

The zref-clever package*

Code documentation

Gustavo Barros[†]

2023-02-13

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	7
4.1	Auxiliary	7
4.2	Messages	8
4.3	Data extraction	11
4.4	Option infra	11
4.5	Reference format	20
4.6	Languages	24
4.7	Language files	29
4.8	Options	42
5	Configuration	67
5.1	\zcsetup	67
5.2	\zcRefTypeSetup	67
5.3	\zcLanguageSetup	72
6	User interface	82
6.1	\zceref	82
6.2	\zcpageref	84
7	Sorting	85
8	Typesetting	91

*This file describes v0.3.4, released 2023-02-13.

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

9	Compatibility	126
9.1	appendix	126
9.2	appendices	127
9.3	memoir	128
9.4	amsmath	130
9.5	mathtools	133
9.6	breqn	134
9.7	listings	135
9.8	enumitem	136
9.9	subcaption	136
9.10	subfig	137
10	Language files	137
10.1	Localization guidelines	138
10.2	English	140
10.3	German	144
10.4	French	153
10.5	Portuguese	157
10.6	Spanish	161
10.7	Dutch	165
10.8	Italian	170
	Index	174

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 language files (which became the default input encoding in the 2018-04-01 release). Finally, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (`ltxcmdhooks`), with implications to the hook we add to `\appendix` (by Phelype Oleinik at <https://tex.stackexchange.com/q/617905> and <https://github.com/latex3/latex2e/pull/699>). Second, the support for `\@currentcounter` has been improved, including `\footnote` and `amsmath` (by Frank Mittelbach and Ulrike Fischer at <https://github.com/latex3/latex2e/issues/687>). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.

```

3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}

```

```

8      {%
9      'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
10     \MessageBreak Loading will abort!%
11     }%
12   \endinput
13 }%

Identify the package.
14 \ProvidesExplPackage {zref-clever} {2023-02-13} {0.3.4}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

Required packages. Besides these, zref-hyperref may also be loaded depending on user options. zref-clever also requires UTF-8 input encoding (see discussion with David Carlisle at <https://chat.stackexchange.com/transcript/message/62644791#62644791>).

```

16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \RequirePackage { zref-abspace }
19 \RequirePackage { ifdraft }

```

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-abspace provides the `abspace` 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 `thecounter` for reserved use. Since `\@currentlabel`, which populates the `default` property, is *more reliable* than `\@currentcounter`, `thecounter` 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 { thecounter }
23 {
24   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
25   { \use:c { the \l__zrefclever_current_counter_tl } }
26   {

```

```

27     \cs_if_exist:cT { c@ \@currentcounter }
28     { \use:c { the \@currentcounter } }
29   }
30 }
31 \zref@addprop \ZREF@mainlist { thecounter }

```

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`.

```

32 \zref@newprop { zc@type }
33 {
34   \tl_if_empty:NTF \l__zrefclever_reftype_override_tl
35   {
36     \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
37     \l__zrefclever_current_counter_tl
38     {
39       \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
40       { \l__zrefclever_current_counter_tl }
41     }
42     { \l__zrefclever_current_counter_tl }
43   }
44   { \l__zrefclever_reftype_override_tl }
45 }
46 \zref@addprop \ZREF@mainlist { zc@type }

```

Since the default/`thecounter` 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`’).

```

47 \zref@newprop { zc@cntval } [0]
48 {
49   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
50   { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
51   {
52     \cs_if_exist:cT { c@ \@currentcounter }
53     { \int_use:c { c@ \@currentcounter } }
54   }
55 }
56 \zref@addprop \ZREF@mainlist { zc@cntval }
57 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
58 \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.

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 somewhat tricky 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 `ltcounts.dtx` in `texdoc 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 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 `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\cl@<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. This function must be expandable, since it gets called from `\zref@newprop` and is the one 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>}`

```

59 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
60 {
61   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
62   {
63     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
64     \__zrefclever_get_enclosing_counters_value:e
65     { \__zrefclever_counter_reset_by:n {#1} }
66   }
67 }

```

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 (helpful comment by Enrico Gregorio, aka ‘egreg’ at https://tex.stackexchange.com/q/611370/#comment1529282_611385).

```

68 \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>}

69 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
70 {
71   \bool_if:nTF
72   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
73   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
74   {
75     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
76     { \__zrefclever_counter_reset_by_aux:nn {#1} }
77   }
78 }
79 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
80 {
81   \cs_if_exist:cT { c@ #2 }
82   {
83     \tl_if_empty:cF { c1@ #2 }
84     {
85       \tl_map_tokens:cn { c1@ #2 }
86       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
87     }
88   }
89 }
90 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
91 {
92   \str_if_eq:nnT {#2} {#3}
93   { \tl_map_break:n { \seq_map_break:n {#1} } }
94 }

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

95 \zref@newprop { zc@enclval }
96 {
97   \__zrefclever_get_enclosing_counters_value:e
98   \l__zrefclever_current_counter_tl
99 }
100 \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 set `\c@page` to “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}`.

```

101 \tl_new:N \g__zrefclever_page_format_tl
102 \AddToHook { shipout / before }
103 {
104   \group_begin:
105   \int_set:Nn \c@page { 1 }
106   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
107   \group_end:
108 }
109 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
110 \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 Auxiliary

`__zrefclever_if_package_loaded:n` Just a convenience, since sometimes we just need one of the branches, and it is particularly easy to miss the empty F branch after a long T one.

```

111 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
112 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

```

113 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
114 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

(End definition for __zrefclever_if_package_loaded:n and __zrefclever_if_class_loaded:n.)

4.2 Messages

```

115 \msg_new:nnn { zref-clever } { option-not-type-specific }
116 {
117   Option~'#1'~is-not-type-specific~\msg_line_context:~
118   Set-it-in~'\iow_char:N\zcLanguageSetup'~before-first~'type'~
119   switch-or-as~package-option.
120 }
121 \msg_new:nnn { zref-clever } { option-only-type-specific }
122 {
123   No~type~specified~for~option~'#1'~\msg_line_context:~
124   Set-it-after~'type'~switch.
125 }
126 \msg_new:nnn { zref-clever } { key-requires-value }
127 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:.~
128 \msg_new:nnn { zref-clever } { language-declared }
129 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing~to~do.~
130 \msg_new:nnn { zref-clever } { unknown-language-alias }
131 {
132   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
133   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
134   '\iow_char:N\zcDeclareLanguageAlias'.
135 }
136 \msg_new:nnn { zref-clever } { unknown-language-setup }
137 {
138   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
139   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
140   '\iow_char:N\zcDeclareLanguageAlias'.
141 }
142 \msg_new:nnn { zref-clever } { unknown-language-opt }
143 {
144   Language~'#1'~is~unknown~\msg_line_context:~
145   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
146   '\iow_char:N\zcDeclareLanguageAlias'.
147 }
148 \msg_new:nnn { zref-clever } { unknown-language-decl }
149 {
150   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
151   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
152   '\iow_char:N\zcDeclareLanguageAlias'.
153 }
154 \msg_new:nnn { zref-clever } { language-no-decl-ref }
155 {
156   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
157   Nothing~to~do~with~option~'d=#2'.
158 }
159 \msg_new:nnn { zref-clever } { language-no-gender }
160 {
161   Language~'#1'~has~no~declared~gender~\msg_line_context:~

```



