

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	68
9.1	\footnote	69
9.2	\appendix	69
9.3	appendix package	70
9.4	amsmath package	71
9.5	mathtools package	74
9.6	listings package	75
9.7	enumitem package	75

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

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

10	Dictionaries	76
10.1	English	76
10.2	German	80
10.3	French	84
10.4	Portuguese	87
10.5	Spanish	91
Index		94

1 Initial setup

Start the DocStrip guards.

```

1 <*package>
   Identify the internal prefix (LATEX3 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: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:Nnnn {\tl val}
  {\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

174 \cs_new_protected:Npn \__zrefclever_def_extract: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:Nnnn { NVnn }

```

(End definition for __zrefclever_def_extract:Nnnn.)

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

```

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

180 \cs_new:Npn \__zrefclever_extract_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 \__zrefclever_extract_unexp:nnn { Vnn , nvnn , Vvn }

```

(End definition for __zrefclever_extract_unexp:nnn.)

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

```

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

186 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
187 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for __zrefclever_extract:nnn.)

4.3 Reference format

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

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

```

188 \tl_new:N \l__zrefclever_setup_type_tl
189 \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

190 \seq_const_from_clist:Nn
191 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
192 {
193   tpairsep ,

```

```

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

```

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:`.

```

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

```

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

```

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

```

```

242 \c__zrefclever_ref_options_reference_seq
243 \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”.

```

244 \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}}
245 \NewDocumentCommand \zcDeclareLanguage { m }
246 {
247   \tl_if_empty:nF {#1}
248   {
249     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
250     { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
251     { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
252   }
253 }
254 \@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}}
255 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
256 {
257   \tl_if_empty:nF {#1}
258   {
259     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
260     {
261       \exp_args:NnNx
262       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
263       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
264     }
265     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
266   }
267 }
268 \@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

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

269 `\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.

270 `\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 {<language>}`

```

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

```

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.

```

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

```

```

341     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
342     { default- #1 } {#2}
343 }

```

(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.

```

344 \keys_define:nn { zref-clever / dictionary }
345 {
346   type .code:n =
347   {
348     \tl_if_empty:NTF {#1}
349     { \tl_clear:N \l__zrefclever_setup_type_tl }
350     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
351   } ,
352 }
353 \seq_map_inline:Nn
354 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
355 {
356   \keys_define:nn { zref-clever / dictionary }
357   {
358     #1 .value_required:n = true ,
359     #1 .code:n =
360     {
361       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
362       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
363       {
364         \msg_info:nnn { zref-clever }
365         { option-not-type-specific } {#1}
366       }
367     } ,
368   }
369 }
370 \seq_map_inline:Nn
371 \c__zrefclever_ref_options_possibly_type_specific_seq
372 {
373   \keys_define:nn { zref-clever / dictionary }
374   {
375     #1 .value_required:n = true ,
376     #1 .code:n =
377     {
378       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
379       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
380       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
381     } ,
382   }
383 }
384 \seq_map_inline:Nn
385 \c__zrefclever_ref_options_necessarily_type_specific_seq
386 {

```

```

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

```

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.

```

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

```

Get translations

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


```

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

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

```

(End definition for _zrefclever_get_type_transl:nnnNF.)

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

```

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

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

```

(End definition for _zrefclever_get_default_transl:nnNF.)

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

```

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

```

```

450 % {<key><tl var to set>
451 \prg_new_protected_conditional:Npnn
452 \__zrefclever_get_fallback_transl:nN #1#2 { F }
453 {
454   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
455     { #1 } #2
456     { \prg_return_true: }
457     { \prg_return_false: }
458 }

```

(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>}

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

```

(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.

```

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

```

```

476 {
477   \AddToHook { begindocument }
478   {
479     \@ifpackageloaded { zref-titleref }
480     { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
481     {
482       \msg_warning:nn { zref-clever } { missing-zref-titleref }
483       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
484     }
485   }
486 },
487 ref .initial:n = default ,
488 ref .default:n = default ,
489 page .meta:n = { ref = page },
490 page .value_forbidden:n = true ,
491 }
492 \AddToHook { begindocument }
493 {
494   \@ifpackageloaded { zref-titleref }
495   {
496     \keys_define:nn { zref-clever / reference }
497     {
498       ref / title .code:n =
499       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
500     }
501   }
502   {
503     \keys_define:nn { zref-clever / reference }
504     {
505       ref / title .code:n =
506       {
507         \msg_warning:nn { zref-clever } { missing-zref-titleref }
508         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
509       }
510     }
511   }
512 }

```

typeset option

```

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

```

```

527     } ,
528     typeset / name .code:n =
529     {
530         \bool_set_false:N \l__zrefclever_typeset_ref_bool
531         \bool_set_true:N \l__zrefclever_typeset_name_bool
532     } ,
533     typeset .initial:n = both ,
534     typeset .value_required:n = true ,
535
536     noname .meta:n = { typeset = ref } ,
537     noname .value_forbidden:n = true ,
538 }

```

sort option

```

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

```

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`.

```

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

```

comp option

```

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

```

```

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

```

range option

```

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

```

cap and capfirst options

```

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

```

abbrev and noabbrevfirst options

```

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

```

S option

```

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

```

hyperref option

```

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

```

nameinlink option

```

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

```

```

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

```

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`.

```

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

```

689 }

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.

```

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

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

735 \AddToHook { begindocument / before }

```



```

736 {
737   \AddToHook { begindocument }
738   {

```

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.

```

739   \__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.

```

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

```

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`.

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

titleref option

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

note option

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

check option

Integration with `zref-check`.

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

```

```

821         } ,
822         check .value_required:n = true ,
823     }
824 }
825 {
826     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
827     \keys_define:nn { zref-clever / reference }
828     {
829         check .value_forbidden:n = false ,
830         check .code:n =
831             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
832     }
833 }
834 }

```

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`.

```

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

```

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.

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

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`.

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

```

900         {
901             \_zrefclever_prop_put_non_empty:Nnn
902             \l__zrefclever_counter_resetby_prop
903         }
904         {#1}
905     } ,
906     counterresetby .value_required:n = true ,
907     counterresetby .initial:n =
908     {

```

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.

```

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

```

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.

```

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

```

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.

```

921 \prop_new:N \l__zrefclever_ref_options_prop
922 \seq_map_inline:Nn
923   \c__zrefclever_ref_options_reference_seq
924   {
925     \keys_define:nn { zref-clever / reference }
926     {

```

```

927     #1 .default:V = \c_novalue_tl ,
928     #1 .code:n =
929     {
930         \tl_if_novalue:nTF {##1}
931         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
932         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
933     } ,
934 }
935 }

```

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.

```

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

```

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

```

944 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{\options}

```

```

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

```

(End definition for `\zcsetup`.)

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

```

\__zrefclever_zcsetup:n{\options}

```

```

947 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
948 { \keys_set:nn { zref-clever / zcsetup } {#1} }
949 \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⟩}
950 \NewDocumentCommand \zcRefTypeSetup { m m }
951 {
952   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
953   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
954   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
955   \keys_set:nn { zref-clever / typesetup } {#2}
956 }
```

(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>.

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

```

975     #1 .code:n =
976     {
977         \tl_if_novalue:nTF {##1}
978         {
979             \prop_remove:cn
980             {
981                 l__zrefclever_type_
982                 \l__zrefclever_setup_type_tl _options_prop
983             }
984             {#1}
985         }
986         {
987             \prop_put:cnn
988             {
989                 l__zrefclever_type_
990                 \l__zrefclever_setup_type_tl _options_prop
991             }
992             {#1} {##1}
993         }
994     } ,
995 }
996 }

```

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

1010 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1011 {
1012   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1013     { type- #2 - #3 } {#4}
1014 }
1015 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
1016 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
1017 {
1018   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
1019     { default- #2 } {#3}
1020 }
1021 \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.

```

1022 \keys_define:nn { zref-clever / langsetup }
1023 {
1024   type .code:n =
1025   {
1026     \tl_if_empty:NTF {#1}
1027       { \tl_clear:N \l__zrefclever_setup_type_tl }
1028       { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1029   } ,
1030 }
1031 \seq_map_inline:Nn
1032   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1033   {
1034     \keys_define:nn { zref-clever / langsetup }
1035     {
1036       #1 .value_required:n = true ,
1037       #1 .code:n =
1038       {
1039         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1040         {
1041           \_zrefclever_declare_default_transl:Vnn
1042             \l__zrefclever_dict_language_tl
1043             {#1} {##1}
1044         }
1045         {
1046           \msg_warning:nnn { zref-clever }
1047             { option-not-type-specific } {#1}
1048         }
1049       } ,
1050     }
1051   }
1052 \seq_map_inline:Nn
1053   \c__zrefclever_ref_options_possibly_type_specific_seq

```

```

1054 {
1055   \keys_define:nn { zref-clever / langsetup }
1056   {
1057     #1 .value_required:n = true ,
1058     #1 .code:n =
1059     {
1060       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1061       {
1062         \__zrefclever_declare_default_transl:Vnn
1063         \l__zrefclever_dict_language_tl
1064         {#1} {##1}
1065       }
1066       {
1067         \__zrefclever_declare_type_transl:Vnn
1068         \l__zrefclever_dict_language_tl
1069         \l__zrefclever_setup_type_tl
1070         {#1} {##1}
1071       }
1072     } ,
1073   }
1074 }
1075 \seq_map_inline:Nn
1076 \c__zrefclever_ref_options_necessarily_type_specific_seq
1077 {
1078   \keys_define:nn { zref-clever / langsetup }
1079   {
1080     #1 .value_required:n = true ,
1081     #1 .code:n =
1082     {
1083       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1084       {
1085         \msg_warning:nnn { zref-clever }
1086         { option-only-type-specific } {#1}
1087       }
1088       {
1089         \__zrefclever_declare_type_transl:Vnn
1090         \l__zrefclever_dict_language_tl
1091         \l__zrefclever_setup_type_tl
1092         {#1} {##1}
1093       }
1094     } ,
1095   }
1096 }

```

6 User interface

6.1 \zcref

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

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

```

1097 \NewDocumentCommand \zcref { s O { } m }
1098 { \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}
```

```
1099 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1100 {
1101   \group_begin:
```

Set options.

```
1102   \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```
1103   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1104   \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.

```
1105   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Integration with zref-check.

```
1106   \bool_lazy_and:nnT
1107     { \l__zrefclever_zrefcheck_available_bool }
1108     { \l__zrefclever_zcref_with_check_bool }
1109     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1110   \bool_lazy_or:nnT
1111     { \l__zrefclever_typeset_sort_bool }
1112     { \l__zrefclever_typeset_range_bool }
1113     { \__zrefclever_sort_labels: }
```

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

```
1114   \group_begin:
1115   \l__zrefclever_ref_typeset_font_tl
1116   \__zrefclever_typeset_refs:
1117   \group_end:
```

Typeset note.

```
1118   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1119   {
1120     \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1121     \l_tmpa_tl
1122     \l__zrefclever_zcref_note_tl
1123   }
```

Integration with zref-check.

```
1124   \bool_lazy_and:nnT
1125     { \l__zrefclever_zrefcheck_available_bool }
1126     { \l__zrefclever_zcref_with_check_bool }
1127   {
```

```

1128         \zrefcheck_zcref_end_label_maybe:
1129         \zrefcheck_zcref_run_checks_on_labels:n
1130         { \l__zrefclever_zcref_labels_seq }
1131     }

```

Integration with mathtools.

```

1132     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1133     {
1134         \__zrefclever_mathtools_showonlyrefs:n
1135         { \l__zrefclever_zcref_labels_seq }
1136     }
1137     \group_end:
1138 }

```

(End definition for `__zrefclever_zcref:nnnn`.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1139 \seq_new:N \l__zrefclever_zcref_labels_seq
1140 \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>}

```

```

1141 \NewDocumentCommand \zcpageref { s O { } m }
1142 {
1143     \IfBooleanTF {#1}
1144     { \zcref*[#2, ref = page] {#3} }
1145     { \zcref [ #2, ref = page] {#3} }
1146 }

```

(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> 1147 \tl_new:N \l__zrefclever_label_type_a_tl 1148 \tl_new:N \l__zrefclever_label_type_b_tl 1149 \tl_new:N \l__zrefclever_label_enclval_a_tl 1150 \tl_new:N \l__zrefclever_label_enclval_b_tl 1151 \tl_new:N \l__zrefclever_label_extdoc_a_tl 1152 \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> 1153 \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> 1154 \int_new:N \l__zrefclever_sort_prior_a_int 1155 \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> 1156 \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> 1157 \cs_new_protected:Npn _zrefclever_sort_labels: 1158 { </pre> <p>Store label types sequence.</p> <pre> 1159 \seq_clear:N \l__zrefclever_label_types_seq 1160 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page } 1161 { 1162 \seq_map_function:NN \l__zrefclever_zcref_labels_seq 1163 _zrefclever_label_type_put_new_right:n 1164 } </pre>

Sort.

```

1165 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1166 {
1167   \zref@ifrefundefined {##1}
1168   {
1169     \zref@ifrefundefined {##2}
1170     {
1171       % Neither label is defined.
1172       \sort_return_same:
1173     }
1174     {
1175       % The second label is defined, but the first isn't, leave the
1176       % undefined first (to be more visible).
1177       \sort_return_same:
1178     }
1179   }
1180   {
1181     \zref@ifrefundefined {##2}
1182     {
1183       % The first label is defined, but the second isn't, bring the
1184       % second forward.
1185       \sort_return_swapped:
1186     }
1187     {
1188       % The interesting case: both labels are defined. References
1189       % to the "default" property or to the "page" are quite
1190       % different with regard to sorting, so we branch them here to
1191       % specialized functions.
1192       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1193       { \__zrefclever_sort_page:nn {##1} {##2} }
1194       { \__zrefclever_sort_default:nn {##1} {##2} }
1195     }
1196   }
1197 }
1198 }

```

(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>}

1199 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1200 {
1201   \__zrefclever_def_extract:Nnnn
1202   \l__zrefclever_label_type_a_tl {#1} {zc@type} { \c_empty_tl }
1203   \seq_if_in:NVF \l__zrefclever_label_types_seq

```

```

1204 \l__zrefclever_label_type_a_tl
1205 {
1206   \seq_put_right:NV \l__zrefclever_label_types_seq
1207   \l__zrefclever_label_type_a_tl
1208 }
1209 }

```

(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}

1210 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1211 {
1212   \__zrefclever_def_extract:Nnnn
1213   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1214   \__zrefclever_def_extract:Nnnn
1215   \l__zrefclever_label_type_b_tl {#2} {zc@type} {\c_empty_tl}
1216
1217   \bool_if:nTF
1218   {
1219     % The second label has a type, but the first doesn't, leave the
1220     % undefined first (to be more visible).
1221     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1222     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1223   }
1224   { \sort_return_same: }
1225   {
1226     \bool_if:nTF
1227     {
1228       % The first label has a type, but the second doesn't, bring the
1229       % second forward.
1230       ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1231       \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1232     }
1233     { \sort_return_swapped: }
1234     {
1235       \bool_if:nTF
1236       {
1237         % The interesting case: both labels have a type...
1238         ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1239         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1240       }
1241       {
1242         \tl_if_eq:NNTF
1243         \l__zrefclever_label_type_a_tl
1244         \l__zrefclever_label_type_b_tl
1245         % ...and it's the same type.
1246         { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1247         % ...and they are different types.

```

```

1248         { \_zrefclever_sort_default_different_types:nn {#1} {#2} }
1249     }
1250     {
1251         % Neither label has a type. We can't do much of meaningful
1252         % here, but if it's the same counter, compare it.
1253         \exp_args:Nxx \tl_if_eq:nnTF
1254         { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
1255         { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
1256         {
1257             \int_compare:nNnTF
1258             { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1259             >
1260             { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1261             { \sort_return_swapped: }
1262             { \sort_return_same: }
1263         }
1264         { \sort_return_same: }
1265     }
1266 }
1267 }
1268 }

```

(End definition for _zrefclever_sort_default:nn.)

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

