 \seq_clear:N \l__zrefclever_label_types_seq 1161 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page } 1162 { 1163 \seq_map_function:NN \l__zrefclever_zcref_labels_seq 1164 _zrefclever_label_type_put_new_right:n 1165 } </pre>

Sort.

```

1166 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1167 {
1168   \zref@ifrefundefined {##1}
1169   {
1170     \zref@ifrefundefined {##2}
1171     {
1172       % Neither label is defined.
1173       \sort_return_same:
1174     }
1175     {
1176       % The second label is defined, but the first isn't, leave the
1177       % undefined first (to be more visible).
1178       \sort_return_same:
1179     }
1180   }
1181   {
1182     \zref@ifrefundefined {##2}
1183     {
1184       % The first label is defined, but the second isn't, bring the
1185       % second forward.
1186       \sort_return_swapped:
1187     }
1188     {
1189       % The interesting case: both labels are defined. References
1190       % to the "default" property or to the "page" are quite
1191       % different with regard to sorting, so we branch them here to
1192       % specialized functions.
1193       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1194       { \__zrefclever_sort_page:nn {##1} {##2} }
1195       { \__zrefclever_sort_default:nn {##1} {##2} }
1196     }
1197   }
1198 }
1199 }

```

(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 \zcreef. 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_zcreef_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>}

1200 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1201 {
1202   \__zrefclever_def_extract_default:Nnnn
1203   \l__zrefclever_label_type_a_tl {#1} { zc@type } { \c_empty_tl }
1204   \seq_if_in:NVF \l__zrefclever_label_types_seq

```

```

1205 \l__zrefclever_label_type_a_tl
1206 {
1207   \seq_put_right:NV \l__zrefclever_label_types_seq
1208   \l__zrefclever_label_type_a_tl
1209 }
1210 }

```

(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}}

1211 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1212 {
1213   \__zrefclever_def_extract_default:Nnnn
1214   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1215   \__zrefclever_def_extract_default:Nnnn
1216   \l__zrefclever_label_type_b_tl {#2} {zc@type} {\c_empty_tl}
1217
1218   \bool_if:nTF
1219   {
1220     % The second label has a type, but the first doesn't, leave the
1221     % undefined first (to be more visible).
1222     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1223     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1224   }
1225   { \sort_return_same: }
1226   {
1227     \bool_if:nTF
1228     {
1229       % The first label has a type, but the second doesn't, bring the
1230       % second forward.
1231       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1232       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1233     }
1234     { \sort_return_swapped: }
1235     {
1236       \bool_if:nTF
1237       {
1238         % The interesting case: both labels have a type...
1239         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1240         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1241       }
1242       {
1243         \tl_if_eq:NNTF
1244         \l__zrefclever_label_type_a_tl
1245         \l__zrefclever_label_type_b_tl
1246         % ...and it's the same type.
1247         { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1248         % ...and they are different types.

```

```

1249         { \_zrefclever_sort_default_different_types:nn {#1} {#2} }
1250     }
1251     {
1252         % Neither label has a type. We can't do much of meaningful
1253         % here, but if it's the same counter, compare it.
1254         \exp_args:Nxx \tl_if_eq:nnTF
1255         {
1256             \_zrefclever_extract_default_unexp:nnn
1257             {#1} { zc@counter } { }
1258         }
1259         {
1260             \_zrefclever_extract_default_unexp:nnn
1261             {#2} { zc@counter } { }
1262         }
1263         {
1264             \int_compare:nNnTF
1265             {
1266                 \_zrefclever_extract_default:nnn
1267                 {#1} { zc@cntval } { -1 }
1268             }
1269             >
1270             {
1271                 \_zrefclever_extract_default:nnn
1272                 {#2} { zc@cntval } { -1 }
1273             }
1274             { \sort_return_swapped: }
1275             { \sort_return_same: }
1276         }
1277         { \sort_return_same: }
1278     }
1279 }
1280 }
1281 }

```

(End definition for _zrefclever_sort_default:nn.)

Variant not provided by the kernel, for use in _zrefclever_sort_default_same_type:nn.

```

1282 \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>}
1283 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1284 {
1285     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
1286     {#1} { zc@enclval } { \c_empty_tl }
1287     \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1288     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
1289     {#2} { zc@enclval } { \c_empty_tl }
1290     \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1291     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
1292     {#1} { externaldocument } { \c_empty_tl }
1293     \_zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
1294     {#2} { externaldocument } { \c_empty_tl }
1295
1296     \bool_set_false:N \l__zrefclever_sort_decided_bool

```



```

1297
1298 % First we check if there's any "external document" difference (coming
1299 % from 'zref-xr') and, if so, sort based on that.
1300 \tl_if_eq:NNF
1301   \l__zrefclever_label_extdoc_a_tl
1302   \l__zrefclever_label_extdoc_b_tl
1303   {
1304     \bool_if:nTF
1305       {
1306         \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1307         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1308       }
1309       {
1310         \bool_set_true:N \l__zrefclever_sort_decided_bool
1311         \sort_return_same:
1312       }
1313       {
1314         \bool_if:nTF
1315           {
1316             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1317             \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1318           }
1319           {
1320             \bool_set_true:N \l__zrefclever_sort_decided_bool
1321             \sort_return_swapped:
1322           }
1323           {
1324             \bool_set_true:N \l__zrefclever_sort_decided_bool
1325             % Two different "external documents": last resort, sort by the
1326             % document name itself.
1327             \str_compare:eNeTF
1328               { \l__zrefclever_label_extdoc_b_tl } <
1329               { \l__zrefclever_label_extdoc_a_tl }
1330               { \sort_return_swapped: }
1331               { \sort_return_same: }
1332           }
1333       }
1334   }
1335
1336 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1337 {
1338   \bool_if:nTF
1339     {
1340       % Both are empty: neither label has any (further) "enclosing
1341       % counters" (left).
1342       \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1343       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1344     }
1345     {
1346       \bool_set_true:N \l__zrefclever_sort_decided_bool
1347       \int_compare:nNnTF
1348         { \__zrefclever_extract_default:nnn {#1} {zc@cntval} { -1 } }
1349         >
1350         { \__zrefclever_extract_default:nnn {#2} {zc@cntval} { -1 } }

```

```

1351     { \sort_return_swapped: }
1352     { \sort_return_same:    }
1353   }
1354   {
1355     \bool_if:nTF
1356     {
1357       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1358       \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1359     }
1360     {
1361       \bool_set_true:N \l__zrefclever_sort_decided_bool
1362       \int_compare:nNnTF
1363       { \__zrefclever_extract_default:nnn {#1} { zc@cntval } { } }
1364       >
1365       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1366       { \sort_return_swapped: }
1367       { \sort_return_same:    }
1368     }
1369   }
1370   \bool_if:nTF
1371   {
1372     % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1373     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1374   }
1375   {
1376     \bool_set_true:N \l__zrefclever_sort_decided_bool
1377     \int_compare:nNnTF
1378     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1379     <
1380     {
1381       \__zrefclever_extract_default:nnn
1382       {#2} { zc@cntval } { }
1383     }
1384     { \sort_return_same:    }
1385     { \sort_return_swapped: }
1386   }
1387   {
1388     % Neither is empty: we can compare the values of the
1389     % current enclosing counter in the loop, if they are
1390     % equal, we are still in the loop, if they are not, a
1391     % sorting decision can be made directly.
1392     \int_compare:nNnTF
1393     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1394     =
1395     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1396     {
1397       \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1398       { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1399       \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1400       { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1401     }
1402     {
1403       \bool_set_true:N \l__zrefclever_sort_decided_bool
1404       \int_compare:nNnTF

```

```

1405         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1406         >
1407         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1408         { \sort_return_swapped: }
1409         { \sort_return_same: }
1410     }
1411 }
1412 }
1413 }
1414 }
1415 }

```

(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>}
1416 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1417 {

```

Retrieve sort priorities for $\langle \text{label } a \rangle$ and $\langle \text{label } b \rangle$. `\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”.

```

1418     \int_zero:N \l__zrefclever_sort_prior_a_int
1419     \int_zero:N \l__zrefclever_sort_prior_b_int
1420     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1421     {
1422         \tl_if_eq:nnTF {##2} {{othertypes}}
1423         {
1424             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1425             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1426             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1427             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1428         }
1429         {
1430             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1431             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1432             {
1433                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1434                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1435             }
1436         }
1437     }

```

Then do the actual sorting.

```

1438     \bool_if:nTF
1439     {
1440         \int_compare_p:nNn
1441         { \l__zrefclever_sort_prior_a_int } <
1442         { \l__zrefclever_sort_prior_b_int }
1443     }
1444     { \sort_return_same: }
1445     {
1446         \bool_if:nTF
1447         {
1448             \int_compare_p:nNn

```

```

1449         { \l__zrefclever_sort_prior_a_int } >
1450         { \l__zrefclever_sort_prior_b_int }
1451     }
1452     { \sort_return_swapped: }
1453     {
1454         % Sort priorities are equal: the type that occurs first in
1455         % 'labels', as given by the user, is kept (or brought) forward.
1456         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1457         {
1458             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1459             { \seq_map_break:n { \sort_return_same: } }
1460             {
1461                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1462                 { \seq_map_break:n { \sort_return_swapped: } }
1463             }
1464         }
1465     }
1466 }
1467 }

```

(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}

1468 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1469 {
1470     \int_compare:nNnTF
1471     { \__zrefclever_extract_default:nnn {#1} { abspage } { -1 } }
1472     >
1473     { \__zrefclever_extract_default:nnn {#2} { abspage } { -1 } }
1474     { \sort_return_swapped: }
1475     { \sort_return_same: }
1476 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

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

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

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

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

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

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this (and good ones at that) see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alter-

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

Variables

<code>\l_zrefclever_typeset_labels_seq</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main stack control.
<code>\l_zrefclever_typeset_last_bool</code>	1477 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l_zrefclever_last_of_type_bool</code>	1478 <code>\bool_new:N \l_zrefclever_typeset_last_bool</code>
	1479 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>
	(End definition for <code>\l_zrefclever_typeset_labels_seq</code> , <code>\l_zrefclever_typeset_last_bool</code> , and <code>\l__zrefclever_last_of_type_bool</code> .)
<code>\l_zrefclever_type_count_int</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main counters.
<code>\l_zrefclever_label_count_int</code>	1480 <code>\int_new:N \l_zrefclever_type_count_int</code>
	1481 <code>\int_new:N \l_zrefclever_label_count_int</code>
	(End definition for <code>\l_zrefclever_type_count_int</code> and <code>\l_zrefclever_label_count_int</code> .)
<code>\l__zrefclever_label_a_tl</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main “queue” control and storage.
<code>\l__zrefclever_label_b_tl</code>	
<code>\l_zrefclever_typeset_queue_prev_tl</code>	1482 <code>\tl_new:N \l_zrefclever_label_a_tl</code>
<code>\l_zrefclever_typeset_queue_curr_tl</code>	1483 <code>\tl_new:N \l_zrefclever_label_b_tl</code>
<code>\l_zrefclever_type_first_label_tl</code>	1484 <code>\tl_new:N \l_zrefclever_typeset_queue_prev_tl</code>
<code>\l__zrefclever_type_first_label_type_tl</code>	1485 <code>\tl_new:N \l_zrefclever_typeset_queue_curr_tl</code>
	1486 <code>\tl_new:N \l_zrefclever_type_first_label_tl</code>
	1487 <code>\tl_new:N \l_zrefclever_type_first_label_type_tl</code>
	(End definition for <code>\l__zrefclever_label_a_tl</code> and others.)
<code>\l__zrefclever_type_name_tl</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : type name handling.
<code>\l_zrefclever_name_in_link_bool</code>	1488 <code>\tl_new:N \l__zrefclever_type_name_tl</code>
<code>\l_zrefclever_name_format_tl</code>	1489 <code>\bool_new:N \l_zrefclever_name_in_link_bool</code>
<code>\l_zrefclever_name_format_fallback_tl</code>	1490 <code>\tl_new:N \l_zrefclever_name_format_tl</code>
	1491 <code>\tl_new:N \l_zrefclever_name_format_fallback_tl</code>
	(End definition for <code>\l__zrefclever_type_name_tl</code> and others.)
<code>\l_zrefclever_range_count_int</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : range handling.
<code>\l_zrefclever_range_same_count_int</code>	1492 <code>\int_new:N \l_zrefclever_range_count_int</code>
<code>\l_zrefclever_range_beg_label_tl</code>	1493 <code>\int_new:N \l_zrefclever_range_same_count_int</code>
<code>\l_zrefclever_next_maybe_range_bool</code>	1494 <code>\tl_new:N \l_zrefclever_range_beg_label_tl</code>
<code>\l_zrefclever_next_is_same_bool</code>	1495 <code>\bool_new:N \l_zrefclever_next_maybe_range_bool</code>
	1496 <code>\bool_new:N \l_zrefclever_next_is_same_bool</code>
	(End definition for <code>\l_zrefclever_range_count_int</code> and others.)

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

```

\l__zrefclever_tpairsep_tl 1497 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl 1498 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl 1499 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl 1500 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl 1501 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl 1502 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl 1503 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_rangeseq_tl 1504 \tl_new:N \l__zrefclever_rangeseq_tl
\l__zrefclever_refpre_out_tl 1505 \tl_new:N \l__zrefclever_refpre_out_tl
\l__zrefclever_refpos_out_tl 1506 \tl_new:N \l__zrefclever_refpos_out_tl
\l__zrefclever_refpre_in_tl 1507 \tl_new:N \l__zrefclever_refpre_in_tl
\l__zrefclever_refpos_in_tl 1508 \tl_new:N \l__zrefclever_refpos_in_tl
\l__zrefclever_namefont_tl 1509 \tl_new:N \l__zrefclever_namefont_tl
\l__zrefclever_reffont_out_tl 1510 \tl_new:N \l__zrefclever_reffont_out_tl
\l__zrefclever_reffont_in_tl 1511 \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`.

```

1512 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1513 {
1514   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1515   \l__zrefclever_zcref_labels_seq
1516   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1517   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1518   \tl_clear:N \l__zrefclever_type_first_label_tl
1519   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1520   \tl_clear:N \l__zrefclever_range_beg_label_tl
1521   \int_zero:N \l__zrefclever_label_count_int
1522   \int_zero:N \l__zrefclever_type_count_int
1523   \int_zero:N \l__zrefclever_range_count_int
1524   \int_zero:N \l__zrefclever_range_same_count_int
1525
1526   % Get type block options (not type-specific).
1527   \__zrefclever_get_ref_string:nN { tpairsep }
1528   \l__zrefclever_tpairsep_tl
1529   \__zrefclever_get_ref_string:nN { tlistsep }
1530   \l__zrefclever_tlistsep_tl
1531   \__zrefclever_get_ref_string:nN { tlastsep }
1532   \l__zrefclever_tlastsep_tl
1533
1534   % Process label stack.
1535   \bool_set_false:N \l__zrefclever_typeset_last_bool
1536   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1537   {
1538     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1539     \l__zrefclever_label_a_tl
1540     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1541     {
1542       \tl_clear:N \l__zrefclever_label_b_tl