```

162     Nothing~to~do~with~option~'#2=#3'.
163 }
164 \msg_new:nnn { zref-clever } { language-no-decl-setup }
165 {
166     Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
167     Nothing~to~do~with~option~'case=#2'.
168 }
169 \msg_new:nnn { zref-clever } { unknown-decl-case }
170 {
171     Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:~
172     Using~default~declension~case.
173 }
174 \msg_new:nnn { zref-clever } { nudge-multitype }
175 {
176     Reference~with~multiple~types~\msg_line_context:~
177     You~may~wish~to~separate~them~or~review~language~around~it.
178 }
179 \msg_new:nnn { zref-clever } { nudge-comptosing }
180 {
181     Multiple~labels~have~been~compressed~into~singular~type~name~
182     for~type~'#1'~\msg_line_context:.
183 }
184 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
185 {
186     Option~'sg'~signals~that~a~singular~type~name~was~expected~
187     \msg_line_context:~But~type~'#1'~has~plural~type~name.
188 }
189 \msg_new:nnn { zref-clever } { gender-not-declared }
190 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
191 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
192 {
193     Gender~mismatch~for~type~'#1'~\msg_line_context:~
194     You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
195 }
196 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
197 {
198     You've~specified~'g=#1'~\msg_line_context:~
199     But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
200 }
201 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
202 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
203 \msg_new:nnn { zref-clever } { option-document-only }
204 { Option~'#1'~is~only~available~after~\iow_char:N\\begin\{document\}. }
205 \msg_new:nnn { zref-clever } { langfile-loaded }
206 { Loaded~'#1'~language~file. }
207 \msg_new:nnn { zref-clever } { zref-property-undefined }
208 {
209     Option~'ref=#1'~requested~\msg_line_context:~
210     But~the~property~'#1'~is~not~declared,~falling~back~to~'default'.
211 }
212 \msg_new:nnn { zref-clever } { endrange-property-undefined }
213 {
214     Option~'endrange=#1'~requested~\msg_line_context:~
215     But~the~property~'#1'~is~not~declared,~'endrange'~not~set.

```

```

216 }
217 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
218 {
219   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:~
220   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
221   '\iow_char:N\zcref'.
222 }
223 \msg_new:nnn { zref-clever } { missing-hyperref }
224 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
225 \msg_new:nnn { zref-clever } { option-preamble-only }
226 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
227 \msg_new:nnn { zref-clever } { unknown-compat-module }
228 {
229   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
230   Nothing~to~do.
231 }
232 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
233 {
234   The~value~of~option~'#1'~must~be~a~comma~separated~list~
235   of~four~items.~We~received~'#2'~items~\msg_line_context:~
236   Option~not~set.
237 }
238 \msg_new:nnn { zref-clever } { missing-zref-check }
239 {
240   Option~'check'~requested~\msg_line_context:~
241   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
242 }
243 \msg_new:nnn { zref-clever } { zref-check-too-old }
244 {
245   Option~'check'~requested~\msg_line_context:~
246   But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
247 }
248 \msg_new:nnn { zref-clever } { missing-type }
249 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
250 \msg_new:nnn { zref-clever } { missing-property }
251 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
252 \msg_new:nnn { zref-clever } { missing-name }
253 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
254 \msg_new:nnn { zref-clever } { single-element-range }
255 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
256 \msg_new:nnn { zref-clever } { compat-package }
257 { Loaded~support~for~'#1'~package. }
258 \msg_new:nnn { zref-clever } { compat-class }
259 { Loaded~support~for~'#1'~documentclass. }
260 \msg_new:nnn { zref-clever } { option-deprecated }
261 {
262   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
263   Use~'#2'~instead.
264 }
265 \msg_new:nnn { zref-clever } { load-time-options }
266 {
267   'zref-clever'~does~not~accept~load~time~options.~
268   To~configure~package~options,~use~'\iow_char:N\zcsetup'.
269 }

```

4.3 Data extraction

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

```

\__zrefclever_extract_default:Nnnn {\tl var}
  {\label} {\prop} {\default}

270 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
271 {
272   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
273   { \zref@extractdefault {#2} {#3} {#4} }
274 }
275 \cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

(End definition for `_zrefclever_extract_default:Nnnn`.)

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

```

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

276 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
277 {
278   \exp_args:NNNo \exp_args:No
279   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
280 }
281 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End definition for `_zrefclever_extract_unexp:nnn`.)

`_zrefclever_extract:nnn` An internal version for `\zref@extractdefault`.

```

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

282 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
283 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for `_zrefclever_extract:nnn`.)

4.4 Option infra

This section provides the functions in which the variables naming scheme of the package options is embodied, and some basic general functions to query these option variables.

I had originally implemented the option handling of the package based on property lists, which are definitely very convenient. But as the number of options grew, I started to get concerned about the performance implications. That there was a toll was noticeable, even when we could live with it, of course. Indeed, at the time of writing, the typesetting of a reference queries about 24 different option values, most of them once per type-block, each of these queries can be potentially made in up to 5 option scope levels. Considering the size of the built-in language files is running at the hundreds, the package does have a lot of work to do in querying option values

alone, and thus it is best to smooth things in this area as much as possible. This also gives me some peace of mind that the package will scale well in the long term. For some interesting discussion about alternative methods and their performance implications, see <https://tex.stackexchange.com/q/147966>. Phelype Oleinik also offered some insight on the matter at https://tex.stackexchange.com/questions/629946/#comment1571118_629946. The only real downside of this change is that we can no longer list the whole set of options in place at a given moment, which was useful for the purposes of regression testing, since we don't know what the whole set of active options is.

`_zrefclever_opt_varname_general:nn` Defines, and leaves in the input stream, the csname of the variable used to store the general $\langle option \rangle$. The data type of the variable must be specified (`tl`, `seq`, `bool`, etc.).

```
\_zrefclever_opt_varname_general:nn {\langle option \rangle} {\langle data type \rangle}

284 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
285 { l\_zrefclever_opt_general_ #1 _ #2 }

(End definition for \_zrefclever_opt_varname_general:nn.)
```

`_zrefclever_opt_varname_type:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the type-specific $\langle option \rangle$ for $\langle ref type \rangle$.

```
\_zrefclever_opt_varname_type:nnn {\langle ref type \rangle} {\langle option \rangle} {\langle data type \rangle}

286 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
287 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
288 \cs_generate_variant:Nn \_zrefclever_opt_varname_type:nnn { enn , een }

(End definition for \_zrefclever_opt_varname_type:nnn.)
```

`_zrefclever_opt_varname_language:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the language $\langle option \rangle$ for $\langle lang \rangle$ (for general language options, those set with `\zcDeclareLanguage`). The “`lang_unknown`” branch should be guarded against, such as we normally should not get there, but this function *must* return some valid csname. The random part is there so that, in the circumstance this could not be avoided, we (hopefully) don't retrieve the value for an “unknown language” inadvertently.

```
\_zrefclever_opt_varname_language:nnn {\langle lang \rangle} {\langle option \rangle} {\langle data type \rangle}

289 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
290 {
291   \_zrefclever_language_if_declared:nTF {#1}
292   {
293     g\_zrefclever_opt_language_
294     \tl_use:c { \_zrefclever_language_varname:n {#1} }
295     _ #2 _ #3
296   }
297   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
298 }
299 \cs_generate_variant:Nn \_zrefclever_opt_varname_language:nnn { enn }

(End definition for \_zrefclever_opt_varname_language:nnn.)
```

`_zrefclever_opt_varname_lang_default:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the language-specific default reference format $\langle option \rangle$ for $\langle lang \rangle$.

```

    \_zrefclever_opt_varname_lang_default:nnn {<lang>} {<option>} {<data type>}}
300 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
301 {
302   \_zrefclever_language_if_declared:nTF {#1}
303   {
304     g__zrefclever_opt_lang_
305     \tl_use:c { \_zrefclever_language_varname:n {#1} }
306     _default_ #2 _ #3
307   }
308   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
309 }
310 \cs_generate_variant:Nn \_zrefclever_opt_varname_lang_default:nnn { enn }
(End definition for \_zrefclever_opt_varname_lang_default:nnn.)

```

`_zrefclever_opt_varname_lang_type:nnnn` Defines, and leaves in the input stream, the csname of the variable used to store the language- and type-specific reference format *<option>* for *<lang>* and *<ref type>*.

```

    \_zrefclever_opt_varname_lang_type:nnnn {<lang>} {<ref type>}
    {<option>} {<data type>}}
311 \cs_new:Npn \_zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
312 {
313   \_zrefclever_language_if_declared:nTF {#1}
314   {
315     g__zrefclever_opt_lang_
316     \tl_use:c { \_zrefclever_language_varname:n {#1} }
317     _type_ #2 _ #3 _ #4
318   }
319   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
320 }
321 \cs_generate_variant:Nn
322 \_zrefclever_opt_varname_lang_type:nnnn { eenn , eenen }
(End definition for \_zrefclever_opt_varname_lang_type:nnnn.)