```

1269 \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>}
1270 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1271 {
1272     \_zrefclever_def_extract:Nnnn \l_zrefclever_label_enclval_a_tl
1273     {#1} { zc@enclval } { \c_empty_tl }
1274     \tl_reverse:N \l_zrefclever_label_enclval_a_tl
1275     \_zrefclever_def_extract:Nnnn \l_zrefclever_label_enclval_b_tl
1276     {#2} { zc@enclval } { \c_empty_tl }
1277     \tl_reverse:N \l_zrefclever_label_enclval_b_tl
1278     \_zrefclever_def_extract:Nnnn \l_zrefclever_label_extdoc_a_tl
1279     {#1} { externaldocument } { \c_empty_tl }
1280     \_zrefclever_def_extract:Nnnn \l_zrefclever_label_extdoc_b_tl
1281     {#2} { externaldocument } { \c_empty_tl }
1282
1283     \bool_set_false:N \l_zrefclever_sort_decided_bool
1284
1285     % First we check if there's any "external document" difference (coming
1286     % from 'zref-xr') and, if so, sort based on that.
1287     \tl_if_eq:NNF
1288     \l_zrefclever_label_extdoc_a_tl
1289     \l_zrefclever_label_extdoc_b_tl
1290     {
1291         \bool_if:nTF
1292         {
1293             \tl_if_empty_p:V \l_zrefclever_label_extdoc_a_tl &&
1294             ! \tl_if_empty_p:V \l_zrefclever_label_extdoc_b_tl
1295         }

```



```

1296     {
1297         \bool_set_true:N \l__zrefclever_sort_decided_bool
1298         \sort_return_same:
1299     }
1300     {
1301         \bool_if:nTF
1302         {
1303             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1304             \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1305         }
1306         {
1307             \bool_set_true:N \l__zrefclever_sort_decided_bool
1308             \sort_return_swapped:
1309         }
1310         {
1311             \bool_set_true:N \l__zrefclever_sort_decided_bool
1312             % Two different "external documents": last resort, sort by the
1313             % document name itself.
1314             \str_compare:eNeTF
1315             { \l__zrefclever_label_extdoc_b_tl } <
1316             { \l__zrefclever_label_extdoc_a_tl }
1317             { \sort_return_swapped: }
1318             { \sort_return_same: }
1319         }
1320     }
1321 }
1322
1323 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1324 {
1325     \bool_if:nTF
1326     {
1327         % Both are empty: neither label has any (further) "enclosing
1328         % counters" (left).
1329         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1330         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1331     }
1332     {
1333         \bool_set_true:N \l__zrefclever_sort_decided_bool
1334         \int_compare:nNnTF
1335         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1336         >
1337         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
1338         { \sort_return_swapped: }
1339         { \sort_return_same: }
1340     }
1341     {
1342         \bool_if:nTF
1343         {
1344             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1345             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1346         }
1347         {
1348             \bool_set_true:N \l__zrefclever_sort_decided_bool
1349             \int_compare:nNnTF

```

```

1350 { \_zrefclever_extract:nnn {#1} { zc@cntval } { } }
1351 >
1352 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1353 { \sort_return_swapped: }
1354 { \sort_return_same: }
1355 }
1356 {
1357 \bool_if:nTF
1358 {
1359 % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1360 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1361 }
1362 {
1363 \bool_set_true:N \l__zrefclever_sort_decided_bool
1364 \int_compare:nNnTF
1365 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1366 <
1367 { \_zrefclever_extract:nnn {#2} { zc@cntval } { } }
1368 { \sort_return_same: }
1369 { \sort_return_swapped: }
1370 }
1371 {
1372 % Neither is empty: we can compare the values of the
1373 % current enclosing counter in the loop, if they are
1374 % equal, we are still in the loop, if they are not, a
1375 % sorting decision can be made directly.
1376 \int_compare:nNnTF
1377 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1378 =
1379 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1380 {
1381 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1382 { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1383 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1384 { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1385 }
1386 {
1387 \bool_set_true:N \l__zrefclever_sort_decided_bool
1388 \int_compare:nNnTF
1389 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1390 >
1391 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1392 { \sort_return_swapped: }
1393 { \sort_return_same: }
1394 }
1395 }
1396 }
1397 }
1398 }
1399 }

```

(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)}

```

1400 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1401 {

```

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”.

```

1402   \int_zero:N \l__zrefclever_sort_prior_a_int
1403   \int_zero:N \l__zrefclever_sort_prior_b_int
1404   \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1405   {
1406     \tl_if_eq:nnTF {##2} {{othertypes}}
1407     {
1408       \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1409       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1410       \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1411       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1412     }
1413     {
1414       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1415       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1416       {
1417         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1418         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1419       }
1420     }
1421   }

```

Then do the actual sorting.

```

1422   \bool_if:nTF
1423   {
1424     \int_compare_p:nNn
1425     { \l__zrefclever_sort_prior_a_int } <
1426     { \l__zrefclever_sort_prior_b_int }
1427   }
1428   { \sort_return_same: }
1429   {
1430     \bool_if:nTF
1431     {
1432       \int_compare_p:nNn
1433       { \l__zrefclever_sort_prior_a_int } >
1434       { \l__zrefclever_sort_prior_b_int }
1435     }
1436     { \sort_return_swapped: }
1437     {
1438       % Sort priorities are equal: the type that occurs first in
1439       % ‘labels’, as given by the user, is kept (or brought) forward.
1440       \seq_map_inline:Nn \l__zrefclever_label_types_seq
1441       {
1442         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1443         { \seq_map_break:n { \sort_return_same: } }
1444         {
1445           \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1446           { \seq_map_break:n { \sort_return_swapped: } }
1447         }
1448       }

```

```

1449         }
1450     }
1451 }

```

(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}

1452 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1453 {
1454     \int_compare:nNnTF
1455         { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1456         >
1457         { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1458         { \sort_return_swapped: }
1459         { \sort_return_same: }
1460 }

```

(End definition for `_zrefclever_sort_page:nn`.)

8 Typesetting

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

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

should be typeset; iv) Similar issues apply to lists of type blocks, each of which is of arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

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

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

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

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

<code>\l__zrefclever_typeset_labels_seq</code>	
<code>\l__zrefclever_typeset_last_bool</code>	1461 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l__zrefclever_last_of_type_bool</code>	1462 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	1463 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>

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

`\l_zrefclever_type_count_int`
`\l__zrefclever_label_count_int`

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

1464 `\int_new:N \l__zrefclever_type_count_int`
1465 `\int_new:N \l__zrefclever_label_count_int`

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

`\l__zrefclever_label_a_tl`
`\l__zrefclever_label_b_tl`
`\l_zrefclever_typeset_queue_prev_tl`
`\l_zrefclever_typeset_queue_curr_tl`
`\l_zrefclever_type_first_label_tl`
`\l_zrefclever_type_first_label_type_tl`

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

1466 `\tl_new:N \l__zrefclever_label_a_tl`
1467 `\tl_new:N \l__zrefclever_label_b_tl`
1468 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
1469 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
1470 `\tl_new:N \l__zrefclever_type_first_label_tl`
1471 `\tl_new:N \l__zrefclever_type_first_label_type_tl`

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

`\l__zrefclever_type_name_tl`
`\l_zrefclever_name_in_link_bool`
`\l_zrefclever_name_format_tl`
`\l_zrefclever_name_format_fallback_tl`

Auxiliary variables for `__zrefclever_typeset_refs`: type name handling.

1472 `\tl_new:N \l__zrefclever_type_name_tl`
1473 `\bool_new:N \l__zrefclever_name_in_link_bool`
1474 `\tl_new:N \l__zrefclever_name_format_tl`
1475 `\tl_new:N \l__zrefclever_name_format_fallback_tl`

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

`\l__zrefclever_range_count_int`
`\l_zrefclever_range_same_count_int`
`\l_zrefclever_range_beg_label_tl`
`\l_zrefclever_next_maybe_range_bool`
`\l_zrefclever_next_is_same_bool`

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

1476 `\int_new:N \l__zrefclever_range_count_int`
1477 `\int_new:N \l__zrefclever_range_same_count_int`
1478 `\tl_new:N \l__zrefclever_range_beg_label_tl`
1479 `\bool_new:N \l__zrefclever_next_maybe_range_bool`
1480 `\bool_new:N \l__zrefclever_next_is_same_bool`

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

`\l__zrefclever_tpairsep_tl`
`\l__zrefclever_tlistsep_tl`
`\l__zrefclever_tlastsep_tl`
`\l__zrefclever_namesep_tl`
`\l__zrefclever_pairsep_tl`
`\l__zrefclever_listsep_tl`
`\l__zrefclever_lastsep_tl`
`\l__zrefclever_rangesep_tl`
`\l_zrefclever_refpre_out_tl`
`\l_zrefclever_refpos_out_tl`
`\l_zrefclever_refpre_in_tl`
`\l_zrefclever_refpos_in_tl`
`\l__zrefclever_namefont_tl`
`\l_zrefclever_reffont_out_tl`
`\l_zrefclever_reffont_in_tl`

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

1481 `\tl_new:N \l__zrefclever_tpairsep_tl`
1482 `\tl_new:N \l__zrefclever_tlistsep_tl`
1483 `\tl_new:N \l__zrefclever_tlastsep_tl`
1484 `\tl_new:N \l__zrefclever_namesep_tl`
1485 `\tl_new:N \l__zrefclever_pairsep_tl`
1486 `\tl_new:N \l__zrefclever_listsep_tl`
1487 `\tl_new:N \l__zrefclever_lastsep_tl`
1488 `\tl_new:N \l__zrefclever_rangesep_tl`
1489 `\tl_new:N \l_zrefclever_refpre_out_tl`
1490 `\tl_new:N \l_zrefclever_refpos_out_tl`
1491 `\tl_new:N \l_zrefclever_refpre_in_tl`
1492 `\tl_new:N \l_zrefclever_refpos_in_tl`
1493 `\tl_new:N \l__zrefclever_namefont_tl`
1494 `\tl_new:N \l_zrefclever_reffont_out_tl`
1495 `\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`.

```

1496 \cs_new_protected:Npn \_zrefclever_typeset_refs:
1497 {
1498   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1499   \l__zrefclever_zcref_labels_seq
1500   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1501   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1502   \tl_clear:N \l__zrefclever_type_first_label_tl
1503   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1504   \tl_clear:N \l__zrefclever_range_beg_label_tl
1505   \int_zero:N \l__zrefclever_label_count_int
1506   \int_zero:N \l__zrefclever_type_count_int
1507   \int_zero:N \l__zrefclever_range_count_int
1508   \int_zero:N \l__zrefclever_range_same_count_int
1509
1510   % Get type block options (not type-specific).
1511   \_zrefclever_get_ref_string:nN { tpairsep }
1512   \l__zrefclever_tpairsep_tl
1513   \_zrefclever_get_ref_string:nN { tlistsep }
1514   \l__zrefclever_tlistsep_tl
1515   \_zrefclever_get_ref_string:nN { tlastsep }
1516   \l__zrefclever_tlastsep_tl
1517
1518   % Process label stack.
1519   \bool_set_false:N \l__zrefclever_typeset_last_bool
1520   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1521   {
1522     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1523     \l__zrefclever_label_a_tl
1524     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1525     {
1526       \tl_clear:N \l__zrefclever_label_b_tl
1527       \bool_set_true:N \l__zrefclever_typeset_last_bool
1528     }
1529     {
1530       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1531       \l__zrefclever_label_b_tl
1532     }
1533
1534     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1535     {
1536       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1537       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1538     }
1539     {
1540       \_zrefclever_def_extract:NVnn \l__zrefclever_label_type_a_tl
1541       \l__zrefclever_label_a_tl { zc@type } { \c_empty_tl }
1542       \_zrefclever_def_extract:NVnn \l__zrefclever_label_type_b_tl
1543       \l__zrefclever_label_b_tl { zc@type } { \c_empty_tl }
1544     }
1545
1546     % First, we establish whether the "current label" (i.e. 'a') is the

```

```

1547 % last one of its type. This can happen because the "next label"
1548 % (i.e. 'b') is of a different type (or different definition status),
1549 % or because we are at the end of the list.
1550 \bool_if:NTF \l__zrefclever_typeset_last_bool
1551 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1552 {
1553   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1554   {
1555     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1556     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1557     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1558   }
1559   {
1560     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1561     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1562     {
1563       % Neither is undefined, we must check the types.
1564       \bool_if:nTF
1565       {
1566         % Both empty: same "type".
1567         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1568         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1569       }
1570       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1571       {
1572         \bool_if:nTF
1573         {
1574           % Neither empty: compare types.
1575           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1576           &&
1577           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1578         }
1579         {
1580           \tl_if_eq:NNTF
1581           \l__zrefclever_label_type_a_tl
1582           \l__zrefclever_label_type_b_tl
1583           {
1584             \bool_set_false:N
1585             \l__zrefclever_last_of_type_bool
1586           }
1587           {
1588             \bool_set_true:N
1589             \l__zrefclever_last_of_type_bool
1590           }
1591         }
1592         % One empty, the other not: different "types".
1593         {
1594           \bool_set_true:N
1595           \l__zrefclever_last_of_type_bool
1596         }
1597       }
1598     }
1599   }
1600 }

```



```

1601
1602 % Handle warnings in case of reference or type undefined.
1603 \zref@refused { \l__zrefclever_label_a_tl }
1604 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1605 {}
1606 {
1607   \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1608   {
1609     \msg_warning:nxx { zref-clever } { missing-type }
1610     { \l__zrefclever_label_a_tl }
1611   }
1612 }
1613
1614 % Get type-specific separators, refpre/pos and font options, once per
1615 % type.
1616 \int_compare:nNtF { \l__zrefclever_label_count_int } = { 0 }
1617 {
1618   \__zrefclever_get_ref_string:nN { namesep      }
1619   \l__zrefclever_namesep_tl
1620   \__zrefclever_get_ref_string:nN { rangesep     }
1621   \l__zrefclever_rangesep_tl
1622   \__zrefclever_get_ref_string:nN { pairsep      }
1623   \l__zrefclever_pairsep_tl
1624   \__zrefclever_get_ref_string:nN { listsep      }
1625   \l__zrefclever_listsep_tl
1626   \__zrefclever_get_ref_string:nN { lastsep      }
1627   \l__zrefclever_lastsep_tl
1628   \__zrefclever_get_ref_string:nN { refpre       }
1629   \l__zrefclever_refpre_out_tl
1630   \__zrefclever_get_ref_string:nN { refpos       }
1631   \l__zrefclever_refpos_out_tl
1632   \__zrefclever_get_ref_string:nN { refpre-in    }
1633   \l__zrefclever_refpre_in_tl
1634   \__zrefclever_get_ref_string:nN { refpos-in    }
1635   \l__zrefclever_refpos_in_tl
1636   \__zrefclever_get_ref_font:nN   { namefont     }
1637   \l__zrefclever_namefont_tl
1638   \__zrefclever_get_ref_font:nN   { reffont      }
1639   \l__zrefclever_reffont_out_tl
1640   \__zrefclever_get_ref_font:nN   { reffont-in   }
1641   \l__zrefclever_reffont_in_tl
1642 }
1643
1644 % Here we send this to a couple of auxiliary functions.
1645 \bool_if:NTF \l__zrefclever_last_of_type_bool
1646 % There exists no next label of the same type as the current.
1647 { \__zrefclever_typeset_refs_last_of_type: }
1648 % There exists a next label of the same type as the current.
1649 { \__zrefclever_typeset_refs_not_last_of_type: }
1650 }
1651 }

```

(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.

```

1652 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1653 {
1654   % Process the current label to the current queue.
1655   \int_case:nnF { \l__zrefclever_label_count_int }
1656   {
1657     % It is the last label of its type, but also the first one, and that's
1658     % what matters here: just store it.
1659     { 0 }
1660     {
1661       \tl_set:NV \l__zrefclever_type_first_label_tl
1662       \l__zrefclever_label_a_tl
1663       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1664       \l__zrefclever_label_type_a_tl
1665     }
1666
1667     % The last is the second: we have a pair (if not repeated).
1668     { 1 }
1669     {
1670       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1671       {
1672         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1673         {
1674           \exp_not:V \l__zrefclever_pairsep_tl
1675           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1676         }
1677       }
1678     }
1679   }
1680   % Last is third or more of its type: without repetition, we'd have the
1681   % last element on a list, but control for possible repetition.
1682   {
1683     \int_case:nnF { \l__zrefclever_range_count_int }
1684     {
1685       % There was no range going on.
1686       { 0 }
1687       {
1688         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1689         {
1690           \exp_not:V \l__zrefclever_lastsep_tl
1691           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1692         }
1693       }
1694     }
    % Last in the range is also the second in it.