```

```

1543         \bool_set_true:N \l__zrefclever_typeset_last_bool
1544     }
1545     {
1546         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1547         \l__zrefclever_label_b_tl
1548     }
1549
1550 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1551 {
1552     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1553     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1554 }
1555 {
1556     \__zrefclever_def_extract_default:NVnn
1557     \l__zrefclever_label_type_a_tl \l__zrefclever_label_a_tl
1558     { zc@type } { \c_empty_tl }
1559     \__zrefclever_def_extract_default:NVnn
1560     \l__zrefclever_label_type_b_tl \l__zrefclever_label_b_tl
1561     { zc@type } { \c_empty_tl }
1562 }
1563
1564 % First, we establish whether the "current label" (i.e. 'a') is the
1565 % last one of its type. This can happen because the "next label"
1566 % (i.e. 'b') is of a different type (or different definition status),
1567 % or because we are at the end of the list.
1568 \bool_if:NTF \l__zrefclever_typeset_last_bool
1569 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1570 {
1571     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1572     {
1573         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1574         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1575         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1576     }
1577     {
1578         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1579         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1580         {
1581             % Neither is undefined, we must check the types.
1582             \bool_if:nTF
1583             {
1584                 % Both empty: same "type".
1585                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1586                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1587             }
1588             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1589             {
1590                 \bool_if:nTF
1591                 {
1592                     % Neither empty: compare types.
1593                     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1594                     &&
1595                     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1596                 }

```



```

1597         {
1598             \tl_if_eq:NNTF
1599                 \l__zrefclever_label_type_a_tl
1600                 \l__zrefclever_label_type_b_tl
1601             {
1602                 \bool_set_false:N
1603                     \l__zrefclever_last_of_type_bool
1604             }
1605             {
1606                 \bool_set_true:N
1607                     \l__zrefclever_last_of_type_bool
1608             }
1609         }
1610         % One empty, the other not: different "types".
1611         {
1612             \bool_set_true:N
1613                 \l__zrefclever_last_of_type_bool
1614         }
1615     }
1616 }
1617 }
1618 }
1619
1620 % Handle warnings in case of reference or type undefined.
1621 \zref@refused { \l__zrefclever_label_a_tl }
1622 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1623 {}
1624 {
1625     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1626     {
1627         \msg_warning:nxx { zref-clever } { missing-type }
1628         { \l__zrefclever_label_a_tl }
1629     }
1630 }
1631
1632 % Get type-specific separators, refpre/pos and font options, once per
1633 % type.
1634 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1635 {
1636     \__zrefclever_get_ref_string:nN { namesep      }
1637     \l__zrefclever_namesep_tl
1638     \__zrefclever_get_ref_string:nN { rangesep     }
1639     \l__zrefclever_rangesep_tl
1640     \__zrefclever_get_ref_string:nN { pairsep      }
1641     \l__zrefclever_pairsep_tl
1642     \__zrefclever_get_ref_string:nN { listsep      }
1643     \l__zrefclever_listsep_tl
1644     \__zrefclever_get_ref_string:nN { lastsep      }
1645     \l__zrefclever_lastsep_tl
1646     \__zrefclever_get_ref_string:nN { refpre       }
1647     \l__zrefclever_refpre_out_tl
1648     \__zrefclever_get_ref_string:nN { refpos       }
1649     \l__zrefclever_refpos_out_tl
1650     \__zrefclever_get_ref_string:nN { refpre-in    }

```

```

1651         \l__zrefclever_refpre_in_tl
1652         \__zrefclever_get_ref_string:nN { refpos-in }
1653         \l__zrefclever_refpos_in_tl
1654         \__zrefclever_get_ref_font:nN { namefont }
1655         \l__zrefclever_namefont_tl
1656         \__zrefclever_get_ref_font:nN { reffont }
1657         \l__zrefclever_reffont_out_tl
1658         \__zrefclever_get_ref_font:nN { reffont-in }
1659         \l__zrefclever_reffont_in_tl
1660     }
1661
1662     % Here we send this to a couple of auxiliary functions.
1663     \bool_if:NTF \l__zrefclever_last_of_type_bool
1664     { % There exists no next label of the same type as the current.
1665       { \__zrefclever_typeset_refs_last_of_type: }
1666       % There exists a next label of the same type as the current.
1667       { \__zrefclever_typeset_refs_not_last_of_type: }
1668     }
1669 }

```

(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.

```

1670 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1671 {
1672   % Process the current label to the current queue.
1673   \int_case:nnF { \l__zrefclever_label_count_int }
1674   {
1675     % It is the last label of its type, but also the first one, and that's
1676     % what matters here: just store it.
1677     { 0 }
1678     {
1679       \tl_set:NV \l__zrefclever_type_first_label_tl
1680       \l__zrefclever_label_a_tl
1681       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1682       \l__zrefclever_label_type_a_tl
1683     }
1684
1685     % The last is the second: we have a pair (if not repeated).
1686     { 1 }
1687     {
1688       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1689       {
1690         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1691         {

```

```

1692         \exp_not:V \l__zrefclever_pairsep_tl
1693         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1694     }
1695 }
1696 }
1697 }
1698 % Last is third or more of its type: without repetition, we'd have the
1699 % last element on a list, but control for possible repetition.
1700 {
1701     \int_case:nnF { \l__zrefclever_range_count_int }
1702     {
1703         % There was no range going on.
1704         { 0 }
1705         {
1706             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1707             {
1708                 \exp_not:V \l__zrefclever_lastsep_tl
1709                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1710             }
1711         }
1712         % Last in the range is also the second in it.
1713         { 1 }
1714         {
1715             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1716             {
1717                 % We know 'range_beg_label' is not empty, since this is the
1718                 % second element in the range, but the third or more in the
1719                 % type list.
1720                 \exp_not:V \l__zrefclever_listsep_tl
1721                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1722                 \int_compare:nNnF
1723                     { \l__zrefclever_range_same_count_int } = { 1 }
1724                 {
1725                     \exp_not:V \l__zrefclever_lastsep_tl
1726                     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1727                 }
1728             }
1729         }
1730     }
1731     % Last in the range is third or more in it.
1732     {
1733         \int_case:nnF
1734         {
1735             \l__zrefclever_range_count_int -
1736             \l__zrefclever_range_same_count_int
1737         }
1738         {
1739             % Repetition, not a range.
1740             { 0 }
1741             {
1742                 % If 'range_beg_label' is empty, it means it was also the
1743                 % first of the type, and hence was already handled.
1744                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1745                 {

```

```

1746         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1747         {
1748             \exp_not:V \l__zrefclever_lastsep_tl
1749             \__zrefclever_get_ref:V
1750             \l__zrefclever_range_beg_label_tl
1751         }
1752     }
1753 }
1754 % A 'range', but with no skipped value, treat as list.
1755 { 1 }
1756 {
1757     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1758     {
1759         % Ditto.
1760         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1761         {
1762             \exp_not:V \l__zrefclever_listsep_tl
1763             \__zrefclever_get_ref:V
1764             \l__zrefclever_range_beg_label_tl
1765         }
1766         \exp_not:V \l__zrefclever_lastsep_tl
1767         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1768     }
1769 }
1770 }
1771 {
1772     % An actual range.
1773     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1774     {
1775         % Ditto.
1776         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1777         {
1778             \exp_not:V \l__zrefclever_lastsep_tl
1779             \__zrefclever_get_ref:V
1780             \l__zrefclever_range_beg_label_tl
1781         }
1782         \exp_not:V \l__zrefclever_rangesep_tl
1783         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1784     }
1785 }
1786 }
1787 }
1788
1789 % Handle "range" option. The idea is simple: if the queue is not empty,
1790 % we replace it with the end of the range (or pair). We can still
1791 % retrieve the end of the range from 'label_a' since we know to be
1792 % processing the last label of its type at this point.
1793 \bool_if:NT \l__zrefclever_typeset_range_bool
1794 {
1795     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1796     {
1797         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1798         { }
1799         {

```

```

1800         \msg_warning:nxx { zref-clever } { single-element-range }
1801         { \l__zrefclever_type_first_label_type_tl }
1802     }
1803 }
1804 {
1805     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1806     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1807     { }
1808     {
1809         \__zrefclever_labels_in_sequence:nn
1810         { \l__zrefclever_type_first_label_tl }
1811         { \l__zrefclever_label_a_tl }
1812     }
1813     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1814     {
1815         \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1816         { \exp_not:V \l__zrefclever_pairsep_tl }
1817         { \exp_not:V \l__zrefclever_rangeseq_tl }
1818         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1819     }
1820 }
1821 }
1822
1823 % Now that the type block is finished, we can add the name and the first
1824 % ref to the queue. Also, if "typeset" option is not "both", handle it
1825 % here as well.
1826 \__zrefclever_type_name_setup:
1827 \bool_if:NTF
1828 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1829 {
1830     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1831     { \__zrefclever_get_ref_first: }
1832 }
1833 {
1834     \bool_if:NTF
1835     { \l__zrefclever_typeset_ref_bool }
1836     {
1837         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1838         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1839     }
1840     {
1841         \bool_if:NTF
1842         { \l__zrefclever_typeset_name_bool }
1843         {
1844             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1845             {
1846                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1847                 {
1848                     \exp_not:N \group_begin:
1849                     \exp_not:V \l__zrefclever_namefont_tl
1850                     % It's two '@s', but escaped for DocStrip.
1851                     \exp_not:N \hyper@@link
1852                     {
1853                         \__zrefclever_extract_url_unexp:V

```

```

1854         \l__zrefclever_type_first_label_tl
1855     }
1856     {
1857         \__zrefclever_extract_default_unexp:Vnn
1858         \l__zrefclever_type_first_label_tl
1859         { anchor } { }
1860     }
1861     { \exp_not:V \l__zrefclever_type_name_tl }
1862     \exp_not:N \group_end:
1863 }
1864 {
1865     \exp_not:N \group_begin:
1866     \exp_not:V \l__zrefclever_namefont_tl
1867     \exp_not:V \l__zrefclever_type_name_tl
1868     \exp_not:N \group_end:
1869 }
1870 }
1871 }
1872 {
1873     % Logically, this case would correspond to "typeset=none", but
1874     % it should not occur, given that the options are set up to
1875     % typeset either "ref" or "name". Still, leave here a
1876     % sensible fallback, equal to the behavior of "both".
1877     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1878     { \__zrefclever_get_ref_first: }
1879 }
1880 }
1881 }
1882
1883 % Typeset the previous type, if there is one.
1884 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1885 {
1886     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1887     { \l__zrefclever_tlistsep_tl }
1888     \l__zrefclever_typeset_queue_prev_tl
1889 }
1890
1891 % Wrap up loop, or prepare for next iteration.
1892 \bool_if:NTF \l__zrefclever_typeset_last_bool
1893 {
1894     % We are finishing, typeset the current queue.
1895     \int_case:nnF { \l__zrefclever_type_count_int }
1896     {
1897         % Single type.
1898         { 0 }
1899         { \l__zrefclever_typeset_queue_curr_tl }
1900         % Pair of types.
1901         { 1 }
1902         {
1903             \l__zrefclever_tpairsep_tl
1904             \l__zrefclever_typeset_queue_curr_tl
1905         }
1906     }
1907     {

```

```

1908         % Last in list of types.
1909         \l__zrefclever_tlastsep_tl
1910         \l__zrefclever_typeset_queue_curr_tl
1911     }
1912 }
1913 {
1914     % There are further labels, set variables for next iteration.
1915     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1916     \l__zrefclever_typeset_queue_curr_tl
1917     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1918     \tl_clear:N \l__zrefclever_type_first_label_tl
1919     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1920     \tl_clear:N \l__zrefclever_range_beg_label_tl
1921     \int_zero:N \l__zrefclever_label_count_int
1922     \int_incr:N \l__zrefclever_type_count_int
1923     \int_zero:N \l__zrefclever_range_count_int
1924     \int_zero:N \l__zrefclever_range_same_count_int
1925 }
1926 }

```

(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.

```

1927 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
1928 {
1929     % Signal if next label may form a range with the current one (only
1930     % considered if compression is enabled in the first place).
1931     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1932     \bool_set_false:N \l__zrefclever_next_is_same_bool
1933     \bool_if:NT \l__zrefclever_typeset_compress_bool
1934     {
1935         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1936         { }
1937         {
1938             \_zrefclever_labels_in_sequence:nn
1939             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1940         }
1941     }
1942
1943     % Process the current label to the current queue.
1944     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1945     {
1946         % Current label is the first of its type (also not the last, but it
1947         % doesn't matter here): just store the label.
1948         \tl_set:NV \l__zrefclever_type_first_label_tl
1949         \l__zrefclever_label_a_tl
1950         \tl_set:NV \l__zrefclever_type_first_label_type_tl
1951         \l__zrefclever_label_type_a_tl
1952
1953         % If the next label may be part of a range, we set 'range_beg_label'
1954         % to "empty" (we deal with it as the "first", and must do it there, to
1955         % handle hyperlinking), but also step the range counters.
1956         \bool_if:NT \l__zrefclever_next_maybe_range_bool
1957         {

```

```

1958      \tl_clear:N \l__zrefclever_range_beg_label_tl
1959      \int_incr:N \l__zrefclever_range_count_int
1960      \bool_if:NT \l__zrefclever_next_is_same_bool
1961      { \int_incr:N \l__zrefclever_range_same_count_int }
1962    }
1963  }
1964  {
1965    % Current label is neither the first (nor the last) of its type.
1966    \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1967    {
1968      % Starting, or continuing a range.
1969      \int_compare:nNnTF
1970      { \l__zrefclever_range_count_int } = { 0 }
1971      {
1972        % There was no range going, we are starting one.
1973        \tl_set:NV \l__zrefclever_range_beg_label_tl
1974        \l__zrefclever_label_a_tl
1975        \int_incr:N \l__zrefclever_range_count_int
1976        \bool_if:NT \l__zrefclever_next_is_same_bool
1977        { \int_incr:N \l__zrefclever_range_same_count_int }
1978      }
1979      {
1980        % Second or more in the range, but not the last.
1981        \int_incr:N \l__zrefclever_range_count_int
1982        \bool_if:NT \l__zrefclever_next_is_same_bool
1983        { \int_incr:N \l__zrefclever_range_same_count_int }
1984      }
1985    }
1986  }
1987  % Next element is not in sequence: there was no range, or we are
1988  % closing one.
1989  \int_case:nnF { \l__zrefclever_range_count_int }
1990  {
1991    % There was no range going on.
1992    { 0 }
1993    {
1994      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1995      {
1996        \exp_not:V \l__zrefclever_listsep_tl
1997        \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1998      }
1999    }
2000    % Last is second in the range: if 'range_same_count' is also
2001    % '1', it's a repetition (drop it), otherwise, it's a "pair
2002    % within a list", treat as list.
2003    { 1 }
2004    {
2005      \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2006      {
2007        \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2008        {
2009          \exp_not:V \l__zrefclever_listsep_tl
2010          \__zrefclever_get_ref:V
2011          \l__zrefclever_range_beg_label_tl

```



```

2012     }
2013     \int_compare:nNnF
2014     { \l__zrefclever_range_same_count_int } = { 1 }
2015     {
2016         \exp_not:V \l__zrefclever_listsep_tl
2017         \__zrefclever_get_ref:V
2018         \l__zrefclever_label_a_tl
2019     }
2020 }
2021 }
2022 }
2023 {
2024     % Last is third or more in the range: if 'range_count' and
2025     % 'range_same_count' are the same, its a repetition (drop it),
2026     % if they differ by '1', its a list, if they differ by more,
2027     % it is a real range.
2028     \int_case:nnF
2029     {
2030         \l__zrefclever_range_count_int -
2031         \l__zrefclever_range_same_count_int
2032     }
2033     {
2034         { 0 }
2035         {
2036             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2037             {
2038                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2039                 {
2040                     \exp_not:V \l__zrefclever_listsep_tl
2041                     \__zrefclever_get_ref:V
2042                     \l__zrefclever_range_beg_label_tl
2043                 }
2044             }
2045         }
2046         { 1 }
2047         {
2048             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2049             {
2050                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2051                 {
2052                     \exp_not:V \l__zrefclever_listsep_tl
2053                     \__zrefclever_get_ref:V
2054                     \l__zrefclever_range_beg_label_tl
2055                 }
2056                 \exp_not:V \l__zrefclever_listsep_tl
2057                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2058             }
2059         }
2060     }
2061 }
2062 {
2063     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2064     {
2065         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2066     }