```

`_zrefclever_opt_varname_fallback:nn` Defines, and leaves in the input stream, the csname of the variable used to store the fallback *<option>*.

```

    \_zrefclever_opt_varname_fallback:nn {<option>} {<data type>}}
323 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
324 { c__zrefclever_opt_fallback_ #1 _ #2 }
(End definition for \_zrefclever_opt_varname_fallback:nn.)

```

`_zrefclever_opt_var_set_bool:n` The L^AT_EX3 programming layer does not have the concept of a variable *existing* only locally, it also considers an “error” if an assignment is made to a variable which was not previously declared, but declaration is always global, which means that “setting a local variable at a local scope”, given these requirements, results in it existing, and being empty, globally. Therefore, we need an independent mechanism from the mere existence of a variable to keep track of whether variables are “set” or “unset”, within the logic of the precedence rules for options in different scopes. `_zrefclever_opt_var_set_bool:n` expands to the name of the boolean variable used to track this state for *<option var>*. See discussion with Phelype Oleinik at https://tex.stackexchange.com/questions/633341/#comment1579825_633347

__zrefclever_opt_var_set_bool:n {\<option var>}

325 \cs_new:Npn __zrefclever_opt_var_set_bool:n #1
326 { \cs_to_str:N #1 _is_set_bool }

(End definition for __zrefclever_opt_var_set_bool:n.)

__zrefclever_opt_tl_set:N {\<option tl>} {\<value>}
__zrefclever_opt_tl_clear:N {\<option tl>}
__zrefclever_opt_tl_gset:N {\<option tl>} {\<value>}
__zrefclever_opt_tl_gclear:N {\<option tl>}

__zrefclever_opt_tl_set:Nn
__zrefclever_opt_tl_clear:N
__zrefclever_opt_tl_gset:Nn
__zrefclever_opt_tl_gclear:N

327 \cs_new_protected:Npn __zrefclever_opt_tl_set:Nn #1#2
328 {
329 \tl_if_exist:NF #1
330 { \tl_new:N #1 }
331 \tl_set:Nn #1 {#2}
332 \bool_if_exist:cF { __zrefclever_opt_var_set_bool:n {#1} }
333 { \bool_new:c { __zrefclever_opt_var_set_bool:n {#1} } }
334 \bool_set_true:c { __zrefclever_opt_var_set_bool:n {#1} }
335 }
336 \cs_generate_variant:Nn __zrefclever_opt_tl_set:Nn { cn }
337 \cs_new_protected:Npn __zrefclever_opt_tl_clear:N #1
338 {
339 \tl_if_exist:NF #1
340 { \tl_new:N #1 }
341 \tl_clear:N #1
342 \bool_if_exist:cF { __zrefclever_opt_var_set_bool:n {#1} }
343 { \bool_new:c { __zrefclever_opt_var_set_bool:n {#1} } }
344 \bool_set_true:c { __zrefclever_opt_var_set_bool:n {#1} }
345 }
346 \cs_generate_variant:Nn __zrefclever_opt_tl_clear:N { c }
347 \cs_new_protected:Npn __zrefclever_opt_tl_gset:Nn #1#2
348 {
349 \tl_if_exist:NF #1
350 { \tl_new:N #1 }
351 \tl_gset:Nn #1 {#2}
352 }
353 \cs_generate_variant:Nn __zrefclever_opt_tl_gset:Nn { cn }
354 \cs_new_protected:Npn __zrefclever_opt_tl_gclear:N #1
355 {
356 \tl_if_exist:NF #1
357 { \tl_new:N #1 }
358 \tl_gclear:N #1
359 }
360 \cs_generate_variant:Nn __zrefclever_opt_tl_gclear:N { c }

(End definition for __zrefclever_opt_tl_set:Nn and others.)

__zrefclever_opt_tl_unset:N Unset {\<option tl>}.

__zrefclever_opt_tl_unset:N {\<option tl>}

361 \cs_new_protected:Npn __zrefclever_opt_tl_unset:N #1
362 {
363 \tl_if_exist:NT #1

```

364     {
365       \tl_clear:N #1 % ?
366       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
367       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
368       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
369     }
370   }
371   \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End definition for __zrefclever_opt_tl_unset:N.)

_zrefclever_opt_tl_if_set:NTF This conditional *defines* what means to be unset for a token list option. Note that the “set bool” not existing signals that the variable *is set*, that would be the case of all global option variables (language-specific ones). But this means care should be taken to always define and set the “set bool” for local variables.

```

\__zrefclever_opt_tl_if_set:N(TF) {<option tl>} {<true>} {<false>}
372 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
373 {
374   \tl_if_exist:NTF #1
375   {
376     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
377     {
378       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
379       { \prg_return_true: }
380       { \prg_return_false: }
381     }
382     { \prg_return_true: }
383   }
384   { \prg_return_false: }
385 }

```

(End definition for __zrefclever_opt_tl_if_set:NTF.)

```

\_zrefclever_opt_tl_gset_if_new:Nn \__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\_zrefclever_opt_tl_gclear_if_new:N \__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
386 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
387 {
388   \__zrefclever_opt_tl_if_set:NF #1
389   {
390     \tl_if_exist:NF #1
391     { \tl_new:N #1 }
392     \tl_gset:Nn #1 {#2}
393   }
394 }
395 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
396 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
397 {
398   \__zrefclever_opt_tl_if_set:NF #1
399   {
400     \tl_if_exist:NF #1
401     { \tl_new:N #1 }
402     \tl_gclear:N #1
403   }

```

```

404 }
405 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }

(End definition for \__zrefclever_opt_tl_gset_if_new:Nn and \__zrefclever_opt_tl_gclear_if_new:N.)

```

```

\__zrefclever_opt_tl_get:NNTF \__zrefclever_opt_tl_get:NN(TF) {<option tl to get>} {<tl var to set>}
    {<true>} {<false>}}
406 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
407 {
408     \__zrefclever_opt_tl_if_set:NTF #1
409     {
410         \tl_set_eq:NN #2 #1
411         \prg_return_true:
412     }
413     { \prg_return_false: }
414 }
415 \prg_generate_conditional_variant:Nnn
416 \__zrefclever_opt_tl_get:NN { cN } { F }

(End definition for \__zrefclever_opt_tl_get:NNTF.)

```

```

\__zrefclever_opt_seq_set_clist_split:Nn \__zrefclever_opt_seq_set_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_gset_clist_split:Nn \__zrefclever_opt_seq_gset_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_set_eq:NN \__zrefclever_opt_seq_set_eq:NN {<option seq>} {<seq var>}
\__zrefclever_opt_seq_gset_eq:NN \__zrefclever_opt_seq_gset_eq:NN {<option seq>} {<seq var>}
417 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
418 { \seq_set_split:Nnn #1 { , } {#2} }
419 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
420 { \seq_gset_split:Nnn #1 { , } {#2} }
421 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
422 {
423     \seq_if_exist:NF #1
424     { \seq_new:N #1 }
425     \seq_set_eq:NN #1 #2
426     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
427     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
428     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
429 }
430 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
431 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
432 {
433     \seq_if_exist:NF #1
434     { \seq_new:N #1 }
435     \seq_gset_eq:NN #1 #2
436 }
437 \cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }

(End definition for \__zrefclever_opt_seq_set_clist_split:Nn and others.)

```

__zrefclever_opt_seq_unset:N Unset <option seq>.

```

\__zrefclever_opt_seq_unset:N {<option seq>}

```



```

438 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
439 {
440   \seq_if_exist:NT #1
441   {
442     \seq_clear:N #1 % ?
443     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
444     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
445     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
446   }
447 }
448 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

(End definition for __zrefclever_opt_seq_unset:N.)