```

```

1695 { 1 }
1696 {
1697   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1698   {
1699     % We know 'range_beg_label' is not empty, since this is the
1700     % second element in the range, but the third or more in the
1701     % type list.
1702     \exp_not:V \l__zrefclever_listsep_tl
1703     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1704     \int_compare:nNnF
1705       { \l__zrefclever_range_same_count_int } = { 1 }
1706       {
1707         \exp_not:V \l__zrefclever_lastsep_tl
1708         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1709       }
1710   }
1711 }
1712 }
1713 % Last in the range is third or more in it.
1714 {
1715   \int_case:nnF
1716   {
1717     \l__zrefclever_range_count_int -
1718     \l__zrefclever_range_same_count_int
1719   }
1720   {
1721     % Repetition, not a range.
1722     { 0 }
1723     {
1724       % If 'range_beg_label' is empty, it means it was also the
1725       % first of the type, and hence was already handled.
1726       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1727       {
1728         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1729         {
1730           \exp_not:V \l__zrefclever_lastsep_tl
1731           \__zrefclever_get_ref:V
1732             \l__zrefclever_range_beg_label_tl
1733         }
1734       }
1735     }
1736     % A 'range', but with no skipped value, treat as list.
1737     { 1 }
1738     {
1739       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1740       {
1741         % Ditto.
1742         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1743         {
1744           \exp_not:V \l__zrefclever_listsep_tl
1745           \__zrefclever_get_ref:V
1746             \l__zrefclever_range_beg_label_tl
1747         }
1748         \exp_not:V \l__zrefclever_lastsep_tl

```

```

1749         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1750     }
1751 }
1752 }
1753 {
1754     % An actual range.
1755     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1756     {
1757         % Ditto.
1758         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1759         {
1760             \exp_not:V \l__zrefclever_lastsep_tl
1761             \l__zrefclever_get_ref:V
1762             \l__zrefclever_range_beg_label_tl
1763         }
1764         \exp_not:V \l__zrefclever_rangesep_tl
1765         \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1766     }
1767 }
1768 }
1769 }
1770
1771 % Handle "range" option. The idea is simple: if the queue is not empty,
1772 % we replace it with the end of the range (or pair). We can still
1773 % retrieve the end of the range from 'label_a' since we know to be
1774 % processing the last label of its type at this point.
1775 \bool_if:NT \l__zrefclever_typeset_range_bool
1776 {
1777     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1778     {
1779         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1780         { }
1781         {
1782             \msg_warning:nxx { zref-clever } { single-element-range }
1783             { \l__zrefclever_type_first_label_type_tl }
1784         }
1785     }
1786     {
1787         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1788         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1789         { }
1790         {
1791             \l__zrefclever_labels_in_sequence:nn
1792             { \l__zrefclever_type_first_label_tl }
1793             { \l__zrefclever_label_a_tl }
1794         }
1795         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1796         {
1797             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1798             { \exp_not:V \l__zrefclever_pairsep_tl }
1799             { \exp_not:V \l__zrefclever_rangesep_tl }
1800             \l__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1801         }
1802     }

```

```

1803     }
1804
1805 % Now that the type block is finished, we can add the name and the first
1806 % ref to the queue. Also, if "typeset" option is not "both", handle it
1807 % here as well.
1808 \__zrefclever_type_name_setup:
1809 \bool_if:nTF
1810 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1811 {
1812     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1813     { \__zrefclever_get_ref_first: }
1814 }
1815 {
1816     \bool_if:nTF
1817     { \l__zrefclever_typeset_ref_bool }
1818     {
1819         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1820         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1821     }
1822     {
1823         \bool_if:nTF
1824         { \l__zrefclever_typeset_name_bool }
1825         {
1826             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1827             {
1828                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1829                 {
1830                     \exp_not:N \group_begin:
1831                     \exp_not:V \l__zrefclever_namefont_tl
1832                     % It's two '@s', but escaped for DocStrip.
1833                     \exp_not:N \hyper@@link
1834                     {
1835                         \__zrefclever_extract_url_unexp:V
1836                         \l__zrefclever_type_first_label_tl
1837                     }
1838                     {
1839                         \__zrefclever_extract_unexp:Vnn
1840                         \l__zrefclever_type_first_label_tl
1841                         { anchor } { }
1842                     }
1843                     { \exp_not:V \l__zrefclever_type_name_tl }
1844                     \exp_not:N \group_end:
1845                 }
1846             }
1847             \exp_not:N \group_begin:
1848             \exp_not:V \l__zrefclever_namefont_tl
1849             \exp_not:V \l__zrefclever_type_name_tl
1850             \exp_not:N \group_end:
1851         }
1852     }
1853 }
1854 {
1855     % Logically, this case would correspond to "typeset=none", but
1856     % it should not occur, given that the options are set up to

```

```

1857             % typeset either "ref" or "name". Still, leave here a
1858             % sensible fallback, equal to the behavior of "both".
1859             \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1860             { \__zrefclever_get_ref_first: }
1861         }
1862     }
1863 }
1864
1865 % Typeset the previous type, if there is one.
1866 \int_compare:nNtT { \l__zrefclever_type_count_int } > { 0 }
1867 {
1868     \int_compare:nNtT { \l__zrefclever_type_count_int } > { 1 }
1869     { \l__zrefclever_tlistsep_tl }
1870     \l__zrefclever_typeset_queue_prev_tl
1871 }
1872
1873 % Wrap up loop, or prepare for next iteration.
1874 \bool_if:NTF \l__zrefclever_typeset_last_bool
1875 {
1876     % We are finishing, typeset the current queue.
1877     \int_case:nnF { \l__zrefclever_type_count_int }
1878     {
1879         % Single type.
1880         { 0 }
1881         { \l__zrefclever_typeset_queue_curr_tl }
1882         % Pair of types.
1883         { 1 }
1884         {
1885             \l__zrefclever_tpairsep_tl
1886             \l__zrefclever_typeset_queue_curr_tl
1887         }
1888     }
1889     {
1890         % Last in list of types.
1891         \l__zrefclever_tlastsep_tl
1892         \l__zrefclever_typeset_queue_curr_tl
1893     }
1894 }
1895 {
1896     % There are further labels, set variables for next iteration.
1897     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1898     \l__zrefclever_typeset_queue_curr_tl
1899     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1900     \tl_clear:N \l__zrefclever_type_first_label_tl
1901     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1902     \tl_clear:N \l__zrefclever_range_beg_label_tl
1903     \int_zero:N \l__zrefclever_label_count_int
1904     \int_incr:N \l__zrefclever_type_count_int
1905     \int_zero:N \l__zrefclever_range_count_int
1906     \int_zero:N \l__zrefclever_range_same_count_int
1907 }
1908 }

```

(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.

```
1909 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1910 {
1911   % Signal if next label may form a range with the current one (only
1912   % considered if compression is enabled in the first place).
1913   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1914   \bool_set_false:N \l__zrefclever_next_is_same_bool
1915   \bool_if:NT \l__zrefclever_typeset_compress_bool
1916   {
1917     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1918     { }
1919     {
1920       \__zrefclever_labels_in_sequence:nn
1921       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1922     }
1923   }
1924
1925   % Process the current label to the current queue.
1926   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1927   {
1928     % Current label is the first of its type (also not the last, but it
1929     % doesn't matter here): just store the label.
1930     \tl_set:NV \l__zrefclever_type_first_label_tl
1931     \l__zrefclever_label_a_tl
1932     \tl_set:NV \l__zrefclever_type_first_label_type_tl
1933     \l__zrefclever_label_type_a_tl
1934
1935     % If the next label may be part of a range, we set 'range_beg_label'
1936     % to "empty" (we deal with it as the "first", and must do it there, to
1937     % handle hyperlinking), but also step the range counters.
1938     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1939     {
1940       \tl_clear:N \l__zrefclever_range_beg_label_tl
1941       \int_incr:N \l__zrefclever_range_count_int
1942       \bool_if:NT \l__zrefclever_next_is_same_bool
1943       { \int_incr:N \l__zrefclever_range_same_count_int }
1944     }
1945   }
1946   {
1947     % Current label is neither the first (nor the last) of its type.
1948     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1949     {
1950       % Starting, or continuing a range.
1951       \int_compare:nNnTF
1952       { \l__zrefclever_range_count_int } = { 0 }
1953       {
1954         % There was no range going, we are starting one.
1955         \tl_set:NV \l__zrefclever_range_beg_label_tl
1956         \l__zrefclever_label_a_tl
1957         \int_incr:N \l__zrefclever_range_count_int
1958         \bool_if:NT \l__zrefclever_next_is_same_bool
1959         { \int_incr:N \l__zrefclever_range_same_count_int }
1960       }
1961       {
```

```

1962         % Second or more in the range, but not the last.
1963         \int_incr:N \l__zrefclever_range_count_int
1964         \bool_if:NT \l__zrefclever_next_is_same_bool
1965         { \int_incr:N \l__zrefclever_range_same_count_int }
1966     }
1967 }
1968 {
1969     % Next element is not in sequence: there was no range, or we are
1970     % closing one.
1971     \int_case:nnF { \l__zrefclever_range_count_int }
1972     {
1973         % There was no range going on.
1974         { 0 }
1975         {
1976             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1977             {
1978                 \exp_not:V \l__zrefclever_listsep_tl
1979                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1980             }
1981         }
1982         % Last is second in the range: if 'range_same_count' is also
1983         % '1', it's a repetition (drop it), otherwise, it's a "pair
1984         % within a list", treat as list.
1985         { 1 }
1986         {
1987             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1988             {
1989                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1990                 {
1991                     \exp_not:V \l__zrefclever_listsep_tl
1992                     \__zrefclever_get_ref:V
1993                     \l__zrefclever_range_beg_label_tl
1994                 }
1995                 \int_compare:nNnF
1996                 { \l__zrefclever_range_same_count_int } = { 1 }
1997                 {
1998                     \exp_not:V \l__zrefclever_listsep_tl
1999                     \__zrefclever_get_ref:V
2000                     \l__zrefclever_label_a_tl
2001                 }
2002             }
2003         }
2004     }
2005 }
2006 {
2007     % Last is third or more in the range: if 'range_count' and
2008     % 'range_same_count' are the same, its a repetition (drop it),
2009     % if they differ by '1', its a list, if they differ by more,
2010     % it is a real range.
2011     \int_case:nnF
2012     {
2013         \l__zrefclever_range_count_int -
2014         \l__zrefclever_range_same_count_int
2015     }
2016     {

```



```

2016         { 0 }
2017     {
2018         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2019         {
2020             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2021             {
2022                 \exp_not:V \l__zrefclever_listsep_tl
2023                 \__zrefclever_get_ref:V
2024                 \l__zrefclever_range_beg_label_tl
2025             }
2026         }
2027     }
2028     { 1 }
2029     {
2030         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2031         {
2032             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2033             {
2034                 \exp_not:V \l__zrefclever_listsep_tl
2035                 \__zrefclever_get_ref:V
2036                 \l__zrefclever_range_beg_label_tl
2037             }
2038             \exp_not:V \l__zrefclever_listsep_tl
2039             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2040         }
2041     }
2042 }
2043 {
2044     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2045     {
2046         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2047         {
2048             \exp_not:V \l__zrefclever_listsep_tl
2049             \__zrefclever_get_ref:V
2050             \l__zrefclever_range_beg_label_tl
2051         }
2052         \exp_not:V \l__zrefclever_rangesep_tl
2053         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2054     }
2055 }
2056 }
2057 % Reset counters.
2058 \int_zero:N \l__zrefclever_range_count_int
2059 \int_zero:N \l__zrefclever_range_same_count_int
2060 }
2061 }
2062 % Step label counter for next iteration.
2063 \int_incr:N \l__zrefclever_label_count_int
2064 }

```

(End definition for __zrefclever_typeset_refs_not_last_of_type:.)

Aux functions

`__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:n` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` get called, as they must, in the context of \mathfrak{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 \mathfrak{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.

```

2065 \cs_new_protected:Npn \__zrefclever_ref_default:
2066   { \zref@default }
2067 \cs_new_protected:Npn \__zrefclever_name_default:
2068   { \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>}

2069 \cs_new:Npn \__zrefclever_get_ref:n #1
2070   {
2071     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2072     {
2073       \bool_if:nTF
2074         {
2075           \l__zrefclever_use_hyperref_bool &&
2076           ! \l__zrefclever_link_star_bool
2077         }
2078         {
2079           \exp_not:N \group_begin:
2080           \exp_not:N \l__zrefclever_reffont_out_tl
2081           \exp_not:N \l__zrefclever_refpre_out_tl

```

```

2082 \exp_not:N \group_begin:
2083 \exp_not:V \l__zrefclever_reffont_in_tl
2084 % It's two '@s', but escaped for DocStrip.
2085 \exp_not:N \hyper@@link
2086 { \__zrefclever_extract_url_unexp:n {#1} }
2087 { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2088 {
2089 \exp_not:V \l__zrefclever_refpre_in_tl
2090 \__zrefclever_extract_unexp:nvn {#1}
2091 { l__zrefclever_ref_property_tl } { }
2092 \exp_not:V \l__zrefclever_refpos_in_tl
2093 }
2094 \exp_not:N \group_end:
2095 \exp_not:V \l__zrefclever_refpos_out_tl
2096 \exp_not:N \group_end:
2097 }
2098 {
2099 \exp_not:N \group_begin:
2100 \exp_not:V \l__zrefclever_reffont_out_tl
2101 \exp_not:V \l__zrefclever_refpre_out_tl
2102 \exp_not:N \group_begin:
2103 \exp_not:V \l__zrefclever_reffont_in_tl
2104 \exp_not:V \l__zrefclever_refpre_in_tl
2105 \__zrefclever_extract_unexp:nvn {#1}
2106 { l__zrefclever_ref_property_tl } { }
2107 \exp_not:V \l__zrefclever_refpos_in_tl
2108 \exp_not:N \group_end:
2109 \exp_not:V \l__zrefclever_refpos_out_tl
2110 \exp_not:N \group_end:
2111 }
2112 }
2113 { \__zrefclever_ref_default: }
2114 }
2115 \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.

```

2116 \cs_new:Npn \__zrefclever_get_ref_first:
2117 {
2118 \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2119 { \__zrefclever_ref_default: }
2120 {
2121 \bool_if:NTF \l__zrefclever_name_in_link_bool
2122 {
2123 \zref@ifrefcontainsprop
2124 { \l__zrefclever_type_first_label_tl }

```

```

2125 { \l__zrefclever_ref_property_tl }
2126 {
2127   % It's two '@s', but escaped for DocStrip.
2128   \exp_not:N \hyper@@link
2129   {
2130     \__zrefclever_extract_url_unexp:V
2131     \l__zrefclever_type_first_label_tl
2132   }
2133   {
2134     \__zrefclever_extract_unexp:Vnn
2135     \l__zrefclever_type_first_label_tl { anchor } { }
2136   }
2137   {
2138     \exp_not:N \group_begin:
2139     \exp_not:V \l__zrefclever_namefont_tl
2140     \exp_not:V \l__zrefclever_type_name_tl
2141     \exp_not:N \group_end:
2142     \exp_not:V \l__zrefclever_namesep_tl
2143     \exp_not:N \group_begin:
2144     \exp_not:V \l__zrefclever_reffont_out_tl
2145     \exp_not:V \l__zrefclever_refpre_out_tl
2146     \exp_not:N \group_begin:
2147     \exp_not:V \l__zrefclever_reffont_in_tl
2148     \exp_not:V \l__zrefclever_refpre_in_tl
2149     \__zrefclever_extract_unexp:Vnn
2150     \l__zrefclever_type_first_label_tl
2151     { \l__zrefclever_ref_property_tl } { }
2152     \exp_not:V \l__zrefclever_refpos_in_tl
2153     \exp_not:N \group_end:
2154     % hyperlink makes it's own group, we'd like to close the
2155     % 'refpre-out' group after 'refpos-out', but... we close
2156     % it here, and give the trailing 'refpos-out' its own
2157     % group. This will result that formatting given to
2158     % 'refpre-out' will not reach 'refpos-out', but I see no
2159     % alternative, and this has to be handled specially.
2160     \exp_not:N \group_end:
2161   }
2162   \exp_not:N \group_begin:
2163   % Ditto: special treatment.
2164   \exp_not:V \l__zrefclever_reffont_out_tl
2165   \exp_not:V \l__zrefclever_refpos_out_tl
2166   \exp_not:N \group_end:
2167 }
2168 {
2169   \exp_not:N \group_begin:
2170   \exp_not:V \l__zrefclever_namefont_tl
2171   \exp_not:V \l__zrefclever_type_name_tl
2172   \exp_not:N \group_end:
2173   \exp_not:V \l__zrefclever_namesep_tl
2174   \__zrefclever_ref_default:
2175 }
2176 }
2177 {
2178   \tl_if_empty:NTF \l__zrefclever_type_name_tl

```

```

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

```

```

2233         \_zrefclever_extract_unexp:Vvn
2234         \l__zrefclever_type_first_label_tl
2235         { \l__zrefclever_ref_property_tl } { }
2236         \exp_not:V \l__zrefclever_refpos_in_tl
2237         \exp_not:N \group_end:
2238         \exp_not:V \l__zrefclever_refpos_out_tl
2239         \exp_not:N \group_end:
2240     }
2241 }
2242 { \_zrefclever_ref_default: }
2243 }
2244 }
2245 }

```

(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.