```

```

2066             \exp_not:V \l__zrefclever_listsep_tl
2067             \__zrefclever_get_ref:V
2068             \l__zrefclever_range_beg_label_tl
2069         }
2070         \exp_not:V \l__zrefclever_rangeseq_tl
2071         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2072     }
2073 }
2074 }
2075 % Reset counters.
2076 \int_zero:N \l__zrefclever_range_count_int
2077 \int_zero:N \l__zrefclever_range_same_count_int
2078 }
2079 }
2080 % Step label counter for next iteration.
2081 \int_incr:N \l__zrefclever_label_count_int
2082 }

```

(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 TeXnical 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.

```

2083 \cs_new_protected:Npn \__zrefclever_ref_default:
2084 { \zref@default }
2085 \cs_new_protected:Npn \__zrefclever_name_default:
2086 { \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}

2087 \cs_new:Npn \__zrefclever_get_ref:n #1
2088 {
2089   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2090   {
2091     \bool_if:nTF
2092     {
2093       \l__zrefclever_use_hyperref_bool &&
2094       ! \l__zrefclever_link_star_bool
2095     }
2096     {
2097       \exp_not:N \group_begin:
2098       \exp_not:V \l__zrefclever_reffont_out_tl
2099       \exp_not:V \l__zrefclever_refpre_out_tl
2100       \exp_not:N \group_begin:
2101       \exp_not:V \l__zrefclever_reffont_in_tl
2102       % It's two '@s', but escaped for DocStrip.
2103       \exp_not:N \hyper@@link
2104       { \__zrefclever_extract_url_unexp:n {#1} }
2105       { \__zrefclever_extract_default_unexp:nnn {#1} { anchor } { } }
2106       {
2107         \exp_not:V \l__zrefclever_refpre_in_tl
2108         \__zrefclever_extract_default_unexp:nvn {#1}
2109         { \l__zrefclever_ref_property_tl } { }
2110         \exp_not:V \l__zrefclever_refpos_in_tl
2111       }
2112       \exp_not:N \group_end:
2113       \exp_not:V \l__zrefclever_refpos_out_tl
2114       \exp_not:N \group_end:
2115     }
2116     {
2117       \exp_not:N \group_begin:
2118       \exp_not:V \l__zrefclever_reffont_out_tl
2119       \exp_not:V \l__zrefclever_refpre_out_tl
2120       \exp_not:N \group_begin:
2121       \exp_not:V \l__zrefclever_reffont_in_tl
2122       \exp_not:V \l__zrefclever_refpre_in_tl
2123       \__zrefclever_extract_default_unexp:nvn {#1}
2124       { \l__zrefclever_ref_property_tl } { }
2125       \exp_not:V \l__zrefclever_refpos_in_tl
2126       \exp_not:N \group_end:
2127       \exp_not:V \l__zrefclever_refpos_out_tl
2128       \exp_not:N \group_end:
2129     }
2130   }
2131   { \__zrefclever_ref_default: }
2132 }
2133 \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.

```

2134 \cs_new:Npn \_zrefclever_get_ref_first:
2135 {
2136   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
2137   { \_zrefclever_ref_default: }
2138   {
2139     \bool_if:NTF \l\_zrefclever_name_in_link_bool
2140     {
2141       \zref@ifrefcontainsprop
2142       { \l\_zrefclever_type_first_label_tl }
2143       { \l\_zrefclever_ref_property_tl }
2144       {
2145         % It's two '@s', but escaped for DocStrip.
2146         \exp_not:N \hyper@@link
2147         {
2148           \_zrefclever_extract_url_unexp:V
2149           \l\_zrefclever_type_first_label_tl
2150         }
2151         {
2152           \_zrefclever_extract_default_unexp:Vnn
2153           \l\_zrefclever_type_first_label_tl
2154           { anchor } { }
2155         }
2156       }
2157       {
2158         \exp_not:N \group_begin:
2159         \exp_not:V \l\_zrefclever_namefont_tl
2160         \exp_not:V \l\_zrefclever_type_name_tl
2161         \exp_not:N \group_end:
2162         \exp_not:V \l\_zrefclever_namesep_tl
2163         \exp_not:N \group_begin:
2164         \exp_not:V \l\_zrefclever_reffont_out_tl
2165         \exp_not:V \l\_zrefclever_refpre_out_tl
2166         \exp_not:N \group_begin:
2167         \exp_not:V \l\_zrefclever_reffont_in_tl
2168         \exp_not:V \l\_zrefclever_refpre_in_tl
2169         \_zrefclever_extract_default_unexp:Vnn
2170         \l\_zrefclever_type_first_label_tl
2171         { \l\_zrefclever_ref_property_tl } { }
2172         \exp_not:V \l\_zrefclever_refpos_in_tl
2173         \exp_not:N \group_end:
2174         % hyperlink makes it's own group, we'd like to close the
2175         % 'refpre-out' group after 'refpos-out', but... we close
2176         % it here, and give the trailing 'refpos-out' its own
2177         % group. This will result that formatting given to
2178         % 'refpre-out' will not reach 'refpos-out', but I see no
2179         % alternative, and this has to be handled specially.

```

```

2179         \exp_not:N \group_end:
2180     }
2181     \exp_not:N \group_begin:
2182     % Ditto: special treatment.
2183     \exp_not:V \l__zrefclever_reffont_out_tl
2184     \exp_not:V \l__zrefclever_refpos_out_tl
2185     \exp_not:N \group_end:
2186 }
2187 {
2188     \exp_not:N \group_begin:
2189     \exp_not:V \l__zrefclever_namefont_tl
2190     \exp_not:V \l__zrefclever_type_name_tl
2191     \exp_not:N \group_end:
2192     \exp_not:V \l__zrefclever_namesep_tl
2193     \__zrefclever_ref_default:
2194 }
2195 }
2196 {
2197     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2198     {
2199         \__zrefclever_name_default:
2200         \exp_not:V \l__zrefclever_namesep_tl
2201     }
2202     {
2203         \exp_not:N \group_begin:
2204         \exp_not:V \l__zrefclever_namefont_tl
2205         \exp_not:V \l__zrefclever_type_name_tl
2206         \exp_not:N \group_end:
2207         \exp_not:V \l__zrefclever_namesep_tl
2208     }
2209     \zref@ifrefcontainsprop
2210     { \l__zrefclever_type_first_label_tl }
2211     { \l__zrefclever_ref_property_tl }
2212     {
2213         \bool_if:nTF
2214         {
2215             \l__zrefclever_use_hyperref_bool &&
2216             ! \l__zrefclever_link_star_bool
2217         }
2218         {
2219             \exp_not:N \group_begin:
2220             \exp_not:V \l__zrefclever_reffont_out_tl
2221             \exp_not:V \l__zrefclever_refpre_out_tl
2222             \exp_not:N \group_begin:
2223             \exp_not:V \l__zrefclever_reffont_in_tl
2224             % It's two '@s', but escaped for DocStrip.
2225             \exp_not:N \hyper@@link
2226             {
2227                 \__zrefclever_extract_url_unexp:V
2228                 \l__zrefclever_type_first_label_tl
2229             }
2230             {
2231                 \__zrefclever_extract_default_unexp:Vnn
2232                 \l__zrefclever_type_first_label_tl

```

```

2233         { anchor } { }
2234     }
2235     {
2236         \exp_not:V \l__zrefclever_refpre_in_tl
2237         \__zrefclever_extract_default_unexp:Vvn
2238         \l__zrefclever_type_first_label_tl
2239         { \l__zrefclever_ref_property_tl } { }
2240         \exp_not:V \l__zrefclever_refpos_in_tl
2241     }
2242     \exp_not:N \group_end:
2243     \exp_not:V \l__zrefclever_refpos_out_tl
2244     \exp_not:N \group_end:
2245 }
2246 {
2247     \exp_not:N \group_begin:
2248     \exp_not:V \l__zrefclever_reffont_out_tl
2249     \exp_not:V \l__zrefclever_refpre_out_tl
2250     \exp_not:N \group_begin:
2251     \exp_not:V \l__zrefclever_reffont_in_tl
2252     \exp_not:V \l__zrefclever_refpre_in_tl
2253     \__zrefclever_extract_default_unexp:Vvn
2254     \l__zrefclever_type_first_label_tl
2255     { \l__zrefclever_ref_property_tl } { }
2256     \exp_not:V \l__zrefclever_refpos_in_tl
2257     \exp_not:N \group_end:
2258     \exp_not:V \l__zrefclever_refpos_out_tl
2259     \exp_not:N \group_end:
2260 }
2261 }
2262 { \__zrefclever_ref_default: }
2263 }
2264 }
2265 }

```

(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.

```

2266 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2267 {
2268     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2269     { \tl_clear:N \l__zrefclever_type_name_tl }
2270     {
2271         \tl_if_empty:nTF \l__zrefclever_type_first_label_type_tl

```

```

2272 { \tl_clear:N \l__zrefclever_type_name_tl }
2273 {
2274   % Determine whether we should use capitalization, abbreviation,
2275   % and plural.
2276   \bool_lazy_or:nnTF
2277     { \l__zrefclever_capitalize_bool }
2278     {
2279       \l__zrefclever_capitalize_first_bool &&
2280       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2281     }
2282     { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2283     { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2284   % If the queue is empty, we have a singular, otherwise, plural.
2285   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2286     { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2287     { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2288   \bool_lazy_and:nnTF
2289     { \l__zrefclever_abbrev_bool }
2290     {
2291       ! \int_compare_p:nNn
2292         { \l__zrefclever_type_count_int } = { 0 } ||
2293       ! \l__zrefclever_noabbrev_first_bool
2294     }
2295     {
2296       \tl_set:NV \l__zrefclever_name_format_fallback_tl
2297         \l__zrefclever_name_format_tl
2298       \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2299     }
2300     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2301
2302   \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2303     {
2304       \prop_get:cVNF
2305       {
2306         l__zrefclever_type_
2307         \l__zrefclever_type_first_label_type_tl _options_prop
2308       }
2309       \l__zrefclever_name_format_tl
2310       \l__zrefclever_type_name_tl
2311       {
2312         \__zrefclever_get_type_transl:xxxNF
2313         { \l__zrefclever_ref_language_tl }
2314         { \l__zrefclever_type_first_label_type_tl }
2315         { \l__zrefclever_name_format_tl }
2316         \l__zrefclever_type_name_tl
2317         {
2318           \tl_clear:N \l__zrefclever_type_name_tl
2319           \msg_warning:nnx { zref-clever } { missing-name }
2320           { \l__zrefclever_type_first_label_type_tl }
2321         }
2322       }
2323     }
2324     {
2325       \prop_get:cVNF

```

```

2326         {
2327             l__zrefclever_type_
2328             \l__zrefclever_type_first_label_type_tl _options_prop
2329         }
2330         \l__zrefclever_name_format_tl
2331         \l__zrefclever_type_name_tl
2332         {
2333             \prop_get:cVNF
2334             {
2335                 l__zrefclever_type_
2336                 \l__zrefclever_type_first_label_type_tl _options_prop
2337             }
2338             \l__zrefclever_name_format_fallback_tl
2339             \l__zrefclever_type_name_tl
2340             {
2341                 \__zrefclever_get_type_transl:xxxNF
2342                 { \l__zrefclever_ref_language_tl }
2343                 { \l__zrefclever_type_first_label_type_tl }
2344                 { \l__zrefclever_name_format_tl }
2345                 \l__zrefclever_type_name_tl
2346                 {
2347                     \__zrefclever_get_type_transl:xxxNF
2348                     { \l__zrefclever_ref_language_tl }
2349                     { \l__zrefclever_type_first_label_type_tl }
2350                     { \l__zrefclever_name_format_fallback_tl }
2351                     \l__zrefclever_type_name_tl
2352                     {
2353                         \tl_clear:N \l__zrefclever_type_name_tl
2354                         \msg_warning:nnx { zref-clever }
2355                         { missing-name }
2356                         { \l__zrefclever_type_first_label_type_tl }
2357                     }
2358                 }
2359             }
2360         }
2361     }
2362 }
2363
2364
2365 % Signal whether the type name is to be included in the hyperlink or not.
2366 \bool_lazy_any:nTF
2367 {
2368     { ! \l__zrefclever_use_hyperref_bool }
2369     { \l__zrefclever_link_star_bool }
2370     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2371     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2372 }
2373 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2374 {
2375     \bool_lazy_any:nTF
2376     {
2377         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2378         {
2379             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&

```



```

2380         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2381     }
2382     {
2383         \str_if_eq_p:Nn \l__zrefclever_nameinlink_str { single } &&
2384         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2385         \l__zrefclever_typeset_last_bool &&
2386         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2387     }
2388 }
2389 { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2390 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2391 }
2392 }

```

(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.

```

2393 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2394 {
2395     \zref@ifpropundefined { urluse }
2396     {
2397         \__zrefclever_extract_default_unexp:nnn
2398         {#1} { url } { \c_empty_tl }
2399     }
2400     {
2401         \zref@ifrefcontainsprop {#1} { urluse }
2402         {
2403             \__zrefclever_extract_default_unexp:nnn
2404             {#1} { urluse } { \c_empty_tl }
2405         }
2406         {
2407             \__zrefclever_extract_default_unexp:nnn
2408             {#1} { url } { \c_empty_tl }
2409         }
2410     }
2411 }
2412 \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}

```

```

2413 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2414 {
2415   \__zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
2416   {#1} { externaldocument } { \c_empty_tl }
2417   \__zrefclever_def_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
2418   {#2} { externaldocument } { \c_empty_tl }
2419
2420   \tl_if_eq:NNT
2421     \l__zrefclever_label_extdoc_a_tl
2422     \l__zrefclever_label_extdoc_b_tl
2423   {
2424     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2425     {
2426       \exp_args:Nxx \tl_if_eq:nnT
2427       {
2428         \__zrefclever_extract_default_unexp:nnn
2429         {#1} { zc@pgfmt } { }
2430       }
2431       {
2432         \__zrefclever_extract_default_unexp:nnn
2433         {#2} { zc@pgfmt } { }
2434       }
2435       {
2436         \int_compare:nNnTF
2437         {
2438           \__zrefclever_extract_default:nnn
2439           {#1} { zc@pgval } { -2 } + 1
2440         }
2441         =
2442         {
2443           \__zrefclever_extract_default:nnn
2444           {#2} { zc@pgval } { -1 }
2445         }
2446         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2447         {
2448           \int_compare:nNnT
2449           {
2450             \__zrefclever_extract_default:nnn
2451             {#1} { zc@pgval } { -1 }
2452           }
2453           =
2454           {
2455             \__zrefclever_extract_default:nnn
2456             {#2} { zc@pgval } { -1 }
2457           }
2458           {
2459             \bool_set_true:N
2460             \l__zrefclever_next_maybe_range_bool
2461             \bool_set_true:N
2462             \l__zrefclever_next_is_same_bool
2463           }
2464         }
2465       }
2466     }

```