__zrefclever_opt_seq_if_set:NTF This conditional *defines* what means to be unset for a sequence option.

```

\__zrefclever_opt_seq_if_set:N(TF) {\<option seq>} {\<true>} {\<false>}
449 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
450 {
451   \seq_if_exist:NTF #1
452   {
453     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
454     {
455       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
456       { \prg_return_true: }
457       { \prg_return_false: }
458     }
459     { \prg_return_true: }
460   }
461   { \prg_return_false: }
462 }
463 \prg_generate_conditional_variant:Nnn
464 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_seq_if_set:NTF.)

```

\__zrefclever_opt_seq_get:NNTF \__zrefclever_opt_seq_get:NN(TF) {\<option seq to get>} {\<seq var to set>}
{\<true>} {\<false>}
465 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
466 {
467   \__zrefclever_opt_seq_if_set:NTF #1
468   {
469     \seq_set_eq:NN #2 #1
470     \prg_return_true:
471   }
472   { \prg_return_false: }
473 }
474 \prg_generate_conditional_variant:Nnn
475 \__zrefclever_opt_seq_get:NN { cN } { F }

```

(End definition for __zrefclever_opt_seq_get:NNTF.)

_zrefclever_opt_bool_unset:N Unset *<option bool>*.

```

\_zrefclever_opt_bool_unset:N {\<option bool>}

```

```

476 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
477 {
478   \bool_if_exist:NT #1
479   {
480     % \bool_set_false:N #1 % ?
481     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
482     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
483     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
484   }
485 }
486 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }

```

(End definition for __zrefclever_opt_bool_unset:N.)

__zrefclever_opt_bool_if_set:NTF This conditional *defines* what means to be unset for a boolean option.

```

\__zrefclever_opt_bool_if_set:N(TF) {\<option bool>} {\<true>} {\<false>}
487 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
488 {
489   \bool_if_exist:NTF #1
490   {
491     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
492     {
493       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
494       { \prg_return_true: }
495       { \prg_return_false: }
496     }
497     { \prg_return_true: }
498   }
499   { \prg_return_false: }
500 }
501 \prg_generate_conditional_variant:Nnn
502 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_bool_if_set:NTF.)

```

\__zrefclever_opt_bool_set_true:N {\<option bool>}
\__zrefclever_opt_bool_set_false:N {\<option bool>}
\__zrefclever_opt_bool_gset_true:N {\<option bool>}
\__zrefclever_opt_bool_gset_false:N {\<option bool>}
503 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
504 {
505   \bool_if_exist:NF #1
506   { \bool_new:N #1 }
507   \bool_set_true:N #1
508   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
509   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
510   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
511 }
512 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
513 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
514 {
515   \bool_if_exist:NF #1
516   { \bool_new:N #1 }

```

```

517     \bool_set_false:N #1
518     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
519     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
520     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
521   }
522   \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
523   \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
524   {
525     \bool_if_exist:NF #1
526     { \bool_new:N #1 }
527     \bool_gset_true:N #1
528   }
529   \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
530   \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
531   {
532     \bool_if_exist:NF #1
533     { \bool_new:N #1 }
534     \bool_gset_false:N #1
535   }
536   \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }

```

(End definition for __zrefclever_opt_bool_set_true:N and others.)

```

\__zrefclever_opt_bool_get:NNTF      \__zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
                                     {{true}} {{false}}
537   \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
538   {
539     \__zrefclever_opt_bool_if_set:NTF #1
540     {
541       \bool_set_eq:NN #2 #1
542       \prg_return_true:
543     }
544     { \prg_return_false: }
545   }
546   \prg_generate_conditional_variant:Nnn
547   \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End definition for __zrefclever_opt_bool_get:NNTF.)

```

\__zrefclever_opt_bool_if:NTF      \__zrefclever_opt_bool_if:N(TF) {<option bool>} {{true}} {{false}}
548   \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
549   {
550     \__zrefclever_opt_bool_if_set:NTF #1
551     { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
552     { \prg_return_false: }
553   }
554   \prg_generate_conditional_variant:Nnn
555   \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

(End definition for __zrefclever_opt_bool_if:NTF.)

4.5 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, these precedence rules are handled / enforced in `__zrefclever_get_rf_opt_tl:nnnN`, `__zrefclever_get_rf_opt_seq:nnnN`, `__zrefclever_get_rf_opt_bool:nnnnN`, and `__zrefclever_type_name_setup`: which are the basic functions to retrieve proper values for reference format settings.

The fact that we have multiple scopes to set reference format options has some implications for how we handle these options, and for the resulting UI. Since there is a clear precedence rule between the different levels, setting an option at a high priority level shadows everything below it. Hence, it may be relevant to be able to “unset” these options too, so as to be able go back to the lower precedence level of the language-specific options at any given point. However, since many of these options are token lists, or clists, for which “empty” is a legitimate value, we cannot rely on emptiness to distinguish that particular intention. How to deal with it, depends on the kind of option (its data type, to be precise). For token lists and clists/sequences, 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 in `\keys_define:nn` by means of the `.default:o` property of the key. For the technique, by Jonathan P. Spratte, aka ‘Skillmon’, and some discussion about it, including further insights by Phelype Oleinik, see <https://tex.stackexchange.com/q/614690> and <https://github.com/latex3/latex3/pull/988>. However, Joseph Wright seems to particularly dislike this use and the general idea of a “key with no value” being somehow meaningful for `l3keys` (e.g. his comments on the previous question, and https://tex.stackexchange.com/q/632157/#comment1576404_632157), which does make it somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the “key with no value” is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value “unset” for this purpose. And similarly for “choice” options.

However, “unsetting” options is only supported at the general and reference type levels, that is, at `\zcsetup`, at `\zceref`, and at `\zcRefTypeSetup`. For language-specific options – in the language files or at `\zcLanguageSetup` – there is no unsetting, an option which has been set can there only be changed to another value. This for two reasons. First, these are low precedence levels, so it is less meaningful to be able to unset these options. Second, these settings can only be done in the preamble (or the package itself). They are meant to be global. So, do it once, do it right, and if you need to locally change something along the document, use a higher precedence level.

<code>\l__zrefclever_setup_type_tl</code>	Store “current” type, language, and declension cases in different places for type-specific and language-specific options handling, notably in <code>__zrefclever_provide_langfile:n</code> , <code>\zcRefTypeSetup</code> , and <code>\zcLanguageSetup</code> , but also for language specific options retrieval.
<code>\l__zrefclever_setup_language_tl</code>	
<code>\l__zrefclever_lang_decl_case_tl</code>	
<code>\l__zrefclever_lang_declension_seq</code>	
<code>\l__zrefclever_lang_gender_seq</code>	

```

556 \tl_new:N \l__zrefclever_setup_type_tl
557 \tl_new:N \l__zrefclever_setup_language_tl
558 \tl_new:N \l__zrefclever_lang_decl_case_tl
559 \seq_new:N \l__zrefclever_lang_declension_seq
560 \seq_new:N \l__zrefclever_lang_gender_seq

```

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

```

zrefclever_rf_opts_tl_not_type_specific_seq
efclever_rf_opts_tl_maybe_type_specific_seq
\g_zrefclever_rf_opts_seq_refbounds_seq
clever_rf_opts_bool_maybe_type_specific_seq
\g_zrefclever_rf_opts_tl_type_names_seq
\g_zrefclever_rf_opts_tl_typesetup_seq
\g_zrefclever_rf_opts_tl_reference_seq

```

Lists of reference format 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. These variables are *constants*, but I don’t seem to be able to find a way to concatenate two constants into a third one without triggering L^AT_EX3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```

561 \seq_new:N \g_zrefclever_rf_opts_tl_not_type_specific_seq
562 \seq_gset_from_clist:Nn
563 \g_zrefclever_rf_opts_tl_not_type_specific_seq
564 {
565     tpairsep ,
566     tlistsep ,
567     tlastsep ,
568     notesep ,
569 }
570 \seq_new:N \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
571 \seq_gset_from_clist:Nn
572 \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
573 {
574     namesep ,
575     pairsep ,
576     listsep ,
577     lastsep ,
578     rangesep ,
579     namefont ,
580     reffont ,
581 }
582 \seq_new:N \g_zrefclever_rf_opts_seq_refbounds_seq
583 \seq_gset_from_clist:Nn
584 \g_zrefclever_rf_opts_seq_refbounds_seq
585 {
586     refbounds-first ,
587     refbounds-first-sg ,
588     refbounds-first-pb ,
589     refbounds-first-rb ,
590     refbounds-mid ,
591     refbounds-mid-rb ,
592     refbounds-mid-re ,
593     refbounds-last ,
594     refbounds-last-pe ,
595     refbounds-last-re ,
596 }
597 \seq_new:N \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
598 \seq_gset_from_clist:Nn
599 \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
600 {
601     cap ,
602     abbrev ,
603     rangetopair ,
604 }

```

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_rf_opt_tl:nnnN, but by __zrefclever_type_name_setup:.