```

2246 \cs_new_protected:Npn \_zrefclever_type_name_setup:
2247 {
2248     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2249     { \tl_clear:N \l__zrefclever_type_name_tl }
2250     {
2251         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2252         { \tl_clear:N \l__zrefclever_type_name_tl }
2253         {
2254             % Determine whether we should use capitalization, abbreviation,
2255             % and plural.
2256             \bool_lazy_or:nnTF
2257             { \l__zrefclever_capitalize_bool }
2258             {
2259                 \l__zrefclever_capitalize_first_bool &&
2260                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2261             }
2262             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2263             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2264             % If the queue is empty, we have a singular, otherwise, plural.
2265             \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2266             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2267             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2268             \bool_lazy_and:nnTF
2269             { \l__zrefclever_abbrev_bool }
2270             {
2271                 ! \int_compare_p:nNn

```

```

2272         { \l__zrefclever_type_count_int } = { 0 } ||
2273         ! \l__zrefclever_noabbrev_first_bool
2274     }
2275     {
2276         \tl_set:NV \l__zrefclever_name_format_fallback_tl
2277             \l__zrefclever_name_format_tl
2278         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2279     }
2280     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2281
2282 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2283 {
2284     \prop_get:cVNF
2285     {
2286         l__zrefclever_type_
2287         \l__zrefclever_type_first_label_type_tl _options_prop
2288     }
2289     \l__zrefclever_name_format_tl
2290     \l__zrefclever_type_name_tl
2291     {
2292         \__zrefclever_get_type_transl:xxxNF
2293         { \l__zrefclever_ref_language_tl }
2294         { \l__zrefclever_type_first_label_type_tl }
2295         { \l__zrefclever_name_format_tl }
2296         \l__zrefclever_type_name_tl
2297         {
2298             \tl_clear:N \l__zrefclever_type_name_tl
2299             \msg_warning:nxx { zref-clever } { missing-name }
2300             { \l__zrefclever_type_first_label_type_tl }
2301         }
2302     }
2303 }
2304 {
2305     \prop_get:cVNF
2306     {
2307         l__zrefclever_type_
2308         \l__zrefclever_type_first_label_type_tl _options_prop
2309     }
2310     \l__zrefclever_name_format_tl
2311     \l__zrefclever_type_name_tl
2312     {
2313         \prop_get:cVNF
2314         {
2315             l__zrefclever_type_
2316             \l__zrefclever_type_first_label_type_tl _options_prop
2317         }
2318         \l__zrefclever_name_format_fallback_tl
2319         \l__zrefclever_type_name_tl
2320         {
2321             \__zrefclever_get_type_transl:xxxNF
2322             { \l__zrefclever_ref_language_tl }
2323             { \l__zrefclever_type_first_label_type_tl }
2324             { \l__zrefclever_name_format_tl }
2325             \l__zrefclever_type_name_tl

```

```

2326         {
2327             \__zrefclever_get_type_transl:xxxNF
2328             { \l__zrefclever_ref_language_tl }
2329             { \l__zrefclever_type_first_label_type_tl }
2330             { \l__zrefclever_name_format_fallback_tl }
2331             \l__zrefclever_type_name_tl
2332             {
2333                 \tl_clear:N \l__zrefclever_type_name_tl
2334                 \msg_warning:nmx { zref-clever }
2335                 { missing-name }
2336                 { \l__zrefclever_type_first_label_type_tl }
2337             }
2338         }
2339     }
2340 }
2341 }
2342 }
2343 }
2344
2345 % Signal whether the type name is to be included in the hyperlink or not.
2346 \bool_lazy_any:nTF
2347 {
2348     { ! \l__zrefclever_use_hyperref_bool }
2349     { \l__zrefclever_link_star_bool }
2350     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2351     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2352 }
2353 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2354 {
2355     \bool_lazy_any:nTF
2356     {
2357         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2358         {
2359             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2360             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2361         }
2362         {
2363             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2364             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2365             \l__zrefclever_typeset_last_bool &&
2366             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2367         }
2368     }
2369     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2370     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2371 }
2372 }

```

(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_unexp:nnn.


```

2373 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2374 {
2375   \zref@ifpropundefined { urluse }
2376   { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2377   {
2378     \zref@ifrefcontainsprop {#1} { urluse }
2379     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2380     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2381   }
2382 }
2383 \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 {\label a} {\label b}

2384 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2385 {
2386   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2387   {#1} { externaldocument } { \c_empty_tl }
2388   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2389   {#2} { externaldocument } { \c_empty_tl }
2390
2391   \tl_if_eq:NNT
2392   \l__zrefclever_label_extdoc_a_tl
2393   \l__zrefclever_label_extdoc_b_tl
2394   {
2395     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2396     {
2397       \exp_args:Nxx \tl_if_eq:nnT
2398       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2399       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2400       {
2401         \int_compare:nNnTF
2402         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2403         =
2404         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2405         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2406         {
2407           \int_compare:nNnTF
2408           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2409           =
2410           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2411           {
2412             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2413             \bool_set_true:N \l__zrefclever_next_is_same_bool
2414           }
2415         }
2416       }
2417     }
2418   }
2419 }

```

```

2415     }
2416   }
2417 }
2418 {
2419   \exp_args:Nxx \tl_if_eq:nnT
2420   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2421   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2422   {
2423     \exp_args:Nxx \tl_if_eq:nnT
2424     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2425     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2426     {
2427       \int_compare:nNnTF
2428       { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2429       =
2430       { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2431       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2432       {
2433         \int_compare:nNnTF
2434         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2435         =
2436         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2437         {
2438           \bool_set_true:N
2439           \l__zrefclever_next_maybe_range_bool
2440           \exp_args:Nxx \tl_if_eq:nnT
2441           {
2442             \__zrefclever_extract_unexp:nvn {#1}
2443             { \l__zrefclever_ref_property_tl } { }
2444           }
2445           {
2446             \__zrefclever_extract_unexp:nvn {#2}
2447             { \l__zrefclever_ref_property_tl } { }
2448           }
2449           {
2450             \bool_set_true:N
2451             \l__zrefclever_next_is_same_bool
2452           }
2453         }
2454       }
2455     }
2456   }
2457 }
2458 }
2459 }

```

(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>}
2460 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2461 {
2462   % First attempt: general options.
2463   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2464   {
2465     % If not found, try type specific options.
2466     \bool_lazy_all:nTF
2467     {
2468       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2469       {
2470         \prop_if_exist_p:c
2471         {
2472           l__zrefclever_type_
2473           \l__zrefclever_label_type_a_tl _options_prop
2474         }
2475       }
2476       {
2477         \prop_if_in_p:cn
2478         {
2479           l__zrefclever_type_
2480           \l__zrefclever_label_type_a_tl _options_prop
2481         }
2482         {#1}
2483       }
2484     }
2485     {
2486       \prop_get:cnN
2487       {
2488         l__zrefclever_type_
2489         \l__zrefclever_label_type_a_tl _options_prop
2490       }
2491       {#1} #2
2492     }
2493     {
2494       % If not found, try type specific translations.
2495       \__zrefclever_get_type_transl:xxnNF
2496       { \l__zrefclever_ref_language_tl }
2497       { \l__zrefclever_label_type_a_tl }
2498       {#1} #2
2499       {
2500         % If not found, try default translations.
2501         \__zrefclever_get_default_transl:xxnNF
2502         { \l__zrefclever_ref_language_tl }
2503         {#1} #2
2504         {
2505           % If not found, try fallback.

```

```

2506         \_zrefclever_get_fallback_transl:nNF {#1} #2
2507         {
2508             \tl_clear:N #2
2509             \msg_warning:nnn { zref-clever }
2510             { missing-string } {#1}
2511         }
2512     }
2513 }
2514 }
2515 }
2516 }

```

(End definition for _zrefclever_get_ref_string:nN.)

```

\_zrefclever_get_ref_font:nN      \_zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2517 \cs_new_protected:Npn \_zrefclever_get_ref_font:nN #1#2
2518 {
2519     % First attempt: general options.
2520     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2521     {
2522         % If not found, try type specific options.
2523         \bool_lazy_and:nnTF
2524         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2525         {
2526             \prop_if_exist_p:c
2527             {
2528                 l__zrefclever_type_
2529                 \l__zrefclever_label_type_a_tl _options_prop
2530             }
2531         }
2532         {
2533             \prop_get:cnNF
2534             {
2535                 l__zrefclever_type_
2536                 \l__zrefclever_label_type_a_tl _options_prop
2537             }
2538             {#1} #2
2539             { \tl_clear:N #2 }
2540         }
2541         { \tl_clear:N #2 }
2542     }
2543 }

```

(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.

```
2544 \AddToHook { begindocument }
2545 {
2546   \cs_set_eq:NN \_zrefclever_orig_label:n \label
2547 }
2548 \cs_new_nopar:Npn \_zrefclever_ride_on_label:n #1
2549 {
2550   \_zrefclever_orig_label:n {#1}
2551   \zlabel {#1}
2552 }
```

(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.

```
2553 \tl_new:N \l__zrefclever_footnote_type_tl
2554 \tl_set:Nn \l__zrefclever_footnote_type_tl { footnote }
2555 \AddToHook { env / minipage / begin }
2556 { \tl_set:Nn \l__zrefclever_footnote_type_tl { mpfootnote } }
2557 \AddToHook { cmd / @makefntext / before }
2558 {
2559   \_zrefclever_zcsetup:x
2560   { currentcounter = \l__zrefclever_footnote_type_tl }
2561 }
```

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.

```

2562 \AddToHook { cmd / appendix / before }
2563 {
2564   \__zrefclever_zcsetup:n
2565   {
2566     countertype =
2567     {
2568       chapter      = appendix ,
2569       section      = appendix ,
2570       subsection   = appendix ,
2571       subsubsection = appendix ,
2572     }
2573   }
2574 }

```

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.

```

2575 \AddToHook { begindocument }
2576 {
2577   \@ifpackageloaded { appendix }
2578   {
2579     \newcounter { zc@appendix }
2580     \newcounter { zc@save@appendix }
2581     \setcounter { zc@appendix } { 0 }
2582     \setcounter { zc@save@appendix } { 0 }
2583     \cs_if_exist:cTF { chapter }
2584     {

```

```

2585         \cs_if_exist:cT { section }
2586         {
2587             \__zrefclever_zcsetup:n
2588             { counterresetby = { section = zc@appendix } }
2589         }
2590     }
2591     {
2592         \__zrefclever_zcsetup:n
2593         { counterresetby = { chapter = zc@appendix } }
2594     }
2595     \AddToHook { env / appendices / begin }
2596     {
2597         \stepcounter { zc@save@appendix }
2598         \setcounter { zc@appendix } { \value { zc@save@appendix } }
2599         \__zrefclever_zcsetup:n
2600         {
2601             countertype =
2602             {
2603                 chapter      = appendix ,
2604                 section      = appendix ,
2605                 subsection   = appendix ,
2606                 subsubsection = appendix ,
2607             }
2608         }
2609     }
2610     \AddToHook { env / appendices / end }
2611     { \setcounter { zc@appendix } { 0 } }
2612     \AddToHook { cmd / appendix / before }
2613     {
2614         \stepcounter { zc@save@appendix }
2615         \setcounter { zc@appendix } { \value { zc@save@appendix } }
2616     }
2617     \AddToHook { env / subappendices / begin }
2618     {
2619         \__zrefclever_zcsetup:n
2620         {
2621             countertype =
2622             {
2623                 section      = appendix ,
2624                 subsection   = appendix ,
2625                 subsubsection = appendix ,
2626             } ,
2627         }
2628     }
2629     \msg_info:nnn { zref-clever } { compat-package } { appendix }
2630 }
2631 {}
2632 }

```

9.4 amsmath package

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

```

2633 \AddToHook { begindocument }

```

```

2634 {
2635   \@ifpackageloaded { amsmath }
2636   {

```

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...).

```

2637     \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
2638     {
2639       \__zrefclever_orig_ltxlabel:n {#1}
2640       \zref@wrapper@babel \zref@label {#1}
2641     }

```

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.

```

2642     \IfFormatAtLeastTF { 2021-11-15 }
2643     {
2644       \@ifpackageloaded { hyperref }
2645       {
2646         \AddToHook { package / nameref / after }
2647         {
2648           \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2649           \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2650         }
2651       }
2652       {
2653         \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2654         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2655       }
2656     }
2657     {
2658       \@ifpackageloaded { hyperref }
2659       {
2660         \ifpackageloaded { nameref }
2661         {
2662           \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2663           \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2664         }
2665         {
2666           \AddToHook { package / after / nameref }
2667           {
2668             \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2669             \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2670           }
2671         }
2672       }

```



```

2673         {
2674         \cs_set_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
2675         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
2676         }
2677     }

```

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).

```

2678     \AddToHook { env / subequations / begin }
2679     {
2680         \__zrefclever_zcsetup:x
2681         {
2682             counterresetby =
2683             {
2684                 parentequation =
2685                 \__zrefclever_counter_reset_by:n { equation } ,
2686                 equation = parentequation ,
2687             } ,
2688             currentcounter = parentequation ,
2689             countertype = { parentequation = equation } ,
2690         }
2691     }

```

`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.

```

2692     \clist_map_inline:nn
2693     {
2694         equation ,
2695         equation* ,
2696         align ,
2697         align* ,
2698         alignat ,
2699         alignat* ,
2700         flalign ,
2701         flalign* ,
2702         xalignat ,

```

```

2703         xalignat* ,
2704         gather ,
2705         gather* ,
2706         multiline ,
2707         multiline* ,
2708     }
2709     {
2710         \AddToHook { env / #1 / begin }
2711             { \__zrefclever_zcsetup:n { currentcounter = equation } }
2712     }

```

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

```

2713     \zcRefTypeSetup { equation } { reffont = \upshape }
2714     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
2715 }
2716 {}
2717 }

```

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.

```

2718 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
2719 \AddToHook { begindocument }
2720 {
2721     \@ifpackageloaded { mathtools }
2722     {
2723         \MH_if_boolean:nT { show_only_refs }
2724         {
2725             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
2726             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
2727             {
2728                 \seq_map_inline:Nn #1
2729                 {
2730                     \exp_args:Nx \tl_if_eq:nnTF
2731                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
2732                     { equation }
2733                     {
2734                         \protected@write \@auxout { }
2735                         { \string \MT@newlabel {##1} }
2736                     }
2737                 }
2738                 \exp_args:Nx \tl_if_eq:nnT
2739                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
2740                 { parentequation }

```

```

2741         {
2742             \protected@write \@auxout { }
2743             { \string \MT@newlabel {##1} }
2744         }
2745     }
2746 }
2747 }
2748 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
2749 }
2750 }
2751 {}
2752 }

```

9.6 listings package

```

2753 \AddToHook { begindocument }
2754 {
2755     \@ifpackageloaded { listings }
2756     {
2757         \__zrefclever_zcsetup:n
2758         {
2759             countertype =
2760             {
2761                 lstlisting = listing ,
2762                 lstnumber = line ,
2763             } ,
2764             counterresetby = { lstnumber = lstlisting } ,
2765         }
2766         \lst@AddToHook { Init }
2767         {

```

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

```

2768         \tl_if_empty:NF \lst@label
2769         { \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.

```

2770         \__zrefclever_zcsetup:n { currentcounter = lstnumber }
2771     }
2772     \msg_info:nnn { zref-clever } { compat-package } { listings }
2773 }
2774 {}
2775 }

```

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 $\langle\max\text{-depth}\rangle$. `\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.