```

2467 {
2468   \exp_args:Nxx \tl_if_eq:nnT
2469   {
2470     \__zrefclever_extract_default_unexp:nnn
2471     {#1} { zc@counter } { }
2472   }
2473   {
2474     \__zrefclever_extract_default_unexp:nnn
2475     {#2} { zc@counter } { }
2476   }
2477   {
2478     \exp_args:Nxx \tl_if_eq:nnT
2479     {
2480       \__zrefclever_extract_default_unexp:nnn
2481       {#1} { zc@enclval } { }
2482     }
2483     {
2484       \__zrefclever_extract_default_unexp:nnn
2485       {#2} { zc@enclval } { }
2486     }
2487     {
2488       \int_compare:nNnTF
2489       {
2490         \__zrefclever_extract_default:nnn
2491         {#1} { zc@cntval } { -2 } + 1
2492       }
2493       =
2494       {
2495         \__zrefclever_extract_default:nnn
2496         {#2} { zc@cntval } { -1 }
2497       }
2498       {
2499         \bool_set_true:N
2500         \l__zrefclever_next_maybe_range_bool
2501       }
2502       {
2503         \int_compare:nNnT
2504         {
2505           \__zrefclever_extract_default:nnn
2506           {#1} { zc@cntval } { -1 }
2507         }
2508         =
2509         {
2510           \__zrefclever_extract_default:nnn
2511           {#2} { zc@cntval } { -1 }
2512         }
2513         {
2514           \bool_set_true:N
2515           \l__zrefclever_next_maybe_range_bool
2516           \exp_args:Nxx \tl_if_eq:nnT
2517           {
2518             \__zrefclever_extract_default_unexp:nvn {#1}
2519             { l__zrefclever_ref_property_tl } { }
2520           }

```

```

2521         {
2522             \__zrefclever_extract_default_unexp:nvn {#2}
2523             { l__zrefclever_ref_property_tl } { }
2524         }
2525         {
2526             \bool_set_true:N
2527             \l__zrefclever_next_is_same_bool
2528         }
2529     }
2530 }
2531 }
2532 }
2533 }
2534 }
2535 }

```

(End definition for __zrefclever_labels_in_sequence:nn.)

Finally, a couple of functions for retrieving options values, according to the relevant precedence rules. They both receive an *<option>* as argument, and store the retrieved value in *<tl variable>*. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of \l__zrefclever_label_type_a_tl, as set during the processing of the label stack. This could be easily generalized, of course, but I don't think it is worth it, \l__zrefclever_label_type_a_tl is indeed what we want in all practical cases. The difference between __zrefclever_get_ref_string:nN and __zrefclever_get_ref_font:nN is the kind of option each should be used for. __zrefclever_get_ref_string:nN is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fallback”). __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>}
2536 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2537 {
2538     % First attempt: general options.
2539     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2540     {
2541         % If not found, try type specific options.
2542         \bool_lazy_all:nTF
2543         {
2544             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2545             {
2546                 \prop_if_exist_p:c
2547                 {
2548                     l__zrefclever_type_
2549                     \l__zrefclever_label_type_a_tl _options_prop
2550                 }
2551             }
2552             {
2553                 \prop_if_in_p:cn
2554                 {
2555                     l__zrefclever_type_
2556                     \l__zrefclever_label_type_a_tl _options_prop
2557                 }

```

```

2558         {#1}
2559     }
2560 }
2561 {
2562     \prop_get:cnN
2563     {
2564         l__zrefclever_type_
2565         \l__zrefclever_label_type_a_tl _options_prop
2566     }
2567     {#1} #2
2568 }
2569 {
2570     % If not found, try type specific translations.
2571     \__zrefclever_get_type_transl:xnNF
2572     { \l__zrefclever_ref_language_tl }
2573     { \l__zrefclever_label_type_a_tl }
2574     {#1} #2
2575     {
2576         % If not found, try default translations.
2577         \__zrefclever_get_default_transl:xnNF
2578         { \l__zrefclever_ref_language_tl }
2579         {#1} #2
2580         {
2581             % If not found, try fallback.
2582             \__zrefclever_get_fallback_transl:nNF {#1} #2
2583             {
2584                 \tl_clear:N #2
2585                 \msg_warning:nnn { zref-clever }
2586                 { missing-string } {#1}
2587             }
2588         }
2589     }
2590 }
2591 }
2592 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

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

```

```

2608         {
2609             \prop_get:cnNF
2610             {
2611                 l__zrefclever_type_
2612                 \l__zrefclever_label_type_a_tl _options_prop
2613             }
2614             {#1} #2
2615             { \tl_clear:N #2 }
2616         }
2617         { \tl_clear:N #2 }
2618     }
2619 }

```

(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.

```

2620 \AddToHook { begindocument }
2621 {
2622     \cs_set_eq:NN \__zrefclever_orig_label:n \label
2623 }
2624 \cs_new_nopar:Npn \__zrefclever_ride_on_label:n #1
2625 {
2626     \__zrefclever_orig_label:n {#1}
2627     \zlabel {#1}
2628 }

```

(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.

```

2629 \tl_new:N \l__zrefclever_footnote_type_tl
2630 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
2631 \AddToHook { env / minipage / begin }
2632 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
2633 \AddToHook { cmd / @makefntext / before }
2634 {

```

```

2635 \__zrefclever_zcsetup:x
2636 { currentcounter = \l__zrefclever_footnote_type_tl }
2637 }

```

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.

```

2638 \AddToHook { cmd / appendix / before }
2639 {
2640   \__zrefclever_zcsetup:n
2641   {
2642     countertype =
2643     {
2644       chapter      = appendix ,
2645       section      = appendix ,
2646       subsection   = appendix ,
2647       subsubsection = appendix ,
2648     }
2649   }
2650 }

```

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.

```

2651 \AddToHook { begindocument }
2652 {
2653   \@ifpackageloaded { appendix }
2654   {
2655     \newcounter { zc@appendix }
2656     \newcounter { zc@save@appendix }
2657     \setcounter { zc@appendix } { 0 }
2658     \setcounter { zc@save@appendix } { 0 }
2659     \cs_if_exist:cTF { chapter }
2660     {
2661       \cs_if_exist:cT { section }
2662       {
2663         \__zrefclever_zcsetup:n
2664           { counterresetby = { section = zc@appendix } }
2665       }
2666     }
2667     {
2668       \__zrefclever_zcsetup:n
2669         { counterresetby = { chapter = zc@appendix } }
2670     }
2671   \AddToHook { env / appendices / begin }
2672   {
2673     \stepcounter { zc@save@appendix }
2674     \setcounter { zc@appendix } { \value { zc@save@appendix } }
2675     \__zrefclever_zcsetup:n
2676     {
2677       countertype =
2678       {
2679         chapter      = appendix ,
2680         section      = appendix ,
2681         subsection   = appendix ,
2682         subsubsection = appendix ,
2683       }
2684     }
2685   }
2686   \AddToHook { env / appendices / end }
2687   { \setcounter { zc@appendix } { 0 } }
2688   \AddToHook { cmd / appendix / before }
2689   {
2690     \stepcounter { zc@save@appendix }
2691     \setcounter { zc@appendix } { \value { zc@save@appendix } }
2692   }

```



```

2693 \AddToHook { env / subappendices / begin }
2694 {
2695   \__zrefclever_zcsetup:n
2696   {
2697     countertype =
2698     {
2699       section      = appendix ,
2700       subsection   = appendix ,
2701       subsubsection = appendix ,
2702     } ,
2703   }
2704 }
2705 \msg_info:nnn { zref-clever } { compat-package } { appendix }
2706 }
2707 {}
2708 }

```

9.4 amsmath package

About this, see <https://tex.stackexchange.com/a/402297>.

```

2709 \AddToHook { begindocument }
2710 {
2711   \@ifpackageloaded { amsmath }
2712   {

```

First, we define a function for label setting inside `amsmath` math environments, we want it to set both `\zlabel` and `\label`. We may “get a ride” but not steal the place altogether. This makes for potentially redundant labels, but seems a good compromise. We *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that’s precisely the case inside the `multline` environment (and, damn!, I took a beating of this detail...).

```

2713   \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
2714   {
2715     \__zrefclever_orig_ltxlabel:n {#1}
2716     \zref@wrapper@babel \zref@label {#1}
2717   }

```

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside `amsmath`’s math environments. And, after that, redefine it to be `__zrefclever_ltxlabel:n` instead. We must handle `hyperref` here, which comes very late in the preamble, and which loads `nameref` at `begindocument`, which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. `cleveref` also redefines it, and comes even later, but this procedure 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   }

```

The `subequations` environment uses `parentequation` and `equation` as counters, but only the later is subject to `\refstepcounter`. What happens is: at the start, `equation` is refstepped, it is then stored in `parentequation` and set to ‘0’ and, at the end of the environment it is restored to the value of `parentequation`. So, here, we really must specify manually `currentcounter` and the resetting. Note that, for `subequations`, `\zlabel` works just fine (that is, if given immediately after `\begin{subequations}`, to refer to the parent equation).

```

2754     \AddToHook { env / subequations / begin }
2755     {
2756       \__zrefclever_zcsetup:x
2757       {
2758         counterresetby =
2759         {
2760           parentequation =
2761             \__zrefclever_counter_reset_by:n { equation } ,
2762           equation = parentequation ,
2763         } ,
2764         currentcounter = parentequation ,
2765         countertype = { parentequation = equation } ,
2766       }
2767     }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout. But we still have to set `currentcounter` manually for two reasons. First: `\tag`, which naturally does not change the counter, and just sets `\@currentlabel`. Thus a label to a tag gets `\@currentcounter` from whatever came last, normally the current sectioning command.

And we also include the starred environments here, so that we can get proper data for `\tagged` equations even if the environment is unnumbered. Second, since we had to manually set `currentcounter` to `parentequation` in `subequations`, we also have to manually set it to `equation` in environments which may be used within it. The `xxalignat` environment is not included, because it is “starred” by default (i.e. unnumbered), and does not display or accepts labels or tags anyway. The `-ed` (`gathered`, `aligned`, and `alignedat`) and `cases` environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like `array`, `matrix` and variations. Finally, `split` too can only be used as part of another environment.

```

2768     \clist_map_inline:nn
2769     {
2770         equation ,
2771         equation* ,
2772         align ,
2773         align* ,
2774         alignat ,
2775         alignat* ,
2776         flalign ,
2777         flalign* ,
2778         xalignat ,
2779         xalignat* ,
2780         gather ,
2781         gather* ,
2782         multiline ,
2783         multiline* ,
2784     }
2785     {
2786         \AddToHook { env / #1 / begin }
2787         { \__zrefclever_zcsetup:n { currentcounter = equation } }
2788     }

```

And a last touch of care for `amsmath`’s refinements: make the equation references `\textup`.

```

2789     \zcRefTypeSetup { equation } { reffont = \upshape }
2790     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
2791 }
2792 {}
2793 }

```

9.5 mathtools package

All math environments defined by `mathtools`, extending the `amsmath` set, are meant to be used within enclosing math environments, hence we don’t need to handle them specially, since the numbering and the counting is being done on the side of `amsmath`. This includes the new `cases` and `matrix` variants, and also `multlined`.

Hence, as far as I can tell, the only cross-reference related feature to deal with is the `showonlyrefs` option, whose machinery involves writing an extra internal label to the `.aux` file to track for labels which get actually referred to. This is a little more involved, and implies in doing special handling inside `\zcref`, but the feature is very cool, so it’s worth it.

```

2794 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool

```

```

2795 \AddToHook { begindocument }
2796 {
2797   \@ifpackageloaded { mathtools }
2798   {
2799     \MH_if_boolean:nT { show_only_refs }
2800     {
2801       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
2802       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
2803       {
2804         \seq_map_inline:Nn #1
2805         {
2806           \exp_args:Nx \tl_if_eq:nnTF
2807           {
2808             \__zrefclever_extract_default_unexp:nnn
2809             {##1} { zc@type } { }
2810           }
2811           { equation }
2812           {
2813             \protected@write \@auxout { }
2814             { \string \MT@newlabel {##1} }
2815           }
2816           {
2817             \exp_args:Nx \tl_if_eq:nnT
2818             {
2819               \__zrefclever_extract_default_unexp:nnn
2820               {##1} { zc@type } { }
2821             }
2822             { parentequation }
2823             {
2824               \protected@write \@auxout { }
2825               { \string \MT@newlabel {##1} }
2826             }
2827           }
2828         }
2829       }
2830       \msg_info:nnn { zref-clever } { compat-package } { mathtools }
2831     }
2832   }
2833   {}
2834 }

```

9.6 listings package

```

2835 \AddToHook { begindocument }
2836 {
2837   \@ifpackageloaded { listings }
2838   {
2839     \__zrefclever_zcsetup:n
2840     {
2841       countertype =
2842       {
2843         lstlisting = listing ,
2844         lstnumber = line ,
2845       } ,

```

```

2846         counterresetby = { lstnumber = lstlisting } ,
2847     }
2848     \lst@AddToHook { Init }
2849     {

```

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

```

2850         \tl_if_empty:NF \lst@label
2851         { \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.

```

2852         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
2853     }
2854     \msg_info:nnn { zref-clever } { compat-package } { listings }
2855 }
2856 {}
2857 }

```

9.7 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.

```

2858 \AddToHook { begindocument }
2859 {
2860     \@ifpackageloaded { enumitem }
2861     {
2862         \int_set:Nn \l_tmpa_int { 5 }
2863         \bool_while_do:nn
2864         {
2865             \cs_if_exist_p:c
2866             { c@ enum \int_to_roman:n { \l_tmpa_int } }
2867         }
2868         {
2869             \__zrefclever_zcsetup:x
2870             {
2871                 counterresetby =

```

```

2872         {
2873             enum \int_to_roman:n { \l_tmpa_int } =
2874             enum \int_to_roman:n { \l_tmpa_int - 1 }
2875         } ,
2876         countertype =
2877         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
2878     }
2879     \int_incr:N \l_tmpa_int
2880 }
2881 \int_compare:nNnT { \l_tmpa_int } > { 5 }
2882 { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
2883 }
2884 {}
2885 }
2886 \end{package}

```

10 Dictionaries

10.1 English

```

2887 \zcdDeclareLanguage { english }
2888 \zcdDeclareLanguageAlias { american } { english }
2889 \zcdDeclareLanguageAlias { australian } { english }
2890 \zcdDeclareLanguageAlias { british } { english }
2891 \zcdDeclareLanguageAlias { canadian } { english }
2892 \zcdDeclareLanguageAlias { newzealand } { english }
2893 \zcdDeclareLanguageAlias { UKenglish } { english }
2894 \zcdDeclareLanguageAlias { USenglish } { english }
2895 \dict-english
2896 \namesep = {\nobreakspace} ,
2897 \pairsep = {\and\nobreakspace} ,
2898 \listsep = {\,~} ,
2899 \lastsep = {\and\nobreakspace} ,
2900 \tpairsep = {\and\nobreakspace} ,
2901 \tlistsep = {\,~} ,
2902 \tlastsep = {\,~and\nobreakspace} ,
2903 \notesep = {\,~} ,
2904 \rangesep = {\to\nobreakspace} ,
2905
2906 \type = part ,
2907     \Name-sg = Part ,
2908     \name-sg = part ,
2909     \Name-pl = Parts ,
2910     \name-pl = parts ,
2911
2912 \type = chapter ,
2913     \Name-sg = Chapter ,
2914     \name-sg = chapter ,
2915     \Name-pl = Chapters ,
2916     \name-pl = chapters ,
2917
2918 \type = section ,

```