```

605 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
606 \seq_gset_from_clist:Nn
607 \g__zrefclever_rf_opts_tl_type_names_seq
608 {
609   Name-sg ,
610   name-sg ,
611   Name-pl ,
612   name-pl ,
613   Name-sg-ab ,
614   name-sg-ab ,
615   Name-pl-ab ,
616   name-pl-ab ,
617 }

```

And, finally, some combined groups of the above variables, for convenience.

```

618 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
619 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
620 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
621 \g__zrefclever_rf_opts_tl_type_names_seq
622 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
623 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
624 \g__zrefclever_rf_opts_tl_not_type_specific_seq
625 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

(End definition for \g__zrefclever_rf_opts_tl_not_type_specific_seq and others.)

We set here also the “derived” refbounds options, which are (almost) the same for every option scope.

```

626 \clist_map_inline:nn
627 {
628   reference ,
629   typesetup ,
630   langsetup ,
631   langfile ,
632 }
633 {
634   \keys_define:nn { zref-clever/ #1 }
635   {
636     +refbounds-first .meta:n =
637     {
638       refbounds-first = {##1} ,
639       refbounds-first-sg = {##1} ,
640       refbounds-first-pb = {##1} ,
641       refbounds-first-rb = {##1} ,
642     } ,
643     +refbounds-mid .meta:n =
644     {
645       refbounds-mid = {##1} ,
646       refbounds-mid-rb = {##1} ,
647       refbounds-mid-re = {##1} ,
648     } ,
649     +refbounds-last .meta:n =
650     {
651       refbounds-last = {##1} ,

```

```

652         refbounds-last-pe = {##1} ,
653         refbounds-last-re = {##1} ,
654     } ,
655     +refbounds-rb .meta:n =
656     {
657         refbounds-first-rb = {##1} ,
658         refbounds-mid-rb = {##1} ,
659     } ,
660     +refbounds-re .meta:n =
661     {
662         refbounds-mid-re = {##1} ,
663         refbounds-last-re = {##1} ,
664     } ,
665     +refbounds .meta:n =
666     {
667         +refbounds-first = {##1} ,
668         +refbounds-mid = {##1} ,
669         +refbounds-last = {##1} ,
670     } ,
671     refbounds .meta:n = { +refbounds = {##1} } ,
672 }
673 }
674 \clist_map_inline:nn
675 {
676     reference ,
677     typesetup ,
678 }
679 {
680     \keys_define:nn { zref-clever/ #1 }
681     {
682         +refbounds-first .default:o = \c_novalue_tl ,
683         +refbounds-mid .default:o = \c_novalue_tl ,
684         +refbounds-last .default:o = \c_novalue_tl ,
685         +refbounds-rb .default:o = \c_novalue_tl ,
686         +refbounds-re .default:o = \c_novalue_tl ,
687         +refbounds .default:o = \c_novalue_tl ,
688         refbounds .default:o = \c_novalue_tl ,
689     }
690 }
691 \clist_map_inline:nn
692 {
693     langsetup ,
694     langfile ,
695 }
696 {
697     \keys_define:nn { zref-clever/ #1 }
698     {
699         +refbounds-first .value_required:n = true ,
700         +refbounds-mid .value_required:n = true ,
701         +refbounds-last .value_required:n = true ,
702         +refbounds-rb .value_required:n = true ,
703         +refbounds-re .value_required:n = true ,
704         +refbounds .value_required:n = true ,
705         refbounds .value_required:n = true ,

```

```

706     }
707 }

```

4.6 Languages

`\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.

```

708 \tl_new:N \l__zrefclever_ref_language_tl
709 \tl_new:N \l__zrefclever_current_language_tl
710 \tl_new:N \l__zrefclever_main_language_tl

```

`\l_zrefclever_ref_language_tl` A public version of `\l__zrefclever_ref_language_tl` for use in `zref-vario`.

```

711 \tl_new:N \l_zrefclever_ref_language_tl
712 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```

(End definition for `\l_zrefclever_ref_language_tl`. This function is documented on page ??.)

`_zrefclever_language_varname:n` Defines, and leaves in the input stream, the csname of the variable used to store the *base language* (as the value of this variable) for a *language* declared for `zref-clever`.

```

    \_zrefclever_language_varname:n {<language>}

713 \cs_new:Npn \_zrefclever_language_varname:n #1
714 { g_zrefclever_declared_language_ #1 _tl }

```

(End definition for `_zrefclever_language_varname:n`.)

`\zrefclever_language_varname:n` A public version of `_zrefclever_language_varname:n` for use in `zref-vario`.

```

715 \cs_set_eq:NN \zrefclever_language_varname:n
716 \_zrefclever_language_varname:n

```

(End definition for `\zrefclever_language_varname:n`. This function is documented on page ??.)

`_zrefclever_language_if_declared:nTF` A language is considered to be declared for `zref-clever` if it passes this conditional, which requires that a variable with `_zrefclever_language_varname:n{<language>}` exists.

```

    \_zrefclever_language_if_declared:n(TF) {<language>}

717 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
718 {
719     \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
720     { \prg_return_true: }
721     { \prg_return_false: }
722 }
723 \prg_generate_conditional_variant:Nnn
724 \_zrefclever_language_if_declared:n { x } { T , F , TF }

```

(End definition for `_zrefclever_language_if_declared:nTF`.)

`\zrefclever_language_if_declared:nTF` A public version of `__zrefclever_language_if_declared:n` for use in `zref-vario`.

```

725 \prg_set_eq_conditional:Nn \zrefclever_language_if_declared:n
726 \__zrefclever_language_if_declared:n { TF }

```

(End definition for `\zrefclever_language_if_declared:nTF`. This function is documented on page ??.)

`\zcDeclareLanguage` Declare a new language for use with `zref-clever`. $\langle language \rangle$ is taken to be both the “language name” and the “base language name”. A “base language” (loose concept here, meaning just “the name we gave for the language file 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 “base language name”, in other words, it is an “alias to itself”. [$\langle options \rangle$] receive a `k=v` set of options, with three valid options. The first, `declension`, takes the noun declension cases prefixes for $\langle language \rangle$ as a comma separated list, whose first element is taken to be the default case. The second, `gender`, receives the genders for $\langle language \rangle$ as comma separated list. The third, `allcaps`, is a boolean, and indicates that for $\langle language \rangle$ all nouns must be capitalized for grammatical reasons, in which case, the `cap` option is disregarded for $\langle language \rangle$. If $\langle language \rangle$ is already known, just warn. This implies a particular restriction regarding [$\langle options \rangle$], namely that these options, when defined by the package, cannot be redefined by the user. This is deliberate, otherwise the built-in language files would become much too sensitive to this particular user input, and unnecessarily so. `\zcDeclareLanguage` is preamble only.