```

2776 \AddToHook { begindocument }
2777 {
2778   \@ifpackageloaded { enumitem }
2779   {
2780     \int_set:Nn \l_tmpa_int { 5 }
2781     \bool_while_do:nn
2782     {
2783       \cs_if_exist_p:c
2784       { c@ enum \int_to_roman:n { \l_tmpa_int } }
2785     }
2786     {
2787       \__zrefclever_zcsetup:x
2788       {
2789         counterresetby =
2790         {
2791           enum \int_to_roman:n { \l_tmpa_int } =
2792           enum \int_to_roman:n { \l_tmpa_int - 1 }
2793         } ,
2794         countertype =
2795         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
2796       }
2797       \int_incr:N \l_tmpa_int
2798     }
2799     \int_compare:nNnT { \l_tmpa_int } > { 5 }
2800     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
2801   }
2802   {}
2803 }
2804 \</package>

```

10 Dictionaries

10.1 English

```

2805 <package>\zcDeclareLanguage { english }
2806 <package>\zcDeclareLanguageAlias { american } { english }
2807 <package>\zcDeclareLanguageAlias { australian } { english }
2808 <package>\zcDeclareLanguageAlias { british } { english }
2809 <package>\zcDeclareLanguageAlias { canadian } { english }
2810 <package>\zcDeclareLanguageAlias { newzealand } { english }
2811 <package>\zcDeclareLanguageAlias { UKenglish } { english }

```

```

2812 <package>\zcDeclareLanguageAlias { USenglish } { english }
2813 <*dict-english>

2814 namesep    = {\nobreakspace} ,
2815 pairsep     = {\~and\nobreakspace} ,
2816 listsep     = {\~,~} ,
2817 lastsep     = {\~and\nobreakspace} ,
2818 tpairsep    = {\~and\nobreakspace} ,
2819 tlistsep    = {\~,~} ,
2820 tlastsep    = {\~,~and\nobreakspace} ,
2821 notesep     = {\~} ,
2822 rangesep    = {\~to\nobreakspace} ,
2823
2824 type = part ,
2825   Name-sg = Part ,
2826   name-sg = part ,
2827   Name-pl = Parts ,
2828   name-pl = parts ,
2829
2830 type = chapter ,
2831   Name-sg = Chapter ,
2832   name-sg = chapter ,
2833   Name-pl = Chapters ,
2834   name-pl = chapters ,
2835
2836 type = section ,
2837   Name-sg = Section ,
2838   name-sg = section ,
2839   Name-pl = Sections ,
2840   name-pl = sections ,
2841
2842 type = paragraph ,
2843   Name-sg = Paragraph ,
2844   name-sg = paragraph ,
2845   Name-pl = Paragraphs ,
2846   name-pl = paragraphs ,
2847   Name-sg-ab = Par. ,
2848   name-sg-ab = par. ,
2849   Name-pl-ab = Par. ,
2850   name-pl-ab = par. ,
2851
2852 type = appendix ,
2853   Name-sg = Appendix ,
2854   name-sg = appendix ,
2855   Name-pl = Appendices ,
2856   name-pl = appendices ,
2857
2858 type = subappendix ,
2859   Name-sg = Appendix ,
2860   name-sg = appendix ,
2861   Name-pl = Appendices ,
2862   name-pl = appendices ,
2863
2864 type = page ,

```

```

2865 Name-sg = Page ,
2866 name-sg = page ,
2867 Name-pl = Pages ,
2868 name-pl = pages ,
2869 name-sg-ab = p. ,
2870 name-pl-ab = pp. ,
2871
2872 type = line ,
2873 Name-sg = Line ,
2874 name-sg = line ,
2875 Name-pl = Lines ,
2876 name-pl = lines ,
2877
2878 type = figure ,
2879 Name-sg = Figure ,
2880 name-sg = figure ,
2881 Name-pl = Figures ,
2882 name-pl = figures ,
2883 Name-sg-ab = Fig. ,
2884 name-sg-ab = fig. ,
2885 Name-pl-ab = Figs. ,
2886 name-pl-ab = figs. ,
2887
2888 type = table ,
2889 Name-sg = Table ,
2890 name-sg = table ,
2891 Name-pl = Tables ,
2892 name-pl = tables ,
2893
2894 type = item ,
2895 Name-sg = Item ,
2896 name-sg = item ,
2897 Name-pl = Items ,
2898 name-pl = items ,
2899
2900 type = footnote ,
2901 Name-sg = Footnote ,
2902 name-sg = footnote ,
2903 Name-pl = Footnotes ,
2904 name-pl = footnotes ,
2905
2906 type = note ,
2907 Name-sg = Note ,
2908 name-sg = note ,
2909 Name-pl = Notes ,
2910 name-pl = notes ,
2911
2912 type = equation ,
2913 Name-sg = Equation ,
2914 name-sg = equation ,
2915 Name-pl = Equations ,
2916 name-pl = equations ,
2917 Name-sg-ab = Eq. ,
2918 name-sg-ab = eq. ,

```

```

2919 Name-pl-ab = Eqs. ,
2920 name-pl-ab = eqs. ,
2921 refpre-in = {} ,
2922 refpos-in = {} ,
2923
2924 type = theorem ,
2925 Name-sg = Theorem ,
2926 name-sg = theorem ,
2927 Name-pl = Theorems ,
2928 name-pl = theorems ,
2929
2930 type = lemma ,
2931 Name-sg = Lemma ,
2932 name-sg = lemma ,
2933 Name-pl = Lemmas ,
2934 name-pl = lemmas ,
2935
2936 type = corollary ,
2937 Name-sg = Corollary ,
2938 name-sg = corollary ,
2939 Name-pl = Corollaries ,
2940 name-pl = corollaries ,
2941
2942 type = proposition ,
2943 Name-sg = Proposition ,
2944 name-sg = proposition ,
2945 Name-pl = Propositions ,
2946 name-pl = propositions ,
2947
2948 type = definition ,
2949 Name-sg = Definition ,
2950 name-sg = definition ,
2951 Name-pl = Definitions ,
2952 name-pl = definitions ,
2953
2954 type = proof ,
2955 Name-sg = Proof ,
2956 name-sg = proof ,
2957 Name-pl = Proofs ,
2958 name-pl = proofs ,
2959
2960 type = result ,
2961 Name-sg = Result ,
2962 name-sg = result ,
2963 Name-pl = Results ,
2964 name-pl = results ,
2965
2966 type = remark ,
2967 Name-sg = Remark ,
2968 name-sg = remark ,
2969 Name-pl = Remarks ,
2970 name-pl = remarks ,
2971
2972 type = example ,

```

```

2973   Name-sg = Example ,
2974   name-sg = example ,
2975   Name-pl = Examples ,
2976   name-pl = examples ,
2977
2978   type = algorithm ,
2979   Name-sg = Algorithm ,
2980   name-sg = algorithm ,
2981   Name-pl = Algorithms ,
2982   name-pl = algorithms ,
2983
2984   type = listing ,
2985   Name-sg = Listing ,
2986   name-sg = listing ,
2987   Name-pl = Listings ,
2988   name-pl = listings ,
2989
2990   type = exercise ,
2991   Name-sg = Exercise ,
2992   name-sg = exercise ,
2993   Name-pl = Exercises ,
2994   name-pl = exercises ,
2995
2996   type = solution ,
2997   Name-sg = Solution ,
2998   name-sg = solution ,
2999   Name-pl = Solutions ,
3000   name-pl = solutions ,
3001 </dict-english>

```

10.2 German

```

3002 <package>\zcDeclareLanguage { german }
3003 <package>\zcDeclareLanguageAlias { austrian      } { german }
3004 <package>\zcDeclareLanguageAlias { germanb       } { german }
3005 <package>\zcDeclareLanguageAlias { ngerman       } { german }
3006 <package>\zcDeclareLanguageAlias { naustrian     } { german }
3007 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
3008 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
3009 <*dict-german>
3010 namesep = {\nobreakspace} ,
3011 pairsep  = {\~und\nobreakspace} ,
3012 listsep  = {,~} ,
3013 lastsep  = {\~und\nobreakspace} ,
3014 tpairsep = {\~und\nobreakspace} ,
3015 tlistsep = {,~} ,
3016 tlastsep = {\~und\nobreakspace} ,
3017 notesep  = {\~} ,
3018 rangesep = {\~bis\nobreakspace} ,
3019
3020 type = part ,
3021   Name-sg = Teil ,
3022   name-sg = Teil ,
3023   Name-pl = Teile ,

```



```

3024     name-pl = Teile ,
3025
3026 type = chapter ,
3027     Name-sg = Kapitel ,
3028     name-sg = Kapitel ,
3029     Name-pl = Kapitel ,
3030     name-pl = Kapitel ,
3031
3032 type = section ,
3033     Name-sg = Abschnitt ,
3034     name-sg = Abschnitt ,
3035     Name-pl = Abschnitte ,
3036     name-pl = Abschnitte ,
3037
3038 type = paragraph ,
3039     Name-sg = Absatz ,
3040     name-sg = Absatz ,
3041     Name-pl = Absätze ,
3042     name-pl = Absätze ,
3043
3044 type = appendix ,
3045     Name-sg = Anhang ,
3046     name-sg = Anhang ,
3047     Name-pl = Anhänge ,
3048     name-pl = Anhänge ,
3049
3050 type = subappendix ,
3051     Name-sg = Anhang ,
3052     name-sg = Anhang ,
3053     Name-pl = Anhänge ,
3054     name-pl = Anhänge ,
3055
3056 type = page ,
3057     Name-sg = Seite ,
3058     name-sg = Seite ,
3059     Name-pl = Seiten ,
3060     name-pl = Seiten ,
3061
3062 type = line ,
3063     Name-sg = Zeile ,
3064     name-sg = Zeile ,
3065     Name-pl = Zeilen ,
3066     name-pl = Zeilen ,
3067
3068 type = figure ,
3069     Name-sg = Abbildung ,
3070     name-sg = Abbildung ,
3071     Name-pl = Abbildungen ,
3072     name-pl = Abbildungen ,
3073     Name-sg-ab = Abb. ,
3074     name-sg-ab = Abb. ,
3075     Name-pl-ab = Abb. ,
3076     name-pl-ab = Abb. ,
3077

```

```

3078 type = table ,
3079     Name-sg = Tabelle ,
3080     name-sg = Tabelle ,
3081     Name-pl = Tabellen ,
3082     name-pl = Tabellen ,
3083
3084 type = item ,
3085     Name-sg = Punkt ,
3086     name-sg = Punkt ,
3087     Name-pl = Punkte ,
3088     name-pl = Punkte ,
3089
3090 type = footnote ,
3091     Name-sg = Fußnote ,
3092     name-sg = Fußnote ,
3093     Name-pl = Fußnoten ,
3094     name-pl = Fußnoten ,
3095
3096 type = note ,
3097     Name-sg = Anmerkung ,
3098     name-sg = Anmerkung ,
3099     Name-pl = Anmerkungen ,
3100     name-pl = Anmerkungen ,
3101
3102 type = equation ,
3103     Name-sg = Gleichung ,
3104     name-sg = Gleichung ,
3105     Name-pl = Gleichungen ,
3106     name-pl = Gleichungen ,
3107     refpre-in = {() ,
3108     refpos-in = {)} ,
3109
3110 type = theorem ,
3111     Name-sg = Theorem ,
3112     name-sg = Theorem ,
3113     Name-pl = Theoreme ,
3114     name-pl = Theoreme ,
3115
3116 type = lemma ,
3117     Name-sg = Lemma ,
3118     name-sg = Lemma ,
3119     Name-pl = Lemmata ,
3120     name-pl = Lemmata ,
3121
3122 type = corollary ,
3123     Name-sg = Korollar ,
3124     name-sg = Korollar ,
3125     Name-pl = Korollare ,
3126     name-pl = Korollare ,
3127
3128 type = proposition ,
3129     Name-sg = Satz ,
3130     name-sg = Satz ,
3131     Name-pl = Sätze ,

```

```

3132     name-pl = Sätze ,
3133
3134 type = definition ,
3135     Name-sg = Definition ,
3136     name-sg = Definition ,
3137     Name-pl = Definitionen ,
3138     name-pl = Definitionen ,
3139
3140 type = proof ,
3141     Name-sg = Beweis ,
3142     name-sg = Beweis ,
3143     Name-pl = Beweise ,
3144     name-pl = Beweise ,
3145
3146 type = result ,
3147     Name-sg = Ergebnis ,
3148     name-sg = Ergebnis ,
3149     Name-pl = Ergebnisse ,
3150     name-pl = Ergebnisse ,
3151
3152 type = remark ,
3153     Name-sg = Bemerkung ,
3154     name-sg = Bemerkung ,
3155     Name-pl = Bemerkungen ,
3156     name-pl = Bemerkungen ,
3157
3158 type = example ,
3159     Name-sg = Beispiel ,
3160     name-sg = Beispiel ,
3161     Name-pl = Beispiele ,
3162     name-pl = Beispiele ,
3163
3164 type = algorithm ,
3165     Name-sg = Algorithmus ,
3166     name-sg = Algorithmus ,
3167     Name-pl = Algorithmen ,
3168     name-pl = Algorithmen ,
3169
3170 type = listing ,
3171     Name-sg = Listing ,
3172     name-sg = Listing ,
3173     Name-pl = Listings ,
3174     name-pl = Listings ,
3175
3176 type = exercise ,
3177     Name-sg = Übungsaufgabe ,
3178     name-sg = Übungsaufgabe ,
3179     Name-pl = Übungsaufgaben ,
3180     name-pl = Übungsaufgaben ,
3181
3182 type = solution ,
3183     Name-sg = Lösung ,
3184     name-sg = Lösung ,
3185     Name-pl = Lösungen ,

```

```

3186   name-pl = Lösungen ,
3187 </dict-german>

```

10.3 French

```

3188 <package>\zcDeclareLanguage { french }
3189 <package>\zcDeclareLanguageAlias { acadian } { french }
3190 <package>\zcDeclareLanguageAlias { canadien } { french }
3191 <package>\zcDeclareLanguageAlias { francais } { french }
3192 <package>\zcDeclareLanguageAlias { frenchb } { french }
3193 <*dict-french>

3194 namesep = {\nobreakspace} ,
3195 pairsep = {\~et\nobreakspace} ,
3196 listsep = {,~} ,
3197 lastsep = {\~et\nobreakspace} ,
3198 tpairsep = {\~et\nobreakspace} ,
3199 tlistsep = {,~} ,
3200 tlastsep = {\~et\nobreakspace} ,
3201 notesep = {~} ,
3202 rangesep = {\~à\nobreakspace} ,
3203
3204 type = part ,
3205   Name-sg = Partie ,
3206   name-sg = partie ,
3207   Name-pl = Parties ,
3208   name-pl = parties ,
3209
3210 type = chapter ,
3211   Name-sg = Chapitre ,
3212   name-sg = chapitre ,
3213   Name-pl = Chapitres ,
3214   name-pl = chapitres ,
3215
3216 type = section ,
3217   Name-sg = Section ,
3218   name-sg = section ,
3219   Name-pl = Sections ,
3220   name-pl = sections ,
3221
3222 type = paragraph ,
3223   Name-sg = Paragraphe ,
3224   name-sg = paragraphe ,
3225   Name-pl = Paragraphes ,
3226   name-pl = paragraphes ,
3227
3228 type = appendix ,
3229   Name-sg = Annexe ,
3230   name-sg = annexe ,
3231   Name-pl = Annexes ,
3232   name-pl = annexes ,
3233
3234 type = subappendix ,
3235   Name-sg = Annexe ,
3236   name-sg = annexe ,

```

```

3237     Name-pl = Annexes ,
3238     name-pl = annexes ,
3239
3240 type = page ,
3241     Name-sg = Page ,
3242     name-sg = page ,
3243     Name-pl = Pages ,
3244     name-pl = pages ,
3245
3246 type = line ,
3247     Name-sg = Ligne ,
3248     name-sg = ligne ,
3249     Name-pl = Lignes ,
3250     name-pl = lignes ,
3251
3252 type = figure ,
3253     Name-sg = Figure ,
3254     name-sg = figure ,
3255     Name-pl = Figures ,
3256     name-pl = figures ,
3257
3258 type = table ,
3259     Name-sg = Table ,
3260     name-sg = table ,
3261     Name-pl = Tables ,
3262     name-pl = tables ,
3263
3264 type = item ,
3265     Name-sg = Point ,
3266     name-sg = point ,
3267     Name-pl = Points ,
3268     name-pl = points ,
3269
3270 type = footnote ,
3271     Name-sg = Note ,
3272     name-sg = note ,
3273     Name-pl = Notes ,
3274     name-pl = notes ,
3275
3276 type = note ,
3277     Name-sg = Note ,
3278     name-sg = note ,
3279     Name-pl = Notes ,
3280     name-pl = notes ,
3281
3282 type = equation ,
3283     Name-sg = Équation ,
3284     name-sg = équation ,
3285     Name-pl = Équations ,
3286     name-pl = équations ,
3287     refpre-in = { ( } ,
3288     refpos-in = { ) } ,
3289
3290 type = theorem ,

```