```

2919     Name-sg = Section ,
2920     name-sg = section ,
2921     Name-pl = Sections ,
2922     name-pl = sections ,
2923
2924 type = paragraph ,
2925     Name-sg = Paragraph ,
2926     name-sg = paragraph ,
2927     Name-pl = Paragraphs ,
2928     name-pl = paragraphs ,
2929     Name-sg-ab = Par. ,
2930     name-sg-ab = par. ,
2931     Name-pl-ab = Par. ,
2932     name-pl-ab = par. ,
2933
2934 type = appendix ,
2935     Name-sg = Appendix ,
2936     name-sg = appendix ,
2937     Name-pl = Appendices ,
2938     name-pl = appendices ,
2939
2940 type = subappendix ,
2941     Name-sg = Appendix ,
2942     name-sg = appendix ,
2943     Name-pl = Appendices ,
2944     name-pl = appendices ,
2945
2946 type = page ,
2947     Name-sg = Page ,
2948     name-sg = page ,
2949     Name-pl = Pages ,
2950     name-pl = pages ,
2951     name-sg-ab = p. ,
2952     name-pl-ab = pp. ,
2953
2954 type = line ,
2955     Name-sg = Line ,
2956     name-sg = line ,
2957     Name-pl = Lines ,
2958     name-pl = lines ,
2959
2960 type = figure ,
2961     Name-sg = Figure ,
2962     name-sg = figure ,
2963     Name-pl = Figures ,
2964     name-pl = figures ,
2965     Name-sg-ab = Fig. ,
2966     name-sg-ab = fig. ,
2967     Name-pl-ab = Figs. ,
2968     name-pl-ab = figs. ,
2969
2970 type = table ,
2971     Name-sg = Table ,
2972     name-sg = table ,

```

```

2973     Name-pl = Tables ,
2974     name-pl = tables ,
2975
2976 type = item ,
2977     Name-sg = Item ,
2978     name-sg = item ,
2979     Name-pl = Items ,
2980     name-pl = items ,
2981
2982 type = footnote ,
2983     Name-sg = Footnote ,
2984     name-sg = footnote ,
2985     Name-pl = Footnotes ,
2986     name-pl = footnotes ,
2987
2988 type = note ,
2989     Name-sg = Note ,
2990     name-sg = note ,
2991     Name-pl = Notes ,
2992     name-pl = notes ,
2993
2994 type = equation ,
2995     Name-sg = Equation ,
2996     name-sg = equation ,
2997     Name-pl = Equations ,
2998     name-pl = equations ,
2999     Name-sg-ab = Eq. ,
3000     name-sg-ab = eq. ,
3001     Name-pl-ab = Eqs. ,
3002     name-pl-ab = eqs. ,
3003     refpre-in = {() ,
3004     refpos-in = {} } ,
3005
3006 type = theorem ,
3007     Name-sg = Theorem ,
3008     name-sg = theorem ,
3009     Name-pl = Theorems ,
3010     name-pl = theorems ,
3011
3012 type = lemma ,
3013     Name-sg = Lemma ,
3014     name-sg = lemma ,
3015     Name-pl = Lemmas ,
3016     name-pl = lemmas ,
3017
3018 type = corollary ,
3019     Name-sg = Corollary ,
3020     name-sg = corollary ,
3021     Name-pl = Corollaries ,
3022     name-pl = corollaries ,
3023
3024 type = proposition ,
3025     Name-sg = Proposition ,
3026     name-sg = proposition ,

```



```

3027     Name-pl = Propositions ,
3028     name-pl = propositions ,
3029
3030 type = definition ,
3031     Name-sg = Definition ,
3032     name-sg = definition ,
3033     Name-pl = Definitions ,
3034     name-pl = definitions ,
3035
3036 type = proof ,
3037     Name-sg = Proof ,
3038     name-sg = proof ,
3039     Name-pl = Proofs ,
3040     name-pl = proofs ,
3041
3042 type = result ,
3043     Name-sg = Result ,
3044     name-sg = result ,
3045     Name-pl = Results ,
3046     name-pl = results ,
3047
3048 type = remark ,
3049     Name-sg = Remark ,
3050     name-sg = remark ,
3051     Name-pl = Remarks ,
3052     name-pl = remarks ,
3053
3054 type = example ,
3055     Name-sg = Example ,
3056     name-sg = example ,
3057     Name-pl = Examples ,
3058     name-pl = examples ,
3059
3060 type = algorithm ,
3061     Name-sg = Algorithm ,
3062     name-sg = algorithm ,
3063     Name-pl = Algorithms ,
3064     name-pl = algorithms ,
3065
3066 type = listing ,
3067     Name-sg = Listing ,
3068     name-sg = listing ,
3069     Name-pl = Listings ,
3070     name-pl = listings ,
3071
3072 type = exercise ,
3073     Name-sg = Exercise ,
3074     name-sg = exercise ,
3075     Name-pl = Exercises ,
3076     name-pl = exercises ,
3077
3078 type = solution ,
3079     Name-sg = Solution ,
3080     name-sg = solution ,

```

```

3081 Name-pl = Solutions ,
3082 name-pl = solutions ,
3083 </dict-english>

```

10.2 German

```

3084 <package>\zcDeclareLanguage { german }
3085 <package>\zcDeclareLanguageAlias { austrian      } { german }
3086 <package>\zcDeclareLanguageAlias { germanb       } { german }
3087 <package>\zcDeclareLanguageAlias { ngerman        } { german }
3088 <package>\zcDeclareLanguageAlias { naustrian      } { german }
3089 <package>\zcDeclareLanguageAlias { nswissgerman   } { german }
3090 <package>\zcDeclareLanguageAlias { swissgerman    } { german }
3091 <*dict-german>

3092 namesep = {\nobreakspace} ,
3093 pairsep  = {\und\nobreakspace} ,
3094 listsep  = {,~} ,
3095 lastsep  = {\und\nobreakspace} ,
3096 tpairsep = {\und\nobreakspace} ,
3097 tlistsep = {,~} ,
3098 tlastsep = {\und\nobreakspace} ,
3099 notesep  = {~} ,
3100 rangesep = {\bis\nobreakspace} ,
3101
3102 type = part ,
3103   Name-sg = Teil ,
3104   name-sg  = Teil ,
3105   Name-pl  = Teile ,
3106   name-pl  = Teile ,
3107
3108 type = chapter ,
3109   Name-sg = Kapitel ,
3110   name-sg  = Kapitel ,
3111   Name-pl  = Kapitel ,
3112   name-pl  = Kapitel ,
3113
3114 type = section ,
3115   Name-sg = Abschnitt ,
3116   name-sg  = Abschnitt ,
3117   Name-pl  = Abschnitte ,
3118   name-pl  = Abschnitte ,
3119
3120 type = paragraph ,
3121   Name-sg = Absatz ,
3122   name-sg  = Absatz ,
3123   Name-pl  = Absätze ,
3124   name-pl  = Absätze ,
3125
3126 type = appendix ,
3127   Name-sg = Anhang ,
3128   name-sg  = Anhang ,
3129   Name-pl  = Anhänge ,
3130   name-pl  = Anhänge ,
3131

```

```

3132 type = subappendix ,
3133     Name-sg = Anhang ,
3134     name-sg = Anhang ,
3135     Name-pl = Anhänge ,
3136     name-pl = Anhänge ,
3137
3138 type = page ,
3139     Name-sg = Seite ,
3140     name-sg = Seite ,
3141     Name-pl = Seiten ,
3142     name-pl = Seiten ,
3143
3144 type = line ,
3145     Name-sg = Zeile ,
3146     name-sg = Zeile ,
3147     Name-pl = Zeilen ,
3148     name-pl = Zeilen ,
3149
3150 type = figure ,
3151     Name-sg = Abbildung ,
3152     name-sg = Abbildung ,
3153     Name-pl = Abbildungen ,
3154     name-pl = Abbildungen ,
3155     Name-sg-ab = Abb. ,
3156     name-sg-ab = Abb. ,
3157     Name-pl-ab = Abb. ,
3158     name-pl-ab = Abb. ,
3159
3160 type = table ,
3161     Name-sg = Tabelle ,
3162     name-sg = Tabelle ,
3163     Name-pl = Tabellen ,
3164     name-pl = Tabellen ,
3165
3166 type = item ,
3167     Name-sg = Punkt ,
3168     name-sg = Punkt ,
3169     Name-pl = Punkte ,
3170     name-pl = Punkte ,
3171
3172 type = footnote ,
3173     Name-sg = Fußnote ,
3174     name-sg = Fußnote ,
3175     Name-pl = Fußnoten ,
3176     name-pl = Fußnoten ,
3177
3178 type = note ,
3179     Name-sg = Anmerkung ,
3180     name-sg = Anmerkung ,
3181     Name-pl = Anmerkungen ,
3182     name-pl = Anmerkungen ,
3183
3184 type = equation ,
3185     Name-sg = Gleichung ,

```

```

3186   name-sg = Gleichung ,
3187   Name-pl = Gleichungen ,
3188   name-pl = Gleichungen ,
3189   refpre-in = {() ,
3190   refpos-in = {} } ,
3191
3192   type = theorem ,
3193   Name-sg = Theorem ,
3194   name-sg = Theorem ,
3195   Name-pl = Theoreme ,
3196   name-pl = Theoreme ,
3197
3198   type = lemma ,
3199   Name-sg = Lemma ,
3200   name-sg = Lemma ,
3201   Name-pl = Lemmata ,
3202   name-pl = Lemmata ,
3203
3204   type = corollary ,
3205   Name-sg = Korollar ,
3206   name-sg = Korollar ,
3207   Name-pl = Korollare ,
3208   name-pl = Korollare ,
3209
3210   type = proposition ,
3211   Name-sg = Satz ,
3212   name-sg = Satz ,
3213   Name-pl = Sätze ,
3214   name-pl = Sätze ,
3215
3216   type = definition ,
3217   Name-sg = Definition ,
3218   name-sg = Definition ,
3219   Name-pl = Definitionen ,
3220   name-pl = Definitionen ,
3221
3222   type = proof ,
3223   Name-sg = Beweis ,
3224   name-sg = Beweis ,
3225   Name-pl = Beweise ,
3226   name-pl = Beweise ,
3227
3228   type = result ,
3229   Name-sg = Ergebnis ,
3230   name-sg = Ergebnis ,
3231   Name-pl = Ergebnisse ,
3232   name-pl = Ergebnisse ,
3233
3234   type = remark ,
3235   Name-sg = Bemerkung ,
3236   name-sg = Bemerkung ,
3237   Name-pl = Bemerkungen ,
3238   name-pl = Bemerkungen ,
3239

```

```

3240 type = example ,
3241   Name-sg = Beispiel ,
3242   name-sg = Beispiel ,
3243   Name-pl = Beispiele ,
3244   name-pl = Beispiele ,
3245
3246 type = algorithm ,
3247   Name-sg = Algorithmus ,
3248   name-sg = Algorithmus ,
3249   Name-pl = Algorithmen ,
3250   name-pl = Algorithmen ,
3251
3252 type = listing ,
3253   Name-sg = Listing ,
3254   name-sg = Listing ,
3255   Name-pl = Listings ,
3256   name-pl = Listings ,
3257
3258 type = exercise ,
3259   Name-sg = Übungsaufgabe ,
3260   name-sg = Übungsaufgabe ,
3261   Name-pl = Übungsaufgaben ,
3262   name-pl = Übungsaufgaben ,
3263
3264 type = solution ,
3265   Name-sg = Lösung ,
3266   name-sg = Lösung ,
3267   Name-pl = Lösungen ,
3268   name-pl = Lösungen ,
3269 </dict-german>

```

10.3 French

```

3270 <package>\zcDeclareLanguage { french }
3271 <package>\zcDeclareLanguageAlias { acadian } { french }
3272 <package>\zcDeclareLanguageAlias { canadien } { french }
3273 <package>\zcDeclareLanguageAlias { francais } { french }
3274 <package>\zcDeclareLanguageAlias { frenchb } { french }
3275 <*dict-french>
3276 namesep = {\nobreakspace} ,
3277 pairsep = {\~et\nobreakspace} ,
3278 listsep = {,~} ,
3279 lastsep = {\~et\nobreakspace} ,
3280 tpairsep = {\~et\nobreakspace} ,
3281 tlistsep = {,~} ,
3282 tlastsep = {\~et\nobreakspace} ,
3283 notesep = {~} ,
3284 rangesep = {\~à\nobreakspace} ,
3285
3286 type = part ,
3287   Name-sg = Partie ,
3288   name-sg = partie ,
3289   Name-pl = Parties ,
3290   name-pl = parties ,

```

```

3291
3292 type = chapter ,
3293     Name-sg = Chapitre ,
3294     name-sg = chapitre ,
3295     Name-pl = Chapitres ,
3296     name-pl = chapitres ,
3297
3298 type = section ,
3299     Name-sg = Section ,
3300     name-sg = section ,
3301     Name-pl = Sections ,
3302     name-pl = sections ,
3303
3304 type = paragraph ,
3305     Name-sg = Paragraphe ,
3306     name-sg = paragraphe ,
3307     Name-pl = Paragraphes ,
3308     name-pl = paragraphes ,
3309
3310 type = appendix ,
3311     Name-sg = Annexe ,
3312     name-sg = annexe ,
3313     Name-pl = Annexes ,
3314     name-pl = annexes ,
3315
3316 type = subappendix ,
3317     Name-sg = Annexe ,
3318     name-sg = annexe ,
3319     Name-pl = Annexes ,
3320     name-pl = annexes ,
3321
3322 type = page ,
3323     Name-sg = Page ,
3324     name-sg = page ,
3325     Name-pl = Pages ,
3326     name-pl = pages ,
3327
3328 type = line ,
3329     Name-sg = Ligne ,
3330     name-sg = ligne ,
3331     Name-pl = Lignes ,
3332     name-pl = lignes ,
3333
3334 type = figure ,
3335     Name-sg = Figure ,
3336     name-sg = figure ,
3337     Name-pl = Figures ,
3338     name-pl = figures ,
3339
3340 type = table ,
3341     Name-sg = Table ,
3342     name-sg = table ,
3343     Name-pl = Tables ,
3344     name-pl = tables ,

```

```

3345
3346 type = item ,
3347     Name-sg = Point ,
3348     name-sg = point ,
3349     Name-pl = Points ,
3350     name-pl = points ,
3351
3352 type = footnote ,
3353     Name-sg = Note ,
3354     name-sg = note ,
3355     Name-pl = Notes ,
3356     name-pl = notes ,
3357
3358 type = note ,
3359     Name-sg = Note ,
3360     name-sg = note ,
3361     Name-pl = Notes ,
3362     name-pl = notes ,
3363
3364 type = equation ,
3365     Name-sg = Équation ,
3366     name-sg = équation ,
3367     Name-pl = Équations ,
3368     name-pl = équations ,
3369     refpre-in = {()} ,
3370     refpos-in = {} ,
3371
3372 type = theorem ,
3373     Name-sg = Théorème ,
3374     name-sg = théorème ,
3375     Name-pl = Théorèmes ,
3376     name-pl = théorèmes ,
3377
3378 type = lemma ,
3379     Name-sg = Lemme ,
3380     name-sg = lemme ,
3381     Name-pl = Lemmes ,
3382     name-pl = lemmes ,
3383
3384 type = corollary ,
3385     Name-sg = Corollaire ,
3386     name-sg = corollaire ,
3387     Name-pl = Corollaires ,
3388     name-pl = corollaires ,
3389
3390 type = proposition ,
3391     Name-sg = Proposition ,
3392     name-sg = proposition ,
3393     Name-pl = Propositions ,
3394     name-pl = propositions ,
3395
3396 type = definition ,
3397     Name-sg = Définition ,
3398     name-sg = définition ,

```