```

\zcDeclareLanguage [ $\langle options \rangle$ ] { $\langle language \rangle$ }

727 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
728 {
729   \group_begin:
730   \tl_if_empty:nF {#2}
731   {
732     \__zrefclever_language_if_declared:nTF {#2}
733     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
734     {
735       \tl_new:c { \__zrefclever_language_varname:n {#2} }
736       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
737       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
738       \keys_set:nn { zref-clever/declarelang } {#1}
739     }
740   }
741   \group_end:
742 }
743 \@onlypreamble \zcDeclareLanguage

```

(End definition for `\zcDeclareLanguage`.)

`\zcDeclareLanguageAlias` Declare $\langle language alias \rangle$ to be an alias of $\langle aliased language \rangle$ (or “base language”). $\langle aliased language \rangle$ must be already known to `zref-clever`. `\zcDeclareLanguageAlias` is preamble only.

```

\zcDeclareLanguageAlias { $\langle language alias \rangle$ } { $\langle aliased language \rangle$ }

744 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
745 {
746   \tl_if_empty:nF {#1}
747   {

```

```

748     \_zrefclever_language_if_declared:nTF {#2}
749     {
750         \tl_new:c { \_zrefclever_language_varname:n {#1} }
751         \tl_gset:cx { \_zrefclever_language_varname:n {#1} }
752         { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
753     }
754     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
755 }
756 }
757 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

758 \keys_define:nn { zref-clever/declarelang }
759 {
760     declension .code:n =
761     {
762         \seq_new:c
763         {
764             \_zrefclever_opt_varname_language:enn
765             { \l__zrefclever_setup_language_tl } { declension } { seq }
766         }
767         \seq_gset_from_clist:cn
768         {
769             \_zrefclever_opt_varname_language:enn
770             { \l__zrefclever_setup_language_tl } { declension } { seq }
771         }
772         {#1}
773     } ,
774     declension .value_required:n = true ,
775     gender .code:n =
776     {
777         \seq_new:c
778         {
779             \_zrefclever_opt_varname_language:enn
780             { \l__zrefclever_setup_language_tl } { gender } { seq }
781         }
782         \seq_gset_from_clist:cn
783         {
784             \_zrefclever_opt_varname_language:enn
785             { \l__zrefclever_setup_language_tl } { gender } { seq }
786         }
787         {#1}
788     } ,
789     gender .value_required:n = true ,
790     allcaps .choices:nn =
791     { true , false }
792     {
793         \bool_new:c
794         {
795             \_zrefclever_opt_varname_language:enn
796             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
797         }
798         \use:c { bool_gset_ \l_keys_choice_tl :c }
799         {

```

```

800         \_zrefclever_opt_varname_language:enn
801         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
802     }
803 },
804     allcaps .default:n = true ,
805 }

```

`_zrefclever_process_language_settings:` Auxiliary function for `_zrefclever_zcref:nnn`, responsible for processing language related settings. It is necessary to separate them from the reference options machinery for two reasons. First, because their behavior is language dependent, but the language itself can also be set as an option (`lang`, value stored in `\l__zrefclever_ref_language_tl`). Second, some of its tasks must be done regardless of any option being given (e.g. the default declension case, the `allcaps` option). Hence, we must validate the language settings after the reference options have been set. It is expected to be called right (or soon) after `\keys_set:nn` in `_zrefclever_zcref:nnn`, where current values for `\l__zrefclever_ref_language_tl` and `\l__zrefclever_ref_decl_case_tl` are in place.

```

806 \cs_new_protected:Npn \_zrefclever_process_language_settings:
807 {
808     \_zrefclever_language_if_declared:xTF
809     { \l__zrefclever_ref_language_tl }
810     {

```

Validate the declension case (d) option against the declared cases for the reference language. If the user value for the latter does not match the declension cases declared for the former, the function sets an appropriate value for `\l__zrefclever_ref_decl_case_tl`, either using the default case, or clearing the variable, depending on the language setup. And also issues a warning about it.

```

811     \_zrefclever_opt_seq_get:cNF
812     {
813         \_zrefclever_opt_varname_language:enn
814         { \l__zrefclever_ref_language_tl } { declension } { seq }
815     }
816     \l__zrefclever_lang_declension_seq
817     { \seq_clear:N \l__zrefclever_lang_declension_seq }
818     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
819     {
820         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
821         {
822             \msg_warning:nxxx { zref-clever }
823             { language-no-decl-ref }
824             { \l__zrefclever_ref_language_tl }
825             { \l__zrefclever_ref_decl_case_tl }
826             \tl_clear:N \l__zrefclever_ref_decl_case_tl
827         }
828     }
829     {
830         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
831         {
832             \seq_get_left:NN \l__zrefclever_lang_declension_seq
833             \l__zrefclever_ref_decl_case_tl
834         }
835         {
836             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

837         \l__zrefclever_ref_decl_case_tl
838     {
839         \msg_warning:nxxx { zref-clever }
840         { unknown-decl-case }
841         { \l__zrefclever_ref_decl_case_tl }
842         { \l__zrefclever_ref_language_tl }
843         \seq_get_left:NN \l__zrefclever_lang_declension_seq
844         \l__zrefclever_ref_decl_case_tl
845     }
846 }
847 }

```

Validate the gender (g) option against the declared genders for the reference language. If the user value for the latter does not match the genders declared for the former, clear `\l__zrefclever_ref_gender_tl` and warn.

```

848     \__zrefclever_opt_seq_get:cNF
849     {
850         \__zrefclever_opt_varname_language:enn
851         { \l__zrefclever_ref_language_tl } { gender } { seq }
852     }
853     \l__zrefclever_lang_gender_seq
854     { \seq_clear:N \l__zrefclever_lang_gender_seq }
855     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
856     {
857         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
858         {
859             \msg_warning:nxxxx { zref-clever }
860             { language-no-gender }
861             { \l__zrefclever_ref_language_tl }
862             { g }
863             { \l__zrefclever_ref_gender_tl }
864             \tl_clear:N \l__zrefclever_ref_gender_tl
865         }
866     }
867     {
868         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
869         {
870             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
871             \l__zrefclever_ref_gender_tl
872             {
873                 \msg_warning:nxxx { zref-clever }
874                 { gender-not-declared }
875                 { \l__zrefclever_ref_language_tl }
876                 { \l__zrefclever_ref_gender_tl }
877                 \tl_clear:N \l__zrefclever_ref_gender_tl
878             }
879         }
880     }

```

Ensure the general cap is set to true when the language was declared with `allcaps` option.

```

881     \__zrefclever_opt_bool_if:cT
882     {
883         \__zrefclever_opt_varname_language:enn
884         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

885     }
886     { \keys_set:nn { zref-clever/reference } { cap = true } }
887   }
888   {

```

If the language itself is not declared, we still have to issue declension and gender warnings, if `d` or `g` options were used.

```

889     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
890     {
891       \msg_warning:nxxx { zref-clever } { unknown-language-decl }
892       { \l__zrefclever_ref_decl_case_tl }
893       { \l__zrefclever_ref_language_tl }
894       \tl_clear:N \l__zrefclever_ref_decl_case_tl
895     }
896     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
897     {
898       \msg_warning:nxxxx { zref-clever }
899       { language-no-gender }
900       { \l__zrefclever_ref_language_tl }
901       { g }
902       { \l__zrefclever_ref_gender_tl }
903       \tl_clear:N \l__zrefclever_ref_gender_tl
904     }
905   }
906 }

```

(End definition for `__zrefclever_process_language_settings:.`)

4.7 Language files

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

The built-in language files and their related infrastructure are designed to perform “on the fly” loading of the language files, “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”. 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 current 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 language files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.lbx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read

and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load language files 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 language files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/langfile}` by `__zrefclever_provide_langfile:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The language file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`__zrefclever_provide_langfile:n` is only meant to load the built-in language files. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a corresponding variables. Hence, there is no need to “load” anything in this case: definitions and assignments made by the user are performed immediately.

`\g__zrefclever_loaded_langfiles_seq` Used to keep track of whether a language file has already been loaded or not.