```

3291 Name-sg = Théorème ,
3292 name-sg = théorème ,
3293 Name-pl = Théorèmes ,
3294 name-pl = théorèmes ,
3295
3296 type = lemma ,
3297 Name-sg = Lemme ,
3298 name-sg = lemme ,
3299 Name-pl = Lemmes ,
3300 name-pl = lemmes ,
3301
3302 type = corollary ,
3303 Name-sg = Corollaire ,
3304 name-sg = corollaire ,
3305 Name-pl = Corollaires ,
3306 name-pl = corollaires ,
3307
3308 type = proposition ,
3309 Name-sg = Proposition ,
3310 name-sg = proposition ,
3311 Name-pl = Propositions ,
3312 name-pl = propositions ,
3313
3314 type = definition ,
3315 Name-sg = Définition ,
3316 name-sg = définition ,
3317 Name-pl = Définitions ,
3318 name-pl = définitions ,
3319
3320 type = proof ,
3321 Name-sg = Démonstration ,
3322 name-sg = démonstration ,
3323 Name-pl = Démonstrations ,
3324 name-pl = démonstrations ,
3325
3326 type = result ,
3327 Name-sg = Résultat ,
3328 name-sg = résultat ,
3329 Name-pl = Résultats ,
3330 name-pl = résultats ,
3331
3332 type = remark ,
3333 Name-sg = Remarque ,
3334 name-sg = remarque ,
3335 Name-pl = Remarques ,
3336 name-pl = remarques ,
3337
3338 type = example ,
3339 Name-sg = Exemple ,
3340 name-sg = exemple ,
3341 Name-pl = Exemples ,
3342 name-pl = exemples ,
3343
3344 type = algorithm ,

```

```

3345   Name-sg = Algorithme ,
3346   name-sg = algorithme ,
3347   Name-pl = Algorithmes ,
3348   name-pl = algorithmes ,
3349
3350   type = listing ,
3351   Name-sg = Liste ,
3352   name-sg = liste ,
3353   Name-pl = Listes ,
3354   name-pl = listes ,
3355
3356   type = exercise ,
3357   Name-sg = Exercice ,
3358   name-sg = exercice ,
3359   Name-pl = Exercices ,
3360   name-pl = exercices ,
3361
3362   type = solution ,
3363   Name-sg = Solution ,
3364   name-sg = solution ,
3365   Name-pl = Solutions ,
3366   name-pl = solutions ,
3367 </dict-french>

```

10.4 Portuguese

```

3368 <package>\zcDeclareLanguage { portuguese }
3369 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3370 <package>\zcDeclareLanguageAlias { brazil    } { portuguese }
3371 <package>\zcDeclareLanguageAlias { portuges  } { portuguese }
3372 <*dict-portuguese>
3373 namesep = {\nobreakspace} ,
3374 pairsep  = {\nobreakspace} ,
3375 listsep  = {,~} ,
3376 lastsep  = {\nobreakspace} ,
3377 tpairsep = {\nobreakspace} ,
3378 tlistsep = {,~} ,
3379 tlastsep = {\nobreakspace} ,
3380 notesep  = {~} ,
3381 rangesep = {\nobreakspace} ,
3382
3383 type = part ,
3384   Name-sg = Parte ,
3385   name-sg = parte ,
3386   Name-pl = Partes ,
3387   name-pl = partes ,
3388
3389 type = chapter ,
3390   Name-sg = Capítulo ,
3391   name-sg = capítulo ,
3392   Name-pl = Capítulos ,
3393   name-pl = capítulos ,
3394
3395 type = section ,

```

```

3396     Name-sg = Seção ,
3397     name-sg = seção ,
3398     Name-pl = Seções ,
3399     name-pl = seções ,
3400
3401     type = paragraph ,
3402     Name-sg = Parágrafo ,
3403     name-sg = parágrafo ,
3404     Name-pl = Parágrafos ,
3405     name-pl = parágrafos ,
3406     Name-sg-ab = Par. ,
3407     name-sg-ab = par. ,
3408     Name-pl-ab = Par. ,
3409     name-pl-ab = par. ,
3410
3411     type = appendix ,
3412     Name-sg = Apêndice ,
3413     name-sg = apêndice ,
3414     Name-pl = Apêndices ,
3415     name-pl = apêndices ,
3416
3417     type = subappendix ,
3418     Name-sg = Apêndice ,
3419     name-sg = apêndice ,
3420     Name-pl = Apêndices ,
3421     name-pl = apêndices ,
3422
3423     type = page ,
3424     Name-sg = Página ,
3425     name-sg = página ,
3426     Name-pl = Páginas ,
3427     name-pl = páginas ,
3428     name-sg-ab = p. ,
3429     name-pl-ab = pp. ,
3430
3431     type = line ,
3432     Name-sg = Linha ,
3433     name-sg = linha ,
3434     Name-pl = Linhas ,
3435     name-pl = linhas ,
3436
3437     type = figure ,
3438     Name-sg = Figura ,
3439     name-sg = figura ,
3440     Name-pl = Figuras ,
3441     name-pl = figuras ,
3442     Name-sg-ab = Fig. ,
3443     name-sg-ab = fig. ,
3444     Name-pl-ab = Figs. ,
3445     name-pl-ab = figs. ,
3446
3447     type = table ,
3448     Name-sg = Tabela ,
3449     name-sg = tabela ,

```



```

3450     Name-pl = Tabelas ,
3451     name-pl = tabelas ,
3452
3453     type = item ,
3454     Name-sg = Item ,
3455     name-sg = item ,
3456     Name-pl = Itens ,
3457     name-pl = itens ,
3458
3459     type = footnote ,
3460     Name-sg = Nota ,
3461     name-sg = nota ,
3462     Name-pl = Notas ,
3463     name-pl = notas ,
3464
3465     type = note ,
3466     Name-sg = Nota ,
3467     name-sg = nota ,
3468     Name-pl = Notas ,
3469     name-pl = notas ,
3470
3471     type = equation ,
3472     Name-sg = Equação ,
3473     name-sg = equação ,
3474     Name-pl = Equações ,
3475     name-pl = equações ,
3476     Name-sg-ab = Eq. ,
3477     name-sg-ab = eq. ,
3478     Name-pl-ab = Eqs. ,
3479     name-pl-ab = eqs. ,
3480     refpre-in = {() ,
3481     refpos-in = {)} ,
3482
3483     type = theorem ,
3484     Name-sg = Teorema ,
3485     name-sg = teorema ,
3486     Name-pl = Teoremas ,
3487     name-pl = teoremas ,
3488
3489     type = lemma ,
3490     Name-sg = Lema ,
3491     name-sg = lema ,
3492     Name-pl = Lemas ,
3493     name-pl = lemas ,
3494
3495     type = corollary ,
3496     Name-sg = Corolário ,
3497     name-sg = corolário ,
3498     Name-pl = Corolários ,
3499     name-pl = corolários ,
3500
3501     type = proposition ,
3502     Name-sg = Proposição ,
3503     name-sg = proposição ,

```

```

3504     Name-pl = Proposições ,
3505     name-pl = proposições ,
3506
3507 type = definition ,
3508     Name-sg = Definição ,
3509     name-sg = definição ,
3510     Name-pl = Definições ,
3511     name-pl = definições ,
3512
3513 type = proof ,
3514     Name-sg = Demonstração ,
3515     name-sg = demonstração ,
3516     Name-pl = Demonstrações ,
3517     name-pl = demonstrações ,
3518
3519 type = result ,
3520     Name-sg = Resultado ,
3521     name-sg = resultado ,
3522     Name-pl = Resultados ,
3523     name-pl = resultados ,
3524
3525 type = remark ,
3526     Name-sg = Observação ,
3527     name-sg = observação ,
3528     Name-pl = Observações ,
3529     name-pl = observações ,
3530
3531 type = example ,
3532     Name-sg = Exemplo ,
3533     name-sg = exemplo ,
3534     Name-pl = Exemplos ,
3535     name-pl = exemplos ,
3536
3537 type = algorithm ,
3538     Name-sg = Algoritmo ,
3539     name-sg = algoritmo ,
3540     Name-pl = Algoritmos ,
3541     name-pl = algoritmos ,
3542
3543 type = listing ,
3544     Name-sg = Listagem ,
3545     name-sg = listagem ,
3546     Name-pl = Listagens ,
3547     name-pl = listagens ,
3548
3549 type = exercise ,
3550     Name-sg = Exercício ,
3551     name-sg = exercício ,
3552     Name-pl = Exercícios ,
3553     name-pl = exercícios ,
3554
3555 type = solution ,
3556     Name-sg = Solução ,
3557     name-sg = solução ,

```

```

3558   Name-pl = Soluções ,
3559   name-pl = soluções ,
3560 </dict-portuguese>

```

10.5 Spanish

```

3561 <package>\zcDeclareLanguage { spanish }
3562 <*dict-spanish>

3563 namesep = {\nobreakspace} ,
3564 pairsep = {\~y\nobreakspace} ,
3565 listsep = {,~} ,
3566 lastsep = {\~y\nobreakspace} ,
3567 tpairsep = {\~y\nobreakspace} ,
3568 tlistsep = {,~} ,
3569 tlastsep = {\~y\nobreakspace} ,
3570 notesep = {\~} ,
3571 rangesep = {\~a\nobreakspace} ,
3572
3573 type = part ,
3574   Name-sg = Parte ,
3575   name-sg = parte ,
3576   Name-pl = Partes ,
3577   name-pl = partes ,
3578
3579 type = chapter ,
3580   Name-sg = Capítulo ,
3581   name-sg = capítulo ,
3582   Name-pl = Capítulos ,
3583   name-pl = capítulos ,
3584
3585 type = section ,
3586   Name-sg = Sección ,
3587   name-sg = sección ,
3588   Name-pl = Secciones ,
3589   name-pl = secciones ,
3590
3591 type = paragraph ,
3592   Name-sg = Párrafo ,
3593   name-sg = párrafo ,
3594   Name-pl = Párrafos ,
3595   name-pl = párrafos ,
3596
3597 type = appendix ,
3598   Name-sg = Apéndice ,
3599   name-sg = apéndice ,
3600   Name-pl = Apéndices ,
3601   name-pl = apéndices ,
3602
3603 type = subappendix ,
3604   Name-sg = Apéndice ,
3605   name-sg = apéndice ,
3606   Name-pl = Apéndices ,
3607   name-pl = apéndices ,
3608

```

```

3609 type = page ,
3610     Name-sg = Página ,
3611     name-sg = página ,
3612     Name-pl = Páginas ,
3613     name-pl = páginas ,
3614
3615 type = line ,
3616     Name-sg = Línea ,
3617     name-sg = línea ,
3618     Name-pl = Líneas ,
3619     name-pl = líneas ,
3620
3621 type = figure ,
3622     Name-sg = Figura ,
3623     name-sg = figura ,
3624     Name-pl = Figuras ,
3625     name-pl = figuras ,
3626
3627 type = table ,
3628     Name-sg = Cuadro ,
3629     name-sg = cuadro ,
3630     Name-pl = Cuadros ,
3631     name-pl = cuadros ,
3632
3633 type = item ,
3634     Name-sg = Punto ,
3635     name-sg = punto ,
3636     Name-pl = Puntos ,
3637     name-pl = puntos ,
3638
3639 type = footnote ,
3640     Name-sg = Nota ,
3641     name-sg = nota ,
3642     Name-pl = Notas ,
3643     name-pl = notas ,
3644
3645 type = note ,
3646     Name-sg = Nota ,
3647     name-sg = nota ,
3648     Name-pl = Notas ,
3649     name-pl = notas ,
3650
3651 type = equation ,
3652     Name-sg = Ecuación ,
3653     name-sg = ecuación ,
3654     Name-pl = Ecuaciones ,
3655     name-pl = ecuaciones ,
3656     refpre-in = {()} ,
3657     refpos-in = {} } ,
3658
3659 type = theorem ,
3660     Name-sg = Teorema ,
3661     name-sg = teorema ,
3662     Name-pl = Teoremas ,

```

```

3663     name-pl = teoremas ,
3664
3665 type = lemma ,
3666     Name-sg = Lema ,
3667     name-sg = lema ,
3668     Name-pl = Lemas ,
3669     name-pl = lemas ,
3670
3671 type = corollary ,
3672     Name-sg = Corolario ,
3673     name-sg = corolario ,
3674     Name-pl = Corolarios ,
3675     name-pl = corolarios ,
3676
3677 type = proposition ,
3678     Name-sg = Proposición ,
3679     name-sg = proposición ,
3680     Name-pl = Proposiciones ,
3681     name-pl = proposiciones ,
3682
3683 type = definition ,
3684     Name-sg = Definición ,
3685     name-sg = definición ,
3686     Name-pl = Definiciones ,
3687     name-pl = definiciones ,
3688
3689 type = proof ,
3690     Name-sg = Demostración ,
3691     name-sg = demostración ,
3692     Name-pl = Demostraciones ,
3693     name-pl = demostraciones ,
3694
3695 type = result ,
3696     Name-sg = Resultado ,
3697     name-sg = resultado ,
3698     Name-pl = Resultados ,
3699     name-pl = resultados ,
3700
3701 type = remark ,
3702     Name-sg = Observación ,
3703     name-sg = observación ,
3704     Name-pl = Observaciones ,
3705     name-pl = observaciones ,
3706
3707 type = example ,
3708     Name-sg = Ejemplo ,
3709     name-sg = ejemplo ,
3710     Name-pl = Ejemplos ,
3711     name-pl = ejemplos ,
3712
3713 type = algorithm ,
3714     Name-sg = Algoritmo ,
3715     name-sg = algoritmo ,
3716     Name-pl = Algoritmos ,

```