```

3399     Name-pl = Définitions ,
3400     name-pl = définitions ,
3401
3402     type = proof ,
3403     Name-sg = Démonstration ,
3404     name-sg = démonstration ,
3405     Name-pl = Démonstrations ,
3406     name-pl = démonstrations ,
3407
3408     type = result ,
3409     Name-sg = Résultat ,
3410     name-sg = résultat ,
3411     Name-pl = Résultats ,
3412     name-pl = résultats ,
3413
3414     type = remark ,
3415     Name-sg = Remarque ,
3416     name-sg = remarque ,
3417     Name-pl = Remarques ,
3418     name-pl = remarques ,
3419
3420     type = example ,
3421     Name-sg = Exemple ,
3422     name-sg = exemple ,
3423     Name-pl = Exemples ,
3424     name-pl = exemples ,
3425
3426     type = algorithm ,
3427     Name-sg = Algorithme ,
3428     name-sg = algorithme ,
3429     Name-pl = Algorithmes ,
3430     name-pl = algorithmes ,
3431
3432     type = listing ,
3433     Name-sg = Liste ,
3434     name-sg = liste ,
3435     Name-pl = Listes ,
3436     name-pl = listes ,
3437
3438     type = exercise ,
3439     Name-sg = Exercice ,
3440     name-sg = exercice ,
3441     Name-pl = Exercices ,
3442     name-pl = exercices ,
3443
3444     type = solution ,
3445     Name-sg = Solution ,
3446     name-sg = solution ,
3447     Name-pl = Solutions ,
3448     name-pl = solutions ,
3449 </dict-french>

```

10.4 Portuguese


```

3450 <package>\zcDeclareLanguage { portuguese }
3451 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3452 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
3453 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3454 <*dict-portuguese>

3455 namesep = {\nobreakspace} ,
3456 pairsep = {\~e\nobreakspace} ,
3457 listsep = {,~} ,
3458 lastsep = {\~e\nobreakspace} ,
3459 tpairsep = {\~e\nobreakspace} ,
3460 tlistsep = {,~} ,
3461 tlastsep = {\~e\nobreakspace} ,
3462 notesep = {\~} ,
3463 rangesep = {\~a\nobreakspace} ,
3464
3465 type = part ,
3466   Name-sg = Parte ,
3467   name-sg = parte ,
3468   Name-pl = Partes ,
3469   name-pl = partes ,
3470
3471 type = chapter ,
3472   Name-sg = Capítulo ,
3473   name-sg = capítulo ,
3474   Name-pl = Capítulos ,
3475   name-pl = capítulos ,
3476
3477 type = section ,
3478   Name-sg = Seção ,
3479   name-sg = seção ,
3480   Name-pl = Seções ,
3481   name-pl = seções ,
3482
3483 type = paragraph ,
3484   Name-sg = Parágrafo ,
3485   name-sg = parágrafo ,
3486   Name-pl = Parágrafos ,
3487   name-pl = parágrafos ,
3488   Name-sg-ab = Par. ,
3489   name-sg-ab = par. ,
3490   Name-pl-ab = Par. ,
3491   name-pl-ab = par. ,
3492
3493 type = appendix ,
3494   Name-sg = Apêndice ,
3495   name-sg = apêndice ,
3496   Name-pl = Apêndices ,
3497   name-pl = apêndices ,
3498
3499 type = subappendix ,
3500   Name-sg = Apêndice ,
3501   name-sg = apêndice ,
3502   Name-pl = Apêndices ,

```

```

3503     name-pl = apêndices ,
3504
3505 type = page ,
3506     Name-sg = Página ,
3507     name-sg = página ,
3508     Name-pl = Páginas ,
3509     name-pl = páginas ,
3510     name-sg-ab = p. ,
3511     name-pl-ab = pp. ,
3512
3513 type = line ,
3514     Name-sg = Linha ,
3515     name-sg = linha ,
3516     Name-pl = Linhas ,
3517     name-pl = linhas ,
3518
3519 type = figure ,
3520     Name-sg = Figura ,
3521     name-sg = figura ,
3522     Name-pl = Figuras ,
3523     name-pl = figuras ,
3524     Name-sg-ab = Fig. ,
3525     name-sg-ab = fig. ,
3526     Name-pl-ab = Figs. ,
3527     name-pl-ab = figs. ,
3528
3529 type = table ,
3530     Name-sg = Tabela ,
3531     name-sg = tabela ,
3532     Name-pl = Tabelas ,
3533     name-pl = tabelas ,
3534
3535 type = item ,
3536     Name-sg = Item ,
3537     name-sg = item ,
3538     Name-pl = Itens ,
3539     name-pl = itens ,
3540
3541 type = footnote ,
3542     Name-sg = Nota ,
3543     name-sg = nota ,
3544     Name-pl = Notas ,
3545     name-pl = notas ,
3546
3547 type = note ,
3548     Name-sg = Nota ,
3549     name-sg = nota ,
3550     Name-pl = Notas ,
3551     name-pl = notas ,
3552
3553 type = equation ,
3554     Name-sg = Equação ,
3555     name-sg = equação ,
3556     Name-pl = Equações ,

```

```

3557 name-pl = equações ,
3558 Name-sg-ab = Eq. ,
3559 name-sg-ab = eq. ,
3560 Name-pl-ab = Eqs. ,
3561 name-pl-ab = eqs. ,
3562 refpre-in = {()} ,
3563 refpos-in = {} ,
3564
3565 type = theorem ,
3566 Name-sg = Teorema ,
3567 name-sg = teorema ,
3568 Name-pl = Teoremas ,
3569 name-pl = teoremas ,
3570
3571 type = lemma ,
3572 Name-sg = Lema ,
3573 name-sg = lema ,
3574 Name-pl = Lemas ,
3575 name-pl = lemas ,
3576
3577 type = corollary ,
3578 Name-sg = Corolário ,
3579 name-sg = corolário ,
3580 Name-pl = Corolários ,
3581 name-pl = corolários ,
3582
3583 type = proposition ,
3584 Name-sg = Proposição ,
3585 name-sg = proposição ,
3586 Name-pl = Proposições ,
3587 name-pl = proposições ,
3588
3589 type = definition ,
3590 Name-sg = Definição ,
3591 name-sg = definição ,
3592 Name-pl = Definições ,
3593 name-pl = definições ,
3594
3595 type = proof ,
3596 Name-sg = Demonstração ,
3597 name-sg = demonstração ,
3598 Name-pl = Demonstrações ,
3599 name-pl = demonstrações ,
3600
3601 type = result ,
3602 Name-sg = Resultado ,
3603 name-sg = resultado ,
3604 Name-pl = Resultados ,
3605 name-pl = resultados ,
3606
3607 type = remark ,
3608 Name-sg = Observação ,
3609 name-sg = observação ,
3610 Name-pl = Observações ,

```

```

3611     name-pl = observações ,
3612
3613 type = example ,
3614     Name-sg = Exemplo ,
3615     name-sg = exemplo ,
3616     Name-pl = Exemplos ,
3617     name-pl = exemplos ,
3618
3619 type = algorithm ,
3620     Name-sg = Algoritmo ,
3621     name-sg = algoritmo ,
3622     Name-pl = Algoritmos ,
3623     name-pl = algoritmos ,
3624
3625 type = listing ,
3626     Name-sg = Listagem ,
3627     name-sg = listagem ,
3628     Name-pl = Listagens ,
3629     name-pl = listagens ,
3630
3631 type = exercise ,
3632     Name-sg = Exercício ,
3633     name-sg = exercício ,
3634     Name-pl = Exercícios ,
3635     name-pl = exercícios ,
3636
3637 type = solution ,
3638     Name-sg = Solução ,
3639     name-sg = solução ,
3640     Name-pl = Soluções ,
3641     name-pl = soluções ,
3642 </dict-portuguese>

```

10.5 Spanish

```

3643 <package>\zcDeclareLanguage { spanish }
3644 <*dict-spanish>
3645 namesep = {\nobreakspace} ,
3646 pairsep = {\~y\nobreakspace} ,
3647 listsep = {,~} ,
3648 lastsep = {\~y\nobreakspace} ,
3649 tpairsep = {\~y\nobreakspace} ,
3650 tlistsep = {,~} ,
3651 tlastsep = {\~y\nobreakspace} ,
3652 notesep = {\~} ,
3653 rangesep = {\~a\nobreakspace} ,
3654
3655 type = part ,
3656     Name-sg = Parte ,
3657     name-sg = parte ,
3658     Name-pl = Partes ,
3659     name-pl = partes ,
3660
3661 type = chapter ,

```

```

3662     Name-sg = Capítulo ,
3663     name-sg = capítulo ,
3664     Name-pl = Capítulos ,
3665     name-pl = capítulos ,
3666
3667 type = section ,
3668     Name-sg = Sección ,
3669     name-sg = sección ,
3670     Name-pl = Secciones ,
3671     name-pl = secciones ,
3672
3673 type = paragraph ,
3674     Name-sg = Párrafo ,
3675     name-sg = párrafo ,
3676     Name-pl = Párrafos ,
3677     name-pl = párrafos ,
3678
3679 type = appendix ,
3680     Name-sg = Apéndice ,
3681     name-sg = apéndice ,
3682     Name-pl = Apéndices ,
3683     name-pl = apéndices ,
3684
3685 type = subappendix ,
3686     Name-sg = Apéndice ,
3687     name-sg = apéndice ,
3688     Name-pl = Apéndices ,
3689     name-pl = apéndices ,
3690
3691 type = page ,
3692     Name-sg = Página ,
3693     name-sg = página ,
3694     Name-pl = Páginas ,
3695     name-pl = páginas ,
3696
3697 type = line ,
3698     Name-sg = Línea ,
3699     name-sg = línea ,
3700     Name-pl = Líneas ,
3701     name-pl = líneas ,
3702
3703 type = figure ,
3704     Name-sg = Figura ,
3705     name-sg = figura ,
3706     Name-pl = Figuras ,
3707     name-pl = figuras ,
3708
3709 type = table ,
3710     Name-sg = Cuadro ,
3711     name-sg = cuadro ,
3712     Name-pl = Cuadros ,
3713     name-pl = cuadros ,
3714
3715 type = item ,

```

```

3716     Name-sg = Punto ,
3717     name-sg = punto ,
3718     Name-pl = Puntos ,
3719     name-pl = puntos ,
3720
3721 type = footnote ,
3722     Name-sg = Nota ,
3723     name-sg = nota ,
3724     Name-pl = Notas ,
3725     name-pl = notas ,
3726
3727 type = note ,
3728     Name-sg = Nota ,
3729     name-sg = nota ,
3730     Name-pl = Notas ,
3731     name-pl = notas ,
3732
3733 type = equation ,
3734     Name-sg = Ecuación ,
3735     name-sg = ecuación ,
3736     Name-pl = Ecuaciones ,
3737     name-pl = ecuaciones ,
3738     refpre-in = {()} ,
3739     refpos-in = {} ,
3740
3741 type = theorem ,
3742     Name-sg = Teorema ,
3743     name-sg = teorema ,
3744     Name-pl = Teoremas ,
3745     name-pl = teoremas ,
3746
3747 type = lemma ,
3748     Name-sg = Lema ,
3749     name-sg = lema ,
3750     Name-pl = Lemas ,
3751     name-pl = lemas ,
3752
3753 type = corollary ,
3754     Name-sg = Corolario ,
3755     name-sg = corolario ,
3756     Name-pl = Corolarios ,
3757     name-pl = corolarios ,
3758
3759 type = proposition ,
3760     Name-sg = Proposición ,
3761     name-sg = proposición ,
3762     Name-pl = Propositiones ,
3763     name-pl = proposiciones ,
3764
3765 type = definition ,
3766     Name-sg = Definición ,
3767     name-sg = definición ,
3768     Name-pl = Definiciones ,
3769     name-pl = definiciones ,

```

```

3770
3771 type = proof ,
3772     Name-sg = Demostración ,
3773     name-sg = demostración ,
3774     Name-pl = Demostraciones ,
3775     name-pl = demostraciones ,
3776
3777 type = result ,
3778     Name-sg = Resultado ,
3779     name-sg = resultado ,
3780     Name-pl = Resultados ,
3781     name-pl = resultados ,
3782
3783 type = remark ,
3784     Name-sg = Observación ,
3785     name-sg = observación ,
3786     Name-pl = Observaciones ,
3787     name-pl = observaciones ,
3788
3789 type = example ,
3790     Name-sg = Ejemplo ,
3791     name-sg = ejemplo ,
3792     Name-pl = Ejemplos ,
3793     name-pl = ejemplos ,
3794
3795 type = algorithm ,
3796     Name-sg = Algoritmo ,
3797     name-sg = algoritmo ,
3798     Name-pl = Algoritmos ,
3799     name-pl = algoritmos ,
3800
3801 type = listing ,
3802     Name-sg = Listado ,
3803     name-sg = listado ,
3804     Name-pl = Listados ,
3805     name-pl = listados ,
3806
3807 type = exercise ,
3808     Name-sg = Ejercicio ,
3809     name-sg = ejercicio ,
3810     Name-pl = Ejercicios ,
3811     name-pl = ejercicios ,
3812
3813 type = solution ,
3814     Name-sg = Solución ,
3815     name-sg = solución ,
3816     Name-pl = Soluciones ,
3817     name-pl = soluciones ,
3818 </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	
\\	97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145
† internal commands:	
\z_zrefclever_current_counter_tl	4
A	
\AddToHook	85, 478, 493, 637, 673, 698, 736, 738, 790, 811, 2620, 2631, 2633, 2638, 2651, 2671, 2686, 2688, 2693, 2709, 2722, 2742, 2754, 2786, 2795, 2835, 2858
\appendix	71
\appendixname	71
B	
\babelname	683
\babelprovide	12, 23
\begin	74
bool commands:	
\bool_case_true:	2
\bool_if:N _{TF}	308, 319, 641, 645, 1133, 1568, 1663, 1793, 1815, 1846, 1892, 1933, 1956, 1960, 1966, 1976, 1982, 2139
\bool_if:n _{TF}	53, 1218, 1227, 1236, 1304, 1314, 1338, 1355, 1370, 1438, 1446, 1582, 1590, 1827, 1834, 1841, 2091, 2213
\bool_lazy_all:n _{TF}	2542
\bool_lazy_and:nn _{TF}	1107, 1125, 2288, 2599
\bool_lazy_any:n _{TF}	2366, 2375
\bool_lazy_or:nn _{TF}	1111, 2276
\bool_new:N	271, 514, 515, 540, 564, 573, 580, 581, 594, 595, 614, 615, 804, 805, 1141, 1154, 1478, 1479, 1489, 1495, 1496, 2794
\bool_set:Nn	1105
\bool_set_false:N	527, 531, 622, 631, 632, 647, 827, 1296, 1535, 1574, 1588, 1602, 1805, 1931, 1932, 2373, 2390
\bool_set_true:N	328, 521, 522, 526, 532, 621, 626, 627, 815, 820, 1310, 1320, 1324, 1346, 1361, 1376, 1403, 1543, 1569, 1575, 1579, 1606, 1612, 2389, 2446, 2459, 2461, 2499, 2514, 2526, 2801
\bool_until_do:Nn	1336, 1536
\bool_while_do:nn	2863
C	
clist commands:	
\clist_map_inline:nn	870, 2768
\counterwithin	4
cs commands:	
\cs_generate_variant:Nn	50, 179, 185, 324, 332, 950, 1016, 1022, 1282, 2133, 2412
\cs_if_exist:N _{TF}	43, 63, 2659, 2661
\cs_if_exist_p:N	2865
\cs_new:Npn	41, 51, 61, 72, 180, 187, 2087, 2134, 2393
\cs_new_nopar:Npn	2624
\cs_new_protected:Npn	174, 272, 325, 333, 339, 460, 948, 1011, 1017, 1100, 1158, 1200, 1211, 1283, 1416, 1468, 1512, 1670, 1927, 2083, 2085, 2266, 2413, 2536, 2593, 2802
\cs_new_protected:Npx	84
\cs_set_eq:NN	88, 2622, 2724, 2725, 2729, 2730, 2738, 2739, 2744, 2745, 2750, 2751
\cs_set_nopar:Npn	2713
E	
\endinput	12
exp commands:	
\exp_args:N _{Ne}	27, 30
\exp_args:NN _{No}	176
\exp_args:NN _{Nx}	262
\exp_args:N _{No}	176, 182
\exp_args:Nn _V	300
\exp_args:N _{Nx}	335
\exp_args:No	182
\exp_args:N _x	282, 2806, 2817
\exp_args:N _{xx}	1254, 2426, 2468, 2478, 2516
\exp_not:N	58, 1848, 1851, 1862, 1865, 1868, 2097, 2100, 2103, 2112, 2114, 2117, 2120, 2126, 2128, 2146, 2157, 2160, 2162, 2165, 2172, 2179, 2181, 2185, 2188, 2191, 2203, 2206, 2219, 2222, 2225, 2242, 2244, 2247, 2250, 2257, 2259