```
907 \seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(End definition for `\g__zrefclever_loaded_langfiles_seq`.)

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

```
\__zrefclever_provide_langfile:n {\language}}

908 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
909 {
910   \group_begin:
911   \@bsphack
912   \__zrefclever_language_if_declared:nT {#1}
913   {
914     \seq_if_in:NxF
915     \g__zrefclever_loaded_langfiles_seq
916     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
917     {
918       \exp_args:Nx \file_get:nnNTF
919       {
920         zref-clever-
921         \tl_use:c { \__zrefclever_language_varname:n {#1} }
922         .lang
923       }
924       { \ExplSyntaxOn }
925       \l_tmpa_tl
926       {
927         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
928         \tl_clear:N \l__zrefclever_setup_type_tl
929         \__zrefclever_opt_seq_get:cNF
930         {
931           \__zrefclever_opt_varname_language:nnn
932           {#1} { declension } { seq }
933         }
934         \l__zrefclever_lang_declension_seq
935         { \seq_clear:N \l__zrefclever_lang_declension_seq }
936         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
```

```

937         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
938       {
939         \seq_get_left:NN \l__zrefclever_lang_declension_seq
940         \l__zrefclever_lang_decl_case_tl
941       }
942     \__zrefclever_opt_seq_get:cNF
943     {
944       \__zrefclever_opt_varname_language:nnn
945       {#1} { gender } { seq }
946     }
947     \l__zrefclever_lang_gender_seq
948     { \seq_clear:N \l__zrefclever_lang_gender_seq }
949     \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
950     \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
951     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
952     \msg_info:nnx { zref-clever } { langfile-loaded }
953     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
954   }
955   {

```

Even if we don't have the actual language file, 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, if it was not found the first time, it won't be the next.

```

956         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
957         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
958       }
959     }
960   }
961   \@esphack
962   \group_end:
963 }
964 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }

```

(End definition for `__zrefclever_provide_langfile:n`.)

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

```

965 \keys_define:nn { zref-clever/langfile }
966 {
967   type .code:n =
968   {
969     \tl_if_empty:nTF {#1}
970     { \tl_clear:N \l__zrefclever_setup_type_tl }
971     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
972   } ,
973
974   case .code:n =
975   {
976     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
977     {
978       \msg_info:nnxx { zref-clever } { language-no-decl-setup }
979       { \l__zrefclever_setup_language_tl } {#1}

```

```

980     }
981     {
982         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
983         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
984         {
985             \msg_info:nxxx { zref-clever } { unknown-decl-case }
986             {#1} { \l__zrefclever_setup_language_tl }
987             \seq_get_left:NN \l__zrefclever_lang_declension_seq
988             \l__zrefclever_lang_decl_case_tl
989         }
990     }
991 },
992 case .value_required:n = true ,
993
994 gender .value_required:n = true ,
995 gender .code:n =
996 {
997     \seq_if_empty:NnTF \l__zrefclever_lang_gender_seq
998     {
999         \msg_info:nxxxx { zref-clever } { language-no-gender }
1000         { \l__zrefclever_setup_language_tl } { gender } {#1}
1001     }
1002     {
1003         \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1004         {
1005             \msg_info:nnn { zref-clever }
1006             { option-only-type-specific } { gender }
1007         }
1008         {
1009             \seq_clear:N \l_tmpa_seq
1010             \clist_map_inline:nn {#1}
1011             {
1012                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1013                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
1014                 {
1015                     \msg_info:nxxx { zref-clever }
1016                     { gender-not-declared }
1017                     { \l__zrefclever_setup_language_tl } {##1}
1018                 }
1019             }
1020             \__zrefclever_opt_seq_if_set:cF
1021             {
1022                 \__zrefclever_opt_varname_lang_type:eenn
1023                 { \l__zrefclever_setup_language_tl }
1024                 { \l__zrefclever_setup_type_tl }
1025                 { gender }
1026                 { seq }
1027             }
1028             {
1029                 \seq_new:c
1030                 {
1031                     \__zrefclever_opt_varname_lang_type:eenn
1032                     { \l__zrefclever_setup_language_tl }
1033                     { \l__zrefclever_setup_type_tl }

```



```

1034         { gender }
1035         { seq }
1036     }
1037     \seq_gset_eq:cN
1038     {
1039         \__zrefclever_opt_varname_lang_type:eenn
1040         { \l__zrefclever_setup_language_tl }
1041         { \l__zrefclever_setup_type_tl }
1042         { gender }
1043         { seq }
1044     }
1045     \l_tmpa_seq
1046 }
1047 }
1048 }
1049 },
1050 }
1051 \seq_map_inline:Nn
1052 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1053 {
1054     \keys_define:nn { zref-clever/langfile }
1055     {
1056         #1 .value_required:n = true ,
1057         #1 .code:n =
1058         {
1059             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1060             {
1061                 \__zrefclever_opt_tl_gset_if_new:cn
1062                 {
1063                     \__zrefclever_opt_varname_lang_default:enn
1064                     { \l__zrefclever_setup_language_tl }
1065                     {#1} { tl }
1066                 }
1067                 {##1}
1068             }
1069             {
1070                 \msg_info:nnn { zref-clever }
1071                 { option-not-type-specific } {#1}
1072             }
1073         },
1074     }
1075 }
1076 \seq_map_inline:Nn
1077 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1078 {
1079     \keys_define:nn { zref-clever/langfile }
1080     {
1081         #1 .value_required:n = true ,
1082         #1 .code:n =
1083         {
1084             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1085             {
1086                 \__zrefclever_opt_tl_gset_if_new:cn
1087                 {

```

```

1088         \_zrefclever_opt_varname_lang_default:enn
1089         { \l_zrefclever_setup_language_tl }
1090         {#1} { tl }
1091     }
1092     {##1}
1093 }
1094 {
1095     \_zrefclever_opt_tl_gset_if_new:cn
1096     {
1097         \_zrefclever_opt_varname_lang_type:enn
1098         { \l_zrefclever_setup_language_tl }
1099         { \l_zrefclever_setup_type_tl }
1100         {#1} { tl }
1101     }
1102     {##1}
1103 }
1104 },
1105 }
1106 }
1107 \keys_define:nn { zref-clever/langfile }
1108 {
1109     endrange .value_required:n = true ,
1110     endrange .code:n =
1111     {
1112         \str_case:nnF {#1}
1113         {
1114             { ref }
1115             {
1116                 \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1117                 {
1118                     \_zrefclever_opt_tl_gclear_if_new:c
1119                     {
1120                         \_zrefclever_opt_varname_lang_default:enn
1121                         { \l_zrefclever_setup_language_tl }
1122                         { endrangefunc } { tl }
1123                     }
1124                     \_zrefclever_opt_tl_gclear_if_new:c
1125                     {
1126                         \_zrefclever_opt_varname_lang_default:enn
1127                         { \l_zrefclever_setup_language_tl }
1128                         { endrangeprop } { tl }
1129                     }
1130                 }
1131             }
1132             \_zrefclever_opt_tl_gclear_if_new:c
1133             {
1134                 \_zrefclever_opt_varname_lang_type:enn
1135                 { \l_zrefclever_setup_language_tl }
1136                 { \l_zrefclever_setup_type_tl }
1137                 { endrangefunc } { tl }
1138             }
1139             \_zrefclever_opt_tl_gclear_if_new:c
1140             {
1141                 \_zrefclever_opt_varname_lang_type:enn