```

3717   name-pl = algoritmos ,
3718
3719   type = listing ,
3720   Name-sg = Listado ,
3721   name-sg = listado ,
3722   Name-pl = Listados ,
3723   name-pl = listados ,
3724
3725   type = exercise ,
3726   Name-sg = Ejercicio ,
3727   name-sg = ejercicio ,
3728   Name-pl = Ejercicios ,
3729   name-pl = ejercicios ,
3730
3731   type = solution ,
3732   Name-sg = Solución ,
3733   name-sg = solución ,
3734   Name-pl = Soluciones ,
3735   name-pl = soluciones ,
3736 </dict-spanish>

```

Index

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

Symbols	
<code>\</code>	97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145
<code>†</code> internal commands:	
<code>\i_zrefclever_current_counter_tl</code>	4
A	
<code>\AddToHook</code>	85, 477, 492, 636, 672, 697, 735, 737, 789, 810, 2544, 2555, 2557, 2562, 2575, 2595, 2610, 2612, 2617, 2633, 2646, 2666, 2678, 2710, 2719, 2753, 2776
<code>\appendix</code>	69, 70
<code>\appendixname</code>	69
B	
<code>\babelname</code>	682
<code>\babelprovide</code>	12, 23
<code>\begin</code>	73
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_if:N</code>	307, 318, 640, 644, 1132, 1550, 1645, 1775, 1797, 1828, 1874, 1915, 1938, 1942, 1948, 1958, 1964, 2121
<code>\bool_if:nTF</code>	53, 1217, 1226, 1235, 1291, 1301, 1325, 1342, 1357, 1422, 1430, 1564, 1572, 1809, 1816, 1823, 2073, 2194
<code>\bool_lazy_all:nTF</code>	2466
<code>\bool_lazy_and:nnTF</code>	1106, 1124, 2268, 2523
<code>\bool_lazy_any:nTF</code>	2346, 2355
<code>\bool_lazy_or:nnTF</code>	1110, 2256
<code>\bool_new:N</code>	270, 513, 514, 539, 563, 572, 579, 580, 593, 594, 613, 614, 803, 804, 1140, 1153, 1462, 1463, 1473, 1479, 1480, 2718
<code>\bool_set:Nn</code>	1104
<code>\bool_set_false:N</code>	526, 530, 621, 630, 631, 646, 826, 1283, 1519, 1556, 1570, 1584, 1787, 1913, 1914, 2353, 2370
<code>\bool_set_true:N</code>	327, 520, 521, 525, 531, 620, 625, 626, 814, 819, 1297, 1307, 1311, 1333, 1348, 1363, 1387, 1527, 1551, 1557, 1561, 1588, 1594, 2369, 2405, 2412, 2413, 2431, 2438, 2450, 2725
<code>\bool_until_do:Nn</code>	1323, 1520
<code>\bool_while_do:nn</code>	2781

C		2201, 2202, 2204, 2216, 2220, 2223, 2228, 2229, 2231, 2232, 2236, 2238
clist commands:		\ExplSyntaxOn 12, 283
\clist_map_inline:nn 869, 2692		
\counterwithin 4		
cs commands:		F
\cs_generate_variant:Nn 50, 179, 185, 323, 331, 949, 1015, 1021, 1269, 2115, 2383		file commands:
\cs_if_exist:NTF ... 43, 63, 2583, 2585		\file_get:nnNTF 281
\cs_if_exist_p:N 2783		\fmtversion 3
\cs_new:Npn 41, 51, 61, 72, 180, 186, 2069, 2116, 2373		\footnote 69
\cs_new_nopar:Npn 2548		
\cs_new_protected:Npn 174, 271, 324, 332, 338, 459, 947, 1010, 1016, 1099, 1157, 1199, 1210, 1270, 1400, 1452, 1496, 1652, 1909, 2065, 2067, 2246, 2384, 2460, 2517, 2726		G
\cs_new_protected:Npx 84		group commands:
\cs_set_eq:NN 88, 2546, 2648, 2649, 2653, 2654, 2662, 2663, 2668, 2669, 2674, 2675		\group_begin: .. 87, 273, 326, 999, 1101, 1114, 1830, 1847, 2079, 2082, 2099, 2102, 2138, 2143, 2146, 2162, 2169, 2184, 2200, 2203, 2227, 2230
\cs_set_nopar:Npn 2637		\group_end: 90, 321, 329, 1007, 1117, 1137, 1844, 1850, 2094, 2096, 2108, 2110, 2141, 2153, 2160, 2166, 2172, 2187, 2222, 2224, 2237, 2239
E		
\endinput 12		I
exp commands:		\IfBooleanTF 1143
\exp_args:NNe 27, 30		\IfFormatAtLeastTF 3, 4, 2642
\exp_args:NNNo 176		\input 12
\exp_args:NNnx 261		int commands:
\exp_args:NNo 176, 182		\int_case:nnTF 1655, 1683, 1715, 1877, 1971, 2010
\exp_args:NnV 299		\int_compare:nNnTF 1257, 1334, 1349, 1364, 1376, 1388, 1408, 1410, 1454, 1616, 1670, 1704, 1866, 1868, 1926, 1951, 1995, 2401, 2407, 2427, 2433, 2799
\exp_args:Nnx 334		\int_compare_p:nNn 1424, 1432, 2260, 2271, 2366
\exp_args:No 182		\int_eval:n 84
\exp_args:Nx 281, 2730, 2738		\int_incr:N 1904, 1941, 1943, 1957, 1959, 1963, 1965, 2063, 2797
\exp_args:Nxx 1253, 2397, 2419, 2423, 2440		\int_new:N 1154, 1155, 1464, 1465, 1476, 1477
\exp_not:N 58, 1830, 1833, 1844, 1847, 1850, 2079, 2082, 2085, 2094, 2096, 2099, 2102, 2108, 2110, 2128, 2138, 2141, 2143, 2146, 2153, 2160, 2162, 2166, 2169, 2172, 2184, 2187, 2200, 2203, 2206, 2222, 2224, 2227, 2230, 2237, 2239		\int_set:Nn 1409, 1411, 1415, 1418, 2780
\exp_not:n 183, 1674, 1690, 1702, 1707, 1730, 1744, 1748, 1760, 1764, 1798, 1799, 1831, 1843, 1848, 1849, 1978, 1991, 1998, 2022, 2034, 2038, 2048, 2052, 2080, 2081, 2083, 2089, 2092, 2095, 2100, 2101, 2103, 2104, 2107, 2109, 2139, 2140, 2142, 2144, 2145, 2147, 2148, 2152, 2164, 2165, 2170, 2171, 2173, 2181, 2185, 2186, 2188,		\int_to_roman:n 2784, 2791, 2792, 2795, 2797, 2799
		\int_use:N 37, 39, 45
		\int_zero:N 1402, 1403, 1505, 1506, 1507, 1508, 1903, 1905, 1906, 2058, 2059
		\l_tmpa_int 2780, 2784, 2791, 2792, 2795, 2797, 2799
		iow commands:
		\iow_char:N 97, 103, 112, 113, 118, 119, 124, 125, 134, 135, 145
		K
		keys commands:
		\keys_define:nn

<p> 31, 344, 356, 373, 387, 466, 496, 503, 515, 540, 549, 564, 573, 581, 595, 607, 615, 648, 655, 693, 740, 782, 784, 791, 798, 805, 815, 827, 836, 865, 891, 915, 925, 936, 960, 972, 1022, 1034, 1055, 1078 \keys_set:nn 12, 31, 35, 300, 820, 948, 955, 1004, 1102 keyval commands: \keyval_parse:nnn 840, 895 </p> <p style="text-align: center;">L</p> <p> \label 69, 72, 2546 \labelformat 3 \language 23, 676 </p> <p style="text-align: center;">M</p> <p> \mainbabelname 23, 683 \MessageBreak 10 MH commands: \MH_if_boolean:nTF 2723 msg commands: \msg_info:nnn 364, 394, 2629, 2714, 2748, 2772, 2800 \msg_line_context: 96, 102, 106, 108, 111, 117, 123, 130, 133, 139, 144, 151, 156, 160, 162, 165, 169 \msg_new:nnn 94, 100, 105, 107, 109, 115, 121, 127, 129, 131, 137, 142, 147, 149, 154, 159, 161, 163, 168, 170, 172 \msg_note:nnn 303 \msg_warning:nn 482, 507, 645, 651, 794, 831 \msg_warning:nnn ... 250, 265, 309, 319, 723, 768, 897, 964, 1006, 1046, 1085, 1609, 1782, 2299, 2334, 2509 \msg_warning:nnnn 842 </p> <p style="text-align: center;">N</p> <p> \newcounter 4, 2579, 2580 \NewDocumentCommand 245, 255, 945, 950, 997, 1097, 1141 \nobreakspace 408, 2814, 2815, 2817, 2818, 2820, 2822, 3010, 3011, 3013, 3014, 3016, 3018, 3194, 3195, 3197, 3198, 3200, 3202, 3373, 3374, 3376, 3377, 3379, 3381, 3563, 3564, 3566, 3567, 3569, 3571 </p> <p style="text-align: center;">P</p> <p> \PackageError 7 \pagenumbering 6 \pageref 36 </p>	<p> prg commands: \prg_generate_conditional_- variant:Nnn 432, 448 \prg_new_protected_conditional:Npnn 418, 434, 451 \prg_return_false: 428, 430, 444, 446, 457 \prg_return_true: 427, 443, 456 \ProcessKeysOptions 944 prop commands: \prop_get:NnN 2486 \prop_get:NnNTF 274, 421, 424, 437, 440, 454, 1000, 2284, 2305, 2313, 2463, 2520, 2533 \prop_gput:Nnn .. 251, 262, 1012, 1018 \prop_gput_if_new:Nnn 334, 340 \prop_gset_from_keyval:Nn 402 \prop_if_exist:NTF 286, 952 \prop_if_exist_p:N 2470, 2526 \prop_if_in:NnTF 27, 249, 259, 718, 763 \prop_if_in_p:Nn 54, 2477 \prop_item:Nn 30, 55, 263 \prop_new:N 244, 292, 401, 835, 890, 921, 953 \prop_put:Nnn 463, 932, 987 \prop_remove:Nn 462, 931, 979 \providecommand 3 \ProvidesExplPackage 14 \ProvidesFile 12 </p> <p style="text-align: center;">R</p> <p> \refstepcounter 3, 69, 73, 75 \renewlist 75, 76 \RequirePackage 16, 17, 18, 19, 641, 786, 807 </p> <p style="text-align: center;">S</p> <p> \scantokens 70 seq commands: \seq_clear:N 560, 1159 \seq_const_from_clist:Nn 190, 198, 211, 223 \seq_gconcat:NNN ... 231, 234, 238, 241 \seq_get_left:NN 1530 \seq_gput_right:Nn 301, 312 \seq_if_empty:NTF 1524 \seq_if_in:NnTF 277, 871, 1203 \seq_map_break:n 75, 1443, 1446 \seq_map_function:NN 1162 \seq_map_indexed_inline:Nn . 20, 1404 \seq_map_inline:Nn 353, 370, 384, 922, 957, 969, 1031, 1052, 1075, 1440, 2728 \seq_map_tokens:Nn 57 </p>
---	--

<code>\seq_new:N</code>	230,	<code>\ltx@label</code>	72, 2648, 2649, 2653, 2654,
	237, 269, 548, 864, 1139, 1156, 1461		2662, 2663, 2668, 2669, 2674, 2675
<code>\seq_pop_left:NN</code>	1522	<code>\MT@newlabel</code>	2735, 2743
<code>\seq_put_right:Nn</code>	873, 1206	<code>\p@...</code>	3
<code>\seq_reverse:N</code>	554	<code>\protected@write</code>	2734, 2742
<code>\seq_set_eq:NN</code>	1498	<code>\zref@addprop</code>	21, 24, 35, 38, 40, 82, 93
<code>\seq_set_from_clist:Nn</code>	553, 1103	<code>\zref@default</code>	58, 2066, 2068
<code>\seq_sort:Nn</code>	38, 1165	<code>\zref@extractdefault</code>	
<code>\setcounter</code> ..	2581, 2582, 2598, 2611, 2615		8, 9, 64, 177, 183, 187
sort commands:		<code>\zref@ifpropundefined</code>	18, 2375
<code>\sort_return_same:</code>		<code>\zref@ifrefcontainsprop</code>	
	39, 44, 1172, 1177,		18, 2071, 2123, 2190, 2378
	1224, 1262, 1264, 1298, 1318, 1339,	<code>\zref@ifrefundefined</code>	
	1354, 1368, 1393, 1428, 1443, 1459		1167, 1169, 1181, 1553, 1555, 1560,
<code>\sort_return_swapped:</code> ...	39, 44,		1604, 1779, 1788, 1917, 2118, 2248
	1185, 1233, 1261, 1308, 1317, 1338,	<code>\zref@label</code>	72, 2640
	1353, 1369, 1392, 1436, 1446, 1458	<code>\ZREF@mainlist</code>	21, 24, 35, 38, 40, 82, 93
<code>\stepcounter</code>	2597, 2614	<code>\zref@newprop</code>	
str commands:			5, 6, 20, 22, 25, 36, 39, 77, 92
<code>\str_case:nnTF</code>	699, 744	<code>\zref@refused</code>	1603
<code>\str_compare:nNnTF</code>	1314	<code>\zref@wrapper@babel</code>	35, 72, 1098, 2640
<code>\str_if_eq:nnTF</code>	74	<code>\textendash</code>	412
<code>\str_if_eq_p:nn</code>	2351, 2357, 2359, 2363	<code>\textup</code>	74
<code>\str_new:N</code>	654	<code>\the</code>	3
<code>\str_set:Nn</code>	659, 661, 663, 665	<code>\thechapter</code>	69
<code>\string</code>	2735, 2743	<code>\thelstnumber</code>	75
T			
<code>\tag</code>	73	<code>\thepage</code>	6, 89
T _E X and L ^A T _E X 2 _ε commands:		<code>\thesection</code>	69
<code>\@Alpha</code>	69	tl commands:	
<code>\@addtoreset</code>	4	<code>\c_empty_tl</code>	1202, 1213,
<code>\@auxout</code>	2734, 2742		1215, 1273, 1276, 1279, 1281, 1541,
<code>\@chapapp</code>	69		1543, 2376, 2379, 2380, 2387, 2389
<code>\@currentcounter</code>	3, 4, 29, 69, 73, 75, 919	<code>\c_novalue_tl</code>	927, 974
<code>\@currentlabel</code>	3, 73, 75	<code>\tl_clear:N</code>	
<code>\@elt</code>	4		298, 349, 1003, 1027, 1500,
<code>\@ifl@t@r</code>	3		1501, 1502, 1503, 1504, 1526, 1899,
<code>\@ifpackageloaded</code>	479,		1900, 1901, 1902, 1940, 2249, 2252,
	494, 638, 674, 680, 812, 2577, 2635,		2280, 2298, 2333, 2508, 2539, 2541
	2644, 2658, 2660, 2721, 2755, 2778	<code>\tl_gset:Nn</code>	89
<code>\@onlypreamble</code>	254, 268, 1009	<code>\tl_head:N</code>	
<code>\bbl@loaded</code>	23		1352, 1365, 1377, 1379, 1389, 1391
<code>\bbl@main@language</code>	23, 677	<code>\tl_if_empty:NTF</code>	65, 361,
<code>\c@</code>	4		378, 392, 1039, 1060, 1083, 1118,
<code>\c@enumN</code>	76		1607, 1777, 2178, 2265, 2282, 2768
<code>\c@lstnumber</code>	75	<code>\tl_if_empty:nTF</code>	247,
<code>\c@page</code>	6, 88		257, 348, 461, 1026, 1726, 1742,
<code>\cl@</code>	4, 5		1758, 1989, 2020, 2032, 2046, 2251
<code>\hyper@link</code>	58, 1833, 2085, 2128, 2206	<code>\tl_if_empty_p:N</code>	1221, 1222, 1230,
<code>\lst@AddToHook</code>	2766		1231, 1238, 1239, 1567, 1568, 1575,
<code>\lst@label</code>	2768, 2769		1577, 2350, 2360, 2364, 2468, 2524
<code>\ltx@gobble</code>	72	<code>\tl_if_empty_p:n</code>	1293, 1294,
			1303, 1304, 1329, 1330, 1345, 1360
		<code>\tl_if_eq:NNTF</code>	1242, 1287, 1580, 2391

<code>\tl_if_eq:NnTF</code>	1160, 1192, 1414, 1417, 1442, 1445, 1534, 2395
<code>\tl_if_eq:nnTF</code>	1253, 1406, 2397, 2419, 2423, 2440, 2730, 2738
<code>\tl_if_novalue:nTF</code>	930, 977
<code>\tl_map_break:n</code>	75
<code>\tl_map_tokens:Nn</code>	67
<code>\tl_new:N</code>	83, 188, 189, 465, 669, 670, 671, 781, 797, 914, 1147, 1148, 1149, 1150, 1151, 1152, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1474, 1475, 1478, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 2553
<code>\tl_put_left:Nn</code>	1812, 1819, 1859
<code>\tl_put_right:Nn</code>	1672, 1688, 1697, 1728, 1739, 1755, 1976, 1987, 2018, 2030, 2044, 2266, 2267, 2278
<code>\tl_reverse:N</code>	1274, 1277
<code>\tl_reverse_items:n</code>	1269
<code>\tl_set:Nn</code>	176, 350, 470, 472, 474, 480, 483, 499, 508, 676, 677, 682, 683, 686, 687, 690, 703, 711, 720, 725, 748, 756, 765, 770, 954, 1028, 1381, 1383, 1536, 1537, 1661, 1663, 1795, 1826, 1930, 1932, 1955, 2262, 2263, 2276, 2554, 2556
<code>\tl_set_eq:NN</code>	1897
<code>\tl_tail:N</code>	1382, 1384