<code>\exp_not:n</code>	183, 1692, 1708, 1720, 1725, 1748, 1762, 1766, 1778, 1782, 1816, 1817, 1849, 1861, 1866, 1867, 1996, 2009, 2016, 2040, 2052, 2056, 2066, 2070, 2098, 2099, 2101, 2107, 2110, 2113, 2118, 2119, 2121, 2122, 2125, 2127, 2158, 2159, 2161, 2163, 2164, 2166, 2167, 2171, 2183, 2184, 2189, 2190, 2192, 2200, 2204, 2205, 2207, 2220, 2221, 2223, 2236, 2240, 2243, 2248, 2249, 2251, 2252, 2256, 2258
<code>\ExplSyntaxOn</code>	12, 284
F	
file commands:	
<code>\file_get:nnNTF</code>	282
<code>\fmtversion</code>	3
<code>\footnote</code>	70
G	
group commands:	
<code>\group_begin:</code> .. 87, 274, 327, 1000, 1102, 1115, 1848, 1865, 2097, 2100, 2117, 2120, 2157, 2162, 2165, 2181, 2188, 2203, 2219, 2222, 2247, 2250	
<code>\group_end:</code> 90, 322, 330, 1008, 1118, 1138, 1862, 1868, 2112, 2114, 2126, 2128, 2160, 2172, 2179, 2185, 2191, 2206, 2242, 2244, 2257, 2259	
I	
<code>\IfBooleanTF</code>	1144
<code>\IfFormatAtLeastTF</code>	3, 4, 2718
<code>\input</code>	12
int commands:	
<code>\int_case:nnTF</code>	1673, 1701, 1733, 1895, 1989, 2028
<code>\int_compare:nNnTF</code>	1264, 1347, 1362, 1377, 1392, 1404, 1424, 1426, 1470, 1634, 1688, 1722, 1884, 1886, 1944, 1969, 2013, 2436, 2448, 2488, 2503, 2881
<code>\int_compare_p:nNn</code>	1440, 1448, 2280, 2291, 2386
<code>\int_eval:n</code>	84
<code>\int_incr:N</code>	1922, 1959, 1961, 1975, 1977, 1981, 1983, 2081, 2879
<code>\int_new:N</code>	1155, 1156, 1480, 1481, 1492, 1493
<code>\int_set:Nn</code> 1425, 1427, 1431, 1434, 2862	
<code>\int_to_roman:n</code> 2866, 2873, 2874, 2877	
<code>\int_use:N</code>	37, 39, 45
<code>\int_zero:N</code>	1418, 1419, 1521, 1522, 1523, 1524, 1921, 1923, 1924, 2076, 2077
<code>\l_tmpa_int</code>	2862, 2866, 2873, 2874, 2877, 2879, 2881
iow commands:	
<code>\iow_char:N</code>	97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145
K	
keys commands:	
<code>\keys_define:nn</code>	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
<code>\keys_set:nn</code>	12, 31, 35, 301, 821, 949, 956, 1005, 1103
keyval commands:	
<code>\keyval_parse:nnn</code>	841, 896
L	
<code>\label</code>	70, 73, 2622
<code>\labelformat</code>	3
<code>\language</code>	23, 677
M	
<code>\mainbabelname</code>	23, 684
<code>\MessageBreak</code>	10
MH commands:	
<code>\MH_if_boolean:nTF</code>	2799
msg commands:	
<code>\msg_info:nnn</code>	365, 395, 2705, 2790, 2830, 2854, 2882
<code>\msg_line_context:</code> 96, 102, 106, 108, 111, 117, 123, 130, 133, 139, 144, 151, 156, 160, 162, 165, 169	
<code>\msg_new:nnn</code>	94, 100, 105, 107, 109, 115, 121, 127, 129, 131, 137, 142, 147, 149, 154, 159, 161, 163, 168, 170, 172
<code>\msg_note:nnn</code>	304
<code>\msg_warning:nn</code>	483, 508, 646, 652, 795, 832
<code>\msg_warning:nnn</code> ... 251, 266, 310, 320, 724, 769, 898, 965, 1007, 1047, 1086, 1627, 1800, 2319, 2354, 2585	
<code>\msg_warning:nnnn</code>	843
N	
<code>\newcounter</code>	4, 2655, 2656
<code>\NewDocumentCommand</code>	246, 256, 946, 951, 998, 1098, 1142

<code>\nobreakspace</code>	409, 2896, 2897, 2899, 2900, 2902, 2904, 3092, 3093, 3095, 3096, 3098, 3100, 3276, 3277, 3279, 3280, 3282, 3284, 3455, 3456, 3458, 3459, 3461, 3463, 3645, 3646, 3648, 3649, 3651, 3653
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:Npmn</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>	2562
<code>\prop_get:NnNTF</code>	275, 422, 425, 438, 441, 455, 1001, 2304, 2325, 2333, 2539, 2596, 2609
<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>	2546, 2602
<code>\prop_if_in:NnTF</code>	27, 250, 260, 719, 764
<code>\prop_if_in_p:Nn</code>	54, 2553
<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, 74, 77
<code>\renewlist</code>	77
<code>\RequirePackage</code>	16, 17, 18, 19, 642, 787, 808
S	
<code>\scantokens</code>	71
seq commands:	
<code>\seq_clear:N</code>	561, 1160
<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>	1546
<code>\seq_gput_right:Nn</code>	302, 313
<code>\seq_if_empty:NTF</code>	1540
<code>\seq_if_in:NnTF</code>	278, 872, 1204
<code>\seq_map_break:n</code>	75, 1459, 1462
<code>\seq_map_function:NN</code>	1163
<code>\seq_map_indexed_inline:Nn</code>	20, 1420
<code>\seq_map_inline:Nn</code>	354, 371, 385, 923, 958, 970, 1032, 1053, 1076, 1456, 2804
<code>\seq_map_tokens:Nn</code>	57
<code>\seq_new:N</code>	231, 238, 270, 549, 865, 1140, 1157, 1477
<code>\seq_pop_left:NN</code>	1538
<code>\seq_put_right:Nn</code>	874, 1207
<code>\seq_reverse:N</code>	555
<code>\seq_set_eq:NN</code>	1514
<code>\seq_set_from_clist:Nn</code>	554, 1104
<code>\seq_sort:Nn</code>	38, 1166
<code>\setcounter</code>	2657, 2658, 2674, 2687, 2691
sort commands:	
<code>\sort_return_same:</code>	39, 44, 1173, 1178, 1225, 1275, 1277, 1311, 1331, 1352, 1367, 1384, 1409, 1444, 1459, 1475
<code>\sort_return_swapped:</code>	39, 44, 1186, 1234, 1274, 1321, 1330, 1351, 1366, 1385, 1408, 1452, 1462, 1474
<code>\stepcounter</code>	2673, 2690
str commands:	
<code>\str_case:nnTF</code>	700, 745
<code>\str_compare:nNnTF</code>	1327
<code>\str_if_eq:nnTF</code>	74
<code>\str_if_eq_p:nn</code>	2371, 2377, 2379, 2383
<code>\str_new:N</code>	655
<code>\str_set:Nn</code>	660, 662, 664, 666
<code>\string</code>	2814, 2825
T	
<code>\tag</code>	74, 75
TeX and L ^A T _E X 2 _ε commands:	
<code>\@Alph</code>	71
<code>\@addtoreset</code>	4
<code>\@auxout</code>	2813, 2824
<code>\@chapapp</code>	71
<code>\@currentcounter</code>	3, 4, 29, 70, 74, 77, 920
<code>\@currentlabel</code>	3, 74, 77
<code>\@elt</code>	4
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	480, 495, 639, 675, 681, 813, 2653, 2711, 2720, 2734, 2736, 2797, 2837, 2860
<code>\@onlypreamble</code>	255, 269, 1010
<code>\bbl@loaded</code>	23
<code>\bbl@main@language</code>	23, 678

<code>\c@</code>	4	<code>\tl_if_empty:nTF</code>	248,
<code>\c@enumN</code>	77	258, 349, 462, 1027, 1744, 1760,	
<code>\c@lstnumber</code>	77	1776, 2007, 2038, 2050, 2064, 2271	
<code>\c@page</code>	6, 88	<code>\tl_if_empty_p:N</code> .	1222, 1223, 1231,
<code>\cl@</code>	4, 5	1232, 1239, 1240, 1585, 1586, 1593,	
<code>\hyper@@link</code> 58, 1851, 2103, 2146, 2225		1595, 2370, 2380, 2384, 2544, 2600	
<code>\lst@AddToHook</code>	2848	<code>\tl_if_empty_p:n</code>	1306, 1307,
<code>\lst@label</code>	2850, 2851	1316, 1317, 1342, 1343, 1358, 1373	
<code>\ltx@gobble</code>	73	<code>\tl_if_eq:NNTF</code> 1243, 1300, 1598, 2420	
<code>\ltx@label</code> 73, 2724, 2725, 2729, 2730,		<code>\tl_if_eq:NnTF</code>	1161, 1193,
2738, 2739, 2744, 2745, 2750, 2751		1430, 1433, 1458, 1461, 1550, 2424	
<code>\MT@newlabel</code>	2814, 2825	<code>\tl_if_eq:nTF</code>	1254, 1422,
<code>\p@...</code>	3	2426, 2468, 2478, 2516, 2806, 2817	
<code>\protected@write</code>	2813, 2824	<code>\tl_if_novalue:nTF</code>	931, 978
<code>\zref@addprop</code> 21, 24, 35, 38, 40, 82, 93		<code>\tl_map_break:n</code>	75
<code>\zref@default</code>	58, 2084, 2086	<code>\tl_map_tokens:Nn</code>	67
<code>\zref@extractdefault</code>		<code>\tl_new:N</code> ...	83, 189, 190, 466, 670,
8, 9, 65, 177, 183, 188		671, 672, 782, 798, 915, 1148, 1149,	
<code>\zref@ifpropundefined</code>	18, 2395	1150, 1151, 1152, 1153, 1482, 1483,	
<code>\zref@ifrefcontainsprop</code>		1484, 1485, 1486, 1487, 1488, 1490,	
18, 2089, 2141, 2209, 2401		1491, 1494, 1497, 1498, 1499, 1500,	
<code>\zref@ifrefundefined</code>		1501, 1502, 1503, 1504, 1505, 1506,	
1168, 1170, 1182, 1571, 1573, 1578,		1507, 1508, 1509, 1510, 1511, 2629	
1622, 1797, 1806, 1935, 2136, 2268		<code>\tl_put_left:Nn</code>	1830, 1837, 1877
<code>\zref@label</code>	73, 2716	<code>\tl_put_right:Nn</code>	1690, 1706,
<code>\ZREF@mainlist</code> 21, 24, 35, 38, 40, 82, 93		1715, 1746, 1757, 1773, 1994, 2005,	
<code>\zref@newprop</code>		2036, 2048, 2062, 2286, 2287, 2298	
5, 6, 20, 22, 25, 36, 39, 77, 92		<code>\tl_reverse:N</code>	1287, 1290
<code>\zref@refused</code>	1621	<code>\tl_reverse_items:n</code>	1282
<code>\zref@wrapper@babel</code> 35, 73, 1099, 2716		<code>\tl_set:Nn</code>	176, 351, 471,
<code>\textendash</code>	413	473, 475, 481, 484, 500, 509, 677,	
<code>\textup</code>	75	678, 683, 684, 687, 688, 691, 704,	
<code>\the</code>	3	712, 721, 726, 749, 757, 766, 771,	
<code>\thechapter</code>	71	955, 1029, 1397, 1399, 1552, 1553,	
<code>\thelstnumber</code>	77	1679, 1681, 1813, 1844, 1948, 1950,	
<code>\thepage</code>	6, 89	1973, 2282, 2283, 2296, 2630, 2632	
<code>\thesection</code>	71	<code>\tl_set_eq:NN</code>	1915
tl commands:		<code>\tl_tail:N</code>	1398, 1400
<code>\c_empty_tl</code>	1203, 1214,	<code>\l_tmpa_tl</code>	285, 301, 1121, 1122
1216, 1286, 1289, 1292, 1294, 1558,			
1561, 2398, 2404, 2408, 2416, 2418			
<code>\c_novalue_tl</code>	928, 975		
<code>\tl_clear:N</code>			
299, 350, 1004, 1028, 1516,			
1517, 1518, 1519, 1520, 1542, 1917,			
1918, 1919, 1920, 1958, 2269, 2272,			
2300, 2318, 2353, 2584, 2615, 2617			
<code>\tl_gset:Nn</code>	89		
<code>\tl_head:N</code>			
1365, 1378, 1393, 1395, 1405, 1407			
<code>\tl_if_empty:NnTF</code>	65, 362,		
379, 393, 1040, 1061, 1084, 1119,			
1625, 1795, 2197, 2285, 2302, 2850			

U

<code>\upshape</code>	2789
use commands:	
<code>\use:N</code>	23

V

<code>\value</code>	2674, 2691
---------------------------	------------

Z

<code>\zcDeclareLanguage</code>	
11, 246, 2887, 3084, 3270, 3450, 3643	
<code>\zcDeclareLanguageAlias</code>	
11, 256, 2888, 2889,	
2890, 2891, 2892, 2893, 2894, 3085,	

3086, 3087, 3088, 3089, 3090, 3271,
 3272, 3273, 3274, 3451, 3452, 3453
 \zcLanguageSetup . . . 9, 12, 14, 31–33, 998
 \zcpageref 36, 1142
 \zceref 25, 26, 29,
 30, 34–38, 46, 47, 75, 1098, 1145, 1146
 \zcRefTypeSetup 9, 31, 951, 2789
 \zcsetup 23, 26, 29, 30, 946
 \zlabel 70, 73, 74, 77, 2627, 2851
 zrefcheck commands:
 \zrefcheck_zceref_beg_label: . . 1110
 \zrefcheck_zceref_end_label_-
 maybe: 1129
 \zrefcheck_zceref_run_checks_on_-
 labels:n 1130
 zrefclever internal commands:
 \l_zrefclever_abbrev_bool
 594, 598, 2289
 \l_zrefclever_capitalize_bool . .
 580, 584, 2277
 \l_zrefclever_capitalize_first_-
 bool 581, 590, 2279
 __zrefclever_counter_reset_by:n
 5, 28, 43, 45, 47, 51, 2761
 __zrefclever_counter_reset_by_-
 aux:nn 58, 61
 __zrefclever_counter_reset_by_-
 aux:nnn 68, 72
 \l_zrefclever_counter_resetby_-
 prop 5, 28, 54, 55, 891, 903
 \l_zrefclever_counter_resetters_-
 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, 1202, 1213, 1215, 1285, 1288,
 1291, 1293, 1556, 1559, 2415, 2417
 \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, 1266, 1271, 1348, 1350,
 1363, 1381, 1471, 1473, 2438, 2443,
 2450, 2455, 2490, 2495, 2505, 2510
 __zrefclever_extract_default_-
 unexp:nnn 9, 65, 180,
 1256, 1260, 1857, 2105, 2108, 2123,
 2152, 2168, 2231, 2237, 2253, 2397,
 2403, 2407, 2428, 2432, 2470, 2474,
 2480, 2484, 2518, 2522, 2808, 2819
 __zrefclever_extract_url_-
 unexp:n 1853, 2104, 2148, 2227, 2393
 \g_zrefclever_fallback_dict_-
 prop 9, 402, 403, 455
 \l_zrefclever_footnote_type_tl .
 2629, 2630, 2632, 2636
 __zrefclever_get_default_-
 transl:nnN 9, 436, 450
 __zrefclever_get_default_-
 transl:nnNTF 17, 435, 2577
 __zrefclever_get_enclosing_-
 counters_value:n . . . 5, 41, 46, 79
 __zrefclever_get_fallback_-
 transl:nN 453
 __zrefclever_get_fallback_-
 transl:nNTF 17, 451, 2582
 __zrefclever_get_ref:n
 58, 59, 1693, 1709,
 1721, 1726, 1749, 1763, 1767, 1779,
 1783, 1818, 1838, 1997, 2010, 2017,
 2041, 2053, 2057, 2067, 2071, 2087
 __zrefclever_get_ref_first: . . .
 58, 59, 62, 1831, 1878, 2134
 __zrefclever_get_ref_font:nN . 9,
 16, 29, 68, 69, 1654, 1656, 1658, 2593
 __zrefclever_get_ref_string:nN .
 9, 10, 16, 29, 68, 1121, 1527,
 1529, 1531, 1636, 1638, 1640, 1642,
 1644, 1646, 1648, 1650, 1652, 2536
 __zrefclever_get_type_transl:nnnN
 9, 420, 434
 __zrefclever_get_type_transl:nnnNTF
 17, 419, 2312, 2341, 2347, 2571
 \l_zrefclever_label_a_tl
 . 45, 1482, 1539, 1557, 1571, 1621,
 1622, 1628, 1680, 1693, 1709, 1726,
 1767, 1783, 1811, 1818, 1935, 1939,
 1949, 1974, 1997, 2018, 2057, 2071
 \l_zrefclever_label_b_tl
 45, 1482,
 1542, 1547, 1560, 1573, 1578, 1939
 \l_zrefclever_label_count_int . .
 45, 1480,
 1521, 1634, 1673, 1921, 1944, 2081

\l_zrefclever_label_enclval_a_- tl	_zrefclever_name_default:
1148, 1285, 1287, 1342, 1358, 1378, 1393, 1397, 1398, 1405	2083, 2199
\l_zrefclever_label_enclval_b_- tl	\l_zrefclever_name_format_- fallback_tl
1148, 1288, 1290, 1343, 1365, 1373, 1395, 1399, 1400, 1407	. . 1488, 2296, 2300, 2302, 2338, 2350
\l_zrefclever_label_extdoc_a_tl	\l_zrefclever_name_format_tl 1488, 2282, 2283, 2286, 2287, 2297, 2298, 2309, 2315, 2330, 2344
1148, 1291, 1301, 1306, 1316, 1329, 2415, 2421	\l_zrefclever_name_in_link_bool
\l_zrefclever_label_extdoc_b_tl	60, 62, 1488, 1846, 2139, 2373, 2389, 2390
1148, 1293, 1302, 1307, 1317, 1328, 2417, 2422	\l_zrefclever_namefont_tl 1497, 1655, 1849, 1866, 2158, 2189, 2204
\l_zrefclever_label_type_a_tl	\l_zrefclever_nameinlink_str
68, 1148, 1203, 1205, 1208, 1214, 1222, 1231, 1239, 1244, 1430, 1458, 1552, 1557, 1585, 1593, 1599, 1625, 1682, 1951, 2544, 2549, 2556, 2565, 2573, 2600, 2605, 2612	655, 660, 662, 664, 666, 2371, 2377, 2379, 2383
\l_zrefclever_label_type_b_tl	\l_zrefclever_namesep_tl 1497, 1637, 2161, 2192, 2200, 2207
1148, 1216, 1223, 1232, 1240, 1245, 1433, 1461, 1553, 1560, 1586, 1595, 1600	\l_zrefclever_next_is_same_bool
_zrefclever_label_type_put_- new_right:n	45, 65, 1492, 1932, 1960, 1976, 1982, 2462, 2527
37, 38, 1164, 1200	\l_zrefclever_next_maybe_range_- bool
\l_zrefclever_label_types_seq 45, 65, 1492, 1805, 1815, 1931, 1956, 1966, 2446, 2460, 2500, 2515
38, 1157, 1160, 1204, 1207, 1456	\l_zrefclever_noabbrev_first_- bool
_zrefclever_labels_in_sequence:nn	595, 604, 2293
46, 65, 1809, 1938, 2413	_zrefclever_orig_label:n 2622, 2626
\g_zrefclever_languages_prop	_zrefclever_orig_ltxlabel:n 2715, 2724, 2729, 2738, 2744, 2750
11, 245, 250, 252, 260, 263, 264, 275, 422, 438, 719, 764, 1001	_zrefclever_page_format_aux:
\l_zrefclever_last_of_type_bool	84, 88
45, 1477, 1569, 1574, 1575, 1579, 1588, 1603, 1607, 1613, 1663	\g_zrefclever_page_format_tl
\l_zrefclever_lastsep_tl . 1497, 1645, 1708, 1725, 1748, 1766, 1778	6, 83, 89, 92
\l_zrefclever_link_star_bool	\l_zrefclever_pairsep_tl
1105, 1140, 2094, 2216, 2369	1497, 1641, 1692, 1816
\l_zrefclever_listsep_tl 1497, 1643, 1720, 1762, 1996, 2009, 2016, 2040, 2052, 2056, 2066	_zrefclever_prop_put_non_- empty:Nnn
\l_zrefclever_load_dict_- verbose_bool . . .	18, 460, 847, 902
271, 308, 319, 328	_zrefclever_provide_dict_- default_transl:nn 14, 333, 363, 380
\g_zrefclever_loaded_dictionaries_- seq	_zrefclever_provide_dict_type_- transl:nn
270, 279, 302, 313	14, 333, 381, 398
_zrefclever_ltxlabel:n 73, 2713, 2725, 2730, 2739, 2745, 2751	_zrefclever_provide_dictionary:n
\l_zrefclever_main_language_tl	9, 12–15, 35, 272, 329, 740, 751, 759, 774, 1106
23, 671, 678, 684, 688, 692, 705, 727, 750, 772	_zrefclever_provide_dictionary_- verbose:n . . .
_zrefclever_mathtools_showonlyrefs:n	14, 325, 706, 714, 729
1135, 2802	\l_zrefclever_range_beg_label_- tl
\l_zrefclever_mathtools_- showonlyrefs_bool 1133, 2794, 2801	45, 1492, 1520, 1721, 1744, 1750, 1760, 1764, 1776, 1780, 1920, 1958, 1973, 2007, 2011, 2038, 2042, 2050, 2054, 2064, 2068
	\l_zrefclever_range_count_int
	45,

[1492](#), [1523](#), [1701](#), [1735](#), [1923](#), [1959](#),
[1970](#), [1975](#), [1981](#), [1989](#), [2030](#), [2076](#)
\l_zrefclever_range_same_count_
int [45](#),
[1492](#), [1524](#), [1688](#), [1723](#), [1736](#), [1924](#),
[1961](#), [1977](#), [1983](#), [2014](#), [2031](#), [2077](#)
\l_zrefclever_rangeseq_tl
..... [1497](#), [1639](#), [1782](#), [1817](#), [2070](#)
_zrefclever_ref_default:
..... [2083](#), [2131](#), [2137](#), [2193](#), [2262](#)
\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](#), [2313](#), [2342](#), [2348](#), [2572](#), [2578](#)
\c_zrefclever_ref_options_font_
seq [10](#), [16](#), [191](#)
_zrefclever_ref_options_
necessarily_not_type_specific_
seq [16](#), [191](#), [355](#), [959](#), [1033](#)
\c_zrefclever_ref_options_
necessarily_type_specific_seq
..... [191](#), [386](#), [1077](#)
\c_zrefclever_ref_options_
possibly_type_specific_seq ..
..... [16](#), [191](#), [372](#), [1054](#)
\l_zrefclever_ref_options_prop ..
.... [29](#), [31](#), [922](#), [932](#), [933](#), [2539](#), [2596](#)
\c_zrefclever_ref_options_
reference_seq [191](#), [924](#)
\c_zrefclever_ref_options_
typesetup_seq [191](#), [971](#)
\l_zrefclever_ref_property_tl ..
..... [18](#), [466](#), [471](#),
[473](#), [475](#), [481](#), [484](#), [500](#), [509](#), [1161](#),
[1193](#), [1550](#), [2089](#), [2143](#), [2211](#), [2424](#)
\l_zrefclever_ref_typeset_font_
tl [782](#), [784](#), [1116](#)
\l_zrefclever_reffont_in_tl [1497](#),
[1659](#), [2101](#), [2121](#), [2166](#), [2223](#), [2251](#)
\l_zrefclever_reffont_out_tl ...
..... [1497](#), [1657](#),
[2098](#), [2118](#), [2163](#), [2183](#), [2220](#), [2248](#)
\l_zrefclever_refpos_in_tl [1497](#),
[1653](#), [2110](#), [2125](#), [2171](#), [2240](#), [2256](#)
\l_zrefclever_refpos_out_tl [1497](#),
[1649](#), [2113](#), [2127](#), [2184](#), [2243](#), [2258](#)
\l_zrefclever_refpre_in_tl [1497](#),
[1651](#), [2107](#), [2122](#), [2167](#), [2236](#), [2252](#)
\l_zrefclever_refpre_out_tl [1497](#),
[1647](#), [2099](#), [2119](#), [2164](#), [2221](#), [2249](#)
_zrefclever_ride_on_label:n . [2620](#)
\l_zrefclever_setup_type_tl ...
. [14](#), [189](#), [299](#), [337](#), [350](#), [351](#), [362](#),
[379](#), [393](#), [955](#), [983](#), [991](#), [1004](#), [1028](#),
[1029](#), [1040](#), [1061](#), [1070](#), [1084](#), [1092](#)
\l_zrefclever_sort_decided_bool
..... [1154](#), [1296](#), [1310](#), [1320](#),
[1324](#), [1336](#), [1346](#), [1361](#), [1376](#), [1403](#)
_zrefclever_sort_default:nn ...
..... [39](#), [1195](#), [1211](#)
_zrefclever_sort_default_
different_types:nn
..... [20](#), [37](#), [43](#), [1249](#), [1416](#)
_zrefclever_sort_default_same_
type:nn [37](#), [40](#), [1247](#), [1283](#)
_zrefclever_sort_labels:
..... [37-39](#), [44](#), [1114](#), [1158](#)
_zrefclever_sort_page:nn
..... [44](#), [1194](#), [1468](#)
\l_zrefclever_sort_prior_a_int .
..... [1155](#),
[1418](#), [1424](#), [1425](#), [1431](#), [1441](#), [1449](#)
\l_zrefclever_sort_prior_b_int .
..... [1155](#),
[1419](#), [1426](#), [1427](#), [1434](#), [1442](#), [1450](#)
\l_zrefclever_tlastsep_tl
..... [1497](#), [1532](#), [1909](#)
\l_zrefclever_tlistsep_tl
..... [1497](#), [1530](#), [1887](#)
\l_zrefclever_tpairsep_tl
..... [1497](#), [1528](#), [1903](#)
\l_zrefclever_type_<type>_
options_prop [31](#)
\l_zrefclever_type_count_int ...
..... [45](#), [62](#), [1480](#), [1522](#), [1884](#),
[1886](#), [1895](#), [1922](#), [2280](#), [2292](#), [2386](#)
\l_zrefclever_type_first_label_
tl [45](#), [60](#), [1482](#), [1518](#), [1679](#), [1797](#),
[1806](#), [1810](#), [1838](#), [1854](#), [1858](#), [1918](#),
[1948](#), [2136](#), [2142](#), [2149](#), [2153](#), [2169](#),
[2210](#), [2228](#), [2232](#), [2238](#), [2254](#), [2268](#)
\l_zrefclever_type_first_label_
type_tl [45](#), [62](#), [1482](#), [1519](#), [1681](#),
[1801](#), [1919](#), [1950](#), [2271](#), [2307](#), [2314](#),
[2320](#), [2328](#), [2336](#), [2343](#), [2349](#), [2356](#)
_zrefclever_type_name_setup: ..
..... [9](#), [10](#), [60](#), [1826](#), [2266](#)
\l_zrefclever_type_name_tl
..... [60](#), [62](#),
[1488](#), [1861](#), [1867](#), [2159](#), [2190](#), [2197](#),
[2205](#), [2269](#), [2272](#), [2310](#), [2316](#), [2318](#),
[2331](#), [2339](#), [2345](#), [2351](#), [2353](#), [2370](#)
\l_zrefclever_typeset_compress_
bool [564](#), [567](#), [1933](#)
\l_zrefclever_typeset_labels_
seq [45](#), [1477](#), [1514](#), [1538](#), [1540](#), [1546](#)

\l_zrefclever_typeset_last_bool	\l_zrefclever_typeset_sort_bool
..... 45 , 1477 , 540 , 543 , 1112
1535 , 1536 , 1543 , 1568 , 1892 , 2385	\l_zrefclever_typesort_seq
\l_zrefclever_typeset_name_bool 20 , 43 , 549 , 554 , 555 , 561 , 1420
..... 515 , 522 , 527 , 532 , 1828 , 1842	\l_zrefclever_use_hyperref_bool
\l_zrefclever_typeset_queue_- 614 , 621 ,
curr_tl 45 ,	626 , 631 , 641 , 647 , 2093 , 2215 , 2368
58 , 62 , 1482 , 1517 , 1690 , 1706 ,	\l_zrefclever_warn_hyperref_-
1715 , 1746 , 1757 , 1773 , 1795 ,	bool 615 , 622 , 627 , 632 , 645
1813 , 1830 , 1837 , 1844 , 1877 , 1899 ,	_zrefclever_zcref:nnn .. 1099 , 1100
1904 , 1910 , 1916 , 1917 , 1994 , 2005 ,	_zrefclever_zcref:nnnn 35 , 37 , 1100
2036 , 2048 , 2062 , 2285 , 2380 , 2384	\l_zrefclever_zcref_labels_seq .
\l_zrefclever_typeset_queue_- 37 , 38 , 1104 ,
prev_tl . 45 , 1482 , 1516 , 1888 , 1915	1131 , 1136 , 1140 , 1163 , 1166 , 1515
\l_zrefclever_typeset_range_-	\l_zrefclever_zcref_note_tl ...
bool 573 , 576 , 1113 , 1793 798 , 801 , 1119 , 1123
\l_zrefclever_typeset_ref_bool .	\l_zrefclever_zcref_with_check_-
..... 514 , 521 , 526 , 531 , 1828 , 1835	bool 805 , 820 , 1109 , 1127
_zrefclever_typeset_refs:	_zrefclever_zcsetup:n . 30 , 947 ,
..... 45-47 , 1117 , 1512	948 , 2635 , 2640 , 2663 , 2668 , 2675 ,
_zrefclever_typeset_refs_last_-	2695 , 2756 , 2787 , 2839 , 2852 , 2869
of_type: . 50 , 58 , 60 , 62 , 1665 , 1670	\l_zrefclever_zrefcheck_-
_zrefclever_typeset_refs_not_-	available_bool
last_of_type: 804 , 815 , 827 , 1108 , 1126
..... 46 , 50 , 58 , 65 , 1667 , 1927	