<code>\l_tmpa_tl</code>	284, 300, 1120, 1121
U	
<code>\upshape</code>	2713
use commands:	
<code>\use:N</code>	23
V	
<code>\value</code>	2598, 2615
Z	
<code>\zcDeclareLanguage</code>	11, 245, 2805, 3002, 3188, 3368, 3561
<code>\zcDeclareLanguageAlias</code>	11, 255, 2806, 2807, 2808, 2809, 2810, 2811, 2812, 3003, 3004, 3005, 3006, 3007, 3008, 3189, 3190, 3191, 3192, 3369, 3370, 3371
<code>\zcLanguageSetup</code>	9, 12, 14, 31–33, 997
<code>\zcpageref</code>	36, 1141
<code>\zcref</code>	25, 26, 29, 30, 34–38, 45, 47, 74, 1097, 1144, 1145
<code>\zcRefTypeSetup</code>	9, 31, 950, 2713
<code>\zcsetup</code>	23, 26, 29, 30, 945
<code>\zlabel</code>	69, 72, 73, 75, 2551, 2769
zrefcheck commands:	
<code>\zrefcheck_zcref_beg_label:</code>	1109
<code>\zrefcheck_zcref_end_label_-</code> maybe:	1128
<code>\zrefcheck_zcref_run_checks_on_-</code> labels:n	1129
zrefclever internal commands:	
<code>\l__zrefclever_abbrev_bool</code>	593, 597, 2269
<code>\l__zrefclever_capitalize_bool</code>	579, 583, 2257
<code>\l__zrefclever_capitalize_first_-</code> bool	580, 589, 2259
<code>__zrefclever_counter_reset_by:n</code>	5, 28, 43, 45, 47, 51, 2685
<code>__zrefclever_counter_reset_by_-</code> aux:nn	58, 61
<code>__zrefclever_counter_reset_by_-</code> aux:nnn	68, 72
<code>\l__zrefclever_counter_resetby_-</code> prop	5, 28, 54, 55, 890, 902
<code>\l__zrefclever_counter_resettters_-</code> seq	4, 5, 28, 57, 864, 871, 874
<code>\l__zrefclever_counter_type_prop</code>	3, 27, 27, 30, 835, 847
<code>\l__zrefclever_current_counter_-</code> tl	3, 29, 20, 23, 28, 31, 33, 37, 80, 914, 917
<code>\l__zrefclever_current_language_-</code> tl	23, 671, 676, 682, 686, 712, 757
<code>__zrefclever_declare_default_-</code> transl:nnn	33, 1010, 1041, 1062
<code>__zrefclever_declare_type_-</code> transl:nnnn	33, 1010, 1067, 1089
<code>__zrefclever_def_extract:Nnnn</code>	8, 174, 1201, 1212, 1214, 1272, 1275, 1278, 1280, 1540, 1542, 2386, 2388
<code>\g__zrefclever_dict_⟨language⟩_prop</code>	12
<code>\l__zrefclever_dict_language_tl</code>	188, 275, 279, 282, 289, 295, 302, 304, 310, 313, 335, 341, 422, 425, 438, 441, 1001, 1042, 1063, 1068, 1090
<code>__zrefclever_extract:nnn</code>	9, 186, 1258, 1260, 1335, 1337, 1350, 1367, 1455, 1457, 2402, 2404, 2408, 2410, 2428, 2430, 2434, 2436
<code>__zrefclever_extract_unexp:nnn</code>	9, 64, 180, 1254, 1255, 1839, 2087, 2090, 2105, 2134, 2149, 2212, 2217, 2233, 2376, 2379, 2380, 2398, 2399, 2420, 2421, 2424, 2425, 2442, 2446, 2731, 2739

`__zrefclever_extract_url_-`
`unexp:n` 1835, 2086, 2130, 2208, [2373](#)
`\g__zrefclever_fallback_dict_-`
`prop` 9, 401, 402, 454
`\l__zrefclever_footnote_type_tl` .
..... 2553, 2554, 2556, 2560
`__zrefclever_get_default_-`
`transl:nnN` 9, 435, 449
`__zrefclever_get_default_-`
`transl:nnNTF` 17, [434](#), 2501
`__zrefclever_get_enclosing_-`
`counters_value:n` ... 5, [41](#), 46, 79
`__zrefclever_get_fallback_-`
`transl:nN` 452
`__zrefclever_get_fallback_-`
`transl:nNTF` 17, [450](#), 2506
`__zrefclever_get_ref:n`
..... 58, 1675, 1691,
1703, 1708, 1731, 1745, 1749, 1761,
1765, 1800, 1820, 1979, 1992, 1999,
2023, 2035, 2039, 2049, 2053, [2069](#)
`__zrefclever_get_ref_first:` ...
..... 58, 62, 1813, 1860, [2116](#)
`__zrefclever_get_ref_font:nN` . 9,
16, 29, 67, 68, 1636, 1638, 1640, [2517](#)
`__zrefclever_get_ref_string:nN` .
.... 9, 10, 16, 29, 67, 1120, 1511,
1513, 1515, 1618, 1620, 1622, 1624,
1626, 1628, 1630, 1632, 1634, [2460](#)
`__zrefclever_get_type_transl:nnnN`
..... 9, 419, 433
`__zrefclever_get_type_transl:nnnNTF`
..... 17, [418](#), 2292, 2321, 2327, 2495
`\l__zrefclever_label_a_tl`
. 44, [1466](#), 1523, 1541, 1553, 1603,
1604, 1610, 1662, 1675, 1691, 1708,
1749, 1765, 1793, 1800, 1917, 1921,
1931, 1956, 1979, 2000, 2039, 2053
`\l__zrefclever_label_b_tl`
..... 44, [1466](#),
1526, 1531, 1543, 1555, 1560, 1921
`\l__zrefclever_label_count_int` ..
..... 45, [1464](#),
1505, 1616, 1655, 1903, 1926, 2063
`\l__zrefclever_label_enclval_a_-`
`tl` [1147](#), 1272, 1274, 1329,
1345, 1365, 1377, 1381, 1382, 1389
`\l__zrefclever_label_enclval_b_-`
`tl` [1147](#), 1275, 1277, 1330,
1352, 1360, 1379, 1383, 1384, 1391
`\l__zrefclever_label_extdoc_a_tl`
..... [1147](#), 1278,
1288, 1293, 1303, 1316, 2386, 2392
`\l__zrefclever_label_extdoc_b_tl`
..... [1147](#), 1280,
1289, 1294, 1304, 1315, 2388, 2393
`\l__zrefclever_label_type_a_tl` ..
..... 66, [1147](#), 1202, 1204,
1207, 1213, 1221, 1230, 1238, 1243,
1414, 1442, 1536, 1540, 1567, 1575,
1581, 1607, 1664, 1933, 2468, 2473,
2480, 2489, 2497, 2524, 2529, 2536
`\l__zrefclever_label_type_b_tl` ..
..... [1147](#),
1215, 1222, 1231, 1239, 1244, 1417,
1445, 1537, 1542, 1568, 1577, 1582
`__zrefclever_label_type_put_-`
`new_right:n` 37, 38, 1163, [1199](#)
`\l__zrefclever_label_types_seq` ..
.... 38, [1156](#), 1159, 1203, 1206, 1440
`__zrefclever_labels_in_sequence:nn`
..... 45, 65, 1791, 1920, [2384](#)
`\g__zrefclever_languages_prop` ...
..... 11, [244](#), 249, 251, 259,
262, 263, 274, 421, 437, 718, 763, 1000
`\l__zrefclever_last_of_type_bool`
..... 45, [1461](#), 1551, 1556, 1557,
1561, 1570, 1585, 1589, 1595, 1645
`\l__zrefclever_lastsep_tl` . [1481](#),
1627, 1690, 1707, 1730, 1748, 1760
`\l__zrefclever_link_star_bool` ...
..... 1104, [1139](#), 2076, 2197, 2349
`\l__zrefclever_listsep_tl`
... [1481](#), 1625, 1702, 1744, 1978,
1991, 1998, 2022, 2034, 2038, 2048
`\l__zrefclever_load_dict_-`
`verbose_bool` ... [270](#), 307, 318, 327
`\g__zrefclever_loaded_dictionaries_-`
`seq` [269](#), 278, 301, 312
`__zrefclever_ltxlabel:n`
72, 2637, 2649, 2654, 2663, 2669, 2675
`\l__zrefclever_main_language_tl` .
..... 23, 670,
677, 683, 687, 691, 704, 726, 749, 771
`__zrefclever_mathtools_showonlyrefs:n`
..... 1134, 2726
`\l__zrefclever_mathtools_-`
`showonlyrefs_bool` 1132, 2718, 2725
`__zrefclever_name_default:`
..... [2065](#), 2180
`\l__zrefclever_name_format_-`
`fallback_tl`
.. [1472](#), 2276, 2280, 2282, 2318, 2330
`\l__zrefclever_name_format_tl` ...
... [1472](#), 2262, 2263, 2266, 2267,
2277, 2278, 2289, 2295, 2310, 2324

<code>\l_zrefclever_name_in_link_bool</code>	<code>_zrefclever_ref_default:</code>
..... 59, 2065, 2113, 2119, 2174, 2242
62, 1472, 1828, 2121, 2353, 2369, 2370	
<code>\l_zrefclever_namefont_tl</code>	<code>\l_zrefclever_ref_language_tl</code>
..... 1481, 23, 24, 669, 690,
1637, 1831, 1848, 2139, 2170, 2185	703, 706, 711, 714, 720, 725, 729,
<code>\l_zrefclever_nameinlink_str</code>	739, 748, 751, 756, 759, 765, 770,
..... 654, 659,	774, 1105, 2293, 2322, 2328, 2496, 2502
661, 663, 665, 2351, 2357, 2359, 2363	
<code>\l_zrefclever_namesep_tl</code>	<code>\c_zrefclever_ref_options_font-</code>
.....	seq
.. 1481, 1619, 2142, 2173, 2181, 2188 10, 16, 190
<code>\l_zrefclever_next_is_same_bool</code>	<code>\c_zrefclever_ref_options-</code>
..... 45, 65, 1476,	necessarily_not_type_specific-
1914, 1942, 1958, 1964, 2413, 2451	seq
<code>\l_zrefclever_next_maybe_range-</code> 16, 190, 354, 958, 1032
bool	<code>\c_zrefclever_ref_options-</code>
.....	necessarily_type_specific_seq
.. 45, 65, 1476, 1787, 1797, 1913, 190, 385, 1076
1938, 1948, 2405, 2412, 2431, 2439	
<code>\l_zrefclever_noabbrev_first-</code>	<code>\c_zrefclever_ref_options-</code>
bool	possibly_type_specific_seq
..... 594, 603, 2273 16, 190, 371, 1053
<code>_zrefclever_orig_label:n</code>	<code>\l_zrefclever_ref_options_prop</code>
2546, 2550
<code>_zrefclever_orig_ltxlabel:n</code> 29, 31, 921, 931, 932, 2463, 2520
..... 2639, 2648, 2653, 2662, 2668, 2674	
<code>_zrefclever_page_format_aux:</code>	<code>\c_zrefclever_ref_options-</code>
..... 84, 88	reference_seq
<code>\g_zrefclever_page_format_tl</code> 190, 923
..... 6, 83, 89, 92	<code>\c_zrefclever_ref_options-</code>
<code>\l_zrefclever_pairsep_tl</code>	typesetup_seq
..... 1481, 1623, 1674, 1798 190, 970
<code>_zrefclever_prop_put_non-</code>	<code>\l_zrefclever_ref_property_tl</code>
empty:Nnn 18, 465, 470,
..... 18, 459, 846, 901	472, 474, 480, 483, 499, 508, 1160,
<code>_zrefclever_provide_dict-</code>	1192, 1534, 2071, 2125, 2192, 2395
default_transl:nn	<code>\l_zrefclever_ref_typeset_font-</code>
..... 14, 332, 362, 379	tl
<code>_zrefclever_provide_dict_type-</code> 781, 783, 1115
transl:nn	<code>\l_zrefclever_reffont_in_tl</code>
..... 14, 332, 380, 397 1481,
<code>_zrefclever_provide_dictionary:n</code>	1641, 2083, 2103, 2147, 2204, 2231
..... 9, 12-15,	<code>\l_zrefclever_reffont_out_tl</code>
35, 271, 328, 739, 750, 758, 773, 1105 1481, 1639,
<code>_zrefclever_provide_dictionary-</code>	2080, 2100, 2144, 2164, 2201, 2228
verbose:n	<code>\l_zrefclever_refpos_in_tl</code>
..... 14, 324, 705, 713, 728 1481,
<code>\l_zrefclever_range_beg_label-</code>	1635, 2092, 2107, 2152, 2220, 2236
tl	<code>\l_zrefclever_refpos_out_tl</code>
..... 45, 1476, 1504, 1481,
1703, 1726, 1732, 1742, 1746, 1758,	1631, 2095, 2109, 2165, 2223, 2238
1762, 1902, 1940, 1955, 1989, 1993,	<code>\l_zrefclever_refpre_in_tl</code>
2020, 2024, 2032, 2036, 2046, 2050 1481,
<code>\l_zrefclever_range_count_int</code>	1633, 2089, 2104, 2148, 2216, 2232
..... 45,	<code>\l_zrefclever_refpre_out_tl</code>
1476, 1507, 1683, 1717, 1905, 1941, 1481,
1952, 1957, 1963, 1971, 2012, 2058	1629, 2081, 2101, 2145, 2202, 2229
<code>\l_zrefclever_range_same_count-</code>	<code>_zrefclever_ride_on_label:n</code>
int 2544
..... 45,	<code>\l_zrefclever_setup_type_tl</code>
1476, 1508, 1670, 1705, 1718, 1906, 14, 188, 298, 336, 349, 350, 361,
1943, 1959, 1965, 1996, 2013, 2059	378, 392, 954, 982, 990, 1003, 1027,
<code>\l_zrefclever_rangesep_tl</code>	1028, 1039, 1060, 1069, 1083, 1091
..... 1481, 1621, 1764, 1799, 2052	<code>\l_zrefclever_sort_decided_bool</code>
 1153, 1283, 1297, 1307,
	1311, 1323, 1333, 1348, 1363, 1387
	<code>_zrefclever_sort_default:nn</code>
 39, 1194, 1210

_zrefclever_sort_default_different_types:nn	1519, 1520, 1527, 1550, 1874, 2365
.....	20, 37, 42, 1248, 1400
_zrefclever_sort_default_same_type:nn	37, 40, 1246, 1270
_zrefclever_sort_labels:	37–39, 44, 1113, 1157
.....	44, 1193, 1452
_zrefclever_sort_page:nn	1154, 1402, 1408, 1409, 1415, 1425, 1433
.....	1154, 1403, 1410, 1411, 1418, 1426, 1434
_zrefclever_sort_prior_a_int	1481, 1516, 1891
.....	1481, 1514, 1869
_zrefclever_sort_prior_b_int	1481, 1512, 1885
.....	options_prop 31
_zrefclever_type_<type>_options_prop	45, 62, 1464, 1506, 1866, 1868, 1877, 1904, 2260, 2272, 2366
_zrefclever_type_count_int	45, 59, 1466, 1502, 1661, 1779, 1788, 1792, 1820, 1836, 1840, 1900, 1930, 2118, 2124, 2131, 2135, 2150, 2191, 2209, 2213, 2218, 2234, 2248
_zrefclever_type_first_label_tl	45, 62, 1466, 1503, 1663, 1783, 1901, 1932, 2251, 2287, 2294, 2300, 2308, 2316, 2323, 2329, 2336
_zrefclever_type_first_label_type_tl	9, 10, 59, 1808, 2246
_zrefclever_type_name_setup:	59, 62, 1472, 1843, 1849, 2140, 2171, 2178, 2186, 2249, 2252, 2290, 2296, 2298, 2311, 2319, 2325, 2331, 2333, 2350
_zrefclever_type_name_tl	563, 566, 1915
_zrefclever_typeset_compress_bool	44, 1461, 1498, 1522, 1524, 1530
_zrefclever_typeset_labels_seq	45, 1461,
_zrefclever_typeset_last_bool	1519, 1520, 1527, 1550, 1874, 2365
_zrefclever_typeset_name_bool	514, 521, 526, 531, 1810, 1824
_zrefclever_typeset_queue_curr_tl	45, 58, 62, 1466, 1501, 1672, 1688, 1697, 1728, 1739, 1755, 1777, 1795, 1812, 1819, 1826, 1859, 1881, 1886, 1892, 1898, 1899, 1976, 1987, 2018, 2030, 2044, 2265, 2360, 2364
_zrefclever_typeset_queue_prev_tl	45, 1466, 1500, 1870, 1897
_zrefclever_typeset_range_bool	572, 575, 1112, 1775
_zrefclever_typeset_ref_bool	513, 520, 525, 530, 1810, 1817
_zrefclever_typeset_refs:	44–46, 1116, 1496
_zrefclever_typeset_refs_last_of_type:	50, 58, 59, 62, 1647, 1652
_zrefclever_typeset_refs_not_last_of_type:	45, 50, 58, 65, 1649, 1909
_zrefclever_typeset_sort_bool	539, 542, 1111
_zrefclever_typeset_seq	20, 43, 548, 553, 554, 560, 1404
_zrefclever_use_hyperref_bool	613, 620, 625, 630, 640, 646, 2075, 2196, 2348
_zrefclever_warn_hyperref_bool	614, 621, 626, 631, 644
_zrefclever_zcref:nnn	1098, 1099
_zrefclever_zcref:nnnn	35, 37, 1099
_zrefclever_zcref_labels_seq	37, 38, 1103, 1130, 1135, 1139, 1162, 1165, 1499
_zrefclever_zcref_note_tl	797, 800, 1118, 1122
_zrefclever_zcref_with_check_bool	804, 819, 1108, 1126
_zrefclever_zcsetup:n	30, 946, 947, 2559, 2564, 2587, 2592, 2599, 2619, 2680, 2711, 2757, 2770, 2787
_zrefclever_zrefcheck_available_bool	803, 814, 826, 1107, 1125