```

```

1142         { \l__zrefclever_setup_language_tl }
1143         { \l__zrefclever_setup_type_tl }
1144         { endrangeprop } { tl }
1145     }
1146 }
1147 }
1148
1149 { stripprefix }
1150 {
1151     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1152     {
1153         \__zrefclever_opt_tl_gset_if_new:cn
1154         {
1155             \__zrefclever_opt_varname_lang_default:enn
1156             { \l__zrefclever_setup_language_tl }
1157             { endrangefunc } { tl }
1158         }
1159         { __zrefclever_get_endrange_stripprefix }
1160         \__zrefclever_opt_tl_gclear_if_new:c
1161         {
1162             \__zrefclever_opt_varname_lang_default:enn
1163             { \l__zrefclever_setup_language_tl }
1164             { endrangeprop } { tl }
1165         }
1166     }
1167     {
1168         \__zrefclever_opt_tl_gset_if_new:cn
1169         {
1170             \__zrefclever_opt_varname_lang_type:eenn
1171             { \l__zrefclever_setup_language_tl }
1172             { \l__zrefclever_setup_type_tl }
1173             { endrangefunc } { tl }
1174         }
1175         { __zrefclever_get_endrange_stripprefix }
1176         \__zrefclever_opt_tl_gclear_if_new:c
1177         {
1178             \__zrefclever_opt_varname_lang_type:eenn
1179             { \l__zrefclever_setup_language_tl }
1180             { \l__zrefclever_setup_type_tl }
1181             { endrangeprop } { tl }
1182         }
1183     }
1184 }
1185
1186 { pagecomp }
1187 {
1188     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1189     {
1190         \__zrefclever_opt_tl_gset_if_new:cn
1191         {
1192             \__zrefclever_opt_varname_lang_default:enn
1193             { \l__zrefclever_setup_language_tl }
1194             { endrangefunc } { tl }
1195         }

```

```

1196         { __zrefclever_get_endrange_pagecomp }
1197     \__zrefclever_opt_tl_gclear_if_new:c
1198     {
1199         \__zrefclever_opt_varname_lang_default:enn
1200         { \l__zrefclever_setup_language_tl }
1201         { endrangeprop } { t1 }
1202     }
1203 }
1204 {
1205     \__zrefclever_opt_tl_gset_if_new:cn
1206     {
1207         \__zrefclever_opt_varname_lang_type:een
1208         { \l__zrefclever_setup_language_tl }
1209         { \l__zrefclever_setup_type_tl }
1210         { endrangefunc } { t1 }
1211     }
1212     { __zrefclever_get_endrange_pagecomp }
1213     \__zrefclever_opt_tl_gclear_if_new:c
1214     {
1215         \__zrefclever_opt_varname_lang_type:een
1216         { \l__zrefclever_setup_language_tl }
1217         { \l__zrefclever_setup_type_tl }
1218         { endrangeprop } { t1 }
1219     }
1220 }
1221 }
1222
1223 { pagecomp2 }
1224 {
1225     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1226     {
1227         \__zrefclever_opt_tl_gset_if_new:cn
1228         {
1229             \__zrefclever_opt_varname_lang_default:enn
1230             { \l__zrefclever_setup_language_tl }
1231             { endrangefunc } { t1 }
1232         }
1233         { __zrefclever_get_endrange_pagecomptwo }
1234         \__zrefclever_opt_tl_gclear_if_new:c
1235         {
1236             \__zrefclever_opt_varname_lang_default:enn
1237             { \l__zrefclever_setup_language_tl }
1238             { endrangeprop } { t1 }
1239         }
1240     }
1241     {
1242         \__zrefclever_opt_tl_gset_if_new:cn
1243         {
1244             \__zrefclever_opt_varname_lang_type:een
1245             { \l__zrefclever_setup_language_tl }
1246             { \l__zrefclever_setup_type_tl }
1247             { endrangefunc } { t1 }
1248         }
1249         { __zrefclever_get_endrange_pagecomptwo }

```

```

1250         \_zrefclever_opt_tl_gclear_if_new:c
1251         {
1252             \_zrefclever_opt_varname_lang_type:enn
1253             { \l_zrefclever_setup_language_tl }
1254             { \l_zrefclever_setup_type_tl }
1255             { endrangeprop } { tl }
1256         }
1257     }
1258 }
1259 }
1260 {
1261     \tl_if_empty:nTF {#1}
1262     {
1263         \msg_info:nnn { zref-clever }
1264         { endrange-property-undefined } {#1}
1265     }
1266     {
1267         \zref@ifpropundefined {#1}
1268         {
1269             \msg_info:nnn { zref-clever }
1270             { endrange-property-undefined } {#1}
1271         }
1272         {
1273             \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1274             {
1275                 \_zrefclever_opt_tl_gset_if_new:cn
1276                 {
1277                     \_zrefclever_opt_varname_lang_default:enn
1278                     { \l_zrefclever_setup_language_tl }
1279                     { endrangefunc } { tl }
1280                 }
1281                 { __zrefclever_get_endrange_property }
1282                 \_zrefclever_opt_tl_gset_if_new:cn
1283                 {
1284                     \_zrefclever_opt_varname_lang_default:enn
1285                     { \l_zrefclever_setup_language_tl }
1286                     { endrangeprop } { tl }
1287                 }
1288                 {#1}
1289             }
1290             {
1291                 \_zrefclever_opt_tl_gset_if_new:cn
1292                 {
1293                     \_zrefclever_opt_varname_lang_type:enn
1294                     { \l_zrefclever_setup_language_tl }
1295                     { \l_zrefclever_setup_type_tl }
1296                     { endrangefunc } { tl }
1297                 }
1298                 { __zrefclever_get_endrange_property }
1299                 \_zrefclever_opt_tl_gset_if_new:cn
1300                 {
1301                     \_zrefclever_opt_varname_lang_type:enn
1302                     { \l_zrefclever_setup_language_tl }
1303                     { \l_zrefclever_setup_type_tl }

```

```

1304             { endrangeprop } { t1 }
1305         }
1306         {#1}
1307     }
1308 }
1309 }
1310 }
1311 } ,
1312 }
1313 \seq_map_inline:Nn
1314 \g__zrefclever_rf_opts_tl_type_names_seq
1315 {
1316     \keys_define:nn { zref-clever/langfile }
1317     {
1318         #1 .value_required:n = true ,
1319         #1 .code:n =
1320         {
1321             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1322             {
1323                 \msg_info:nnn { zref-clever }
1324                 { option-only-type-specific } {#1}
1325             }
1326             {
1327                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1328                 {
1329                     \__zrefclever_opt_tl_gset_if_new:cn
1330                     {
1331                         \__zrefclever_opt_varname_lang_type:een
1332                         { \l__zrefclever_setup_language_tl }
1333                         { \l__zrefclever_setup_type_tl }
1334                         {#1} { t1 }
1335                     }
1336                     {##1}
1337                 }
1338                 {
1339                     \__zrefclever_opt_tl_gset_if_new:cn
1340                     {
1341                         \__zrefclever_opt_varname_lang_type:een
1342                         { \l__zrefclever_setup_language_tl }
1343                         { \l__zrefclever_setup_type_tl }
1344                         { \l__zrefclever_lang_decl_case_tl - #1 } { t1 }
1345                     }
1346                     {##1}
1347                 }
1348             }
1349         } ,
1350     }
1351 }
1352 \seq_map_inline:Nn
1353 \g__zrefclever_rf_opts_seq_refbounds_seq
1354 {
1355     \keys_define:nn { zref-clever/langfile }
1356     {
1357         #1 .value_required:n = true ,

```

```

1358 #1 .code:n =
1359 {
1360   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1361   {
1362     \__zrefclever_opt_seq_if_set:cF
1363     {
1364       \__zrefclever_opt_varname_lang_default:enn
1365       { \l__zrefclever_setup_language_tl } {#1} { seq }
1366     }
1367     {
1368       \seq_gclear:N \g_tmpa_seq
1369       \__zrefclever_opt_seq_gset_clist_split:Nn
1370       \g_tmpa_seq {##1}
1371       \bool_lazy_or:nnTF
1372       { \tl_if_empty_p:n {##1} }
1373       {
1374         \int_compare_p:nNn
1375         { \seq_count:N \g_tmpa_seq } = { 4 }
1376       }
1377       {
1378         \__zrefclever_opt_seq_gset_eq:cN
1379         {
1380           \__zrefclever_opt_varname_lang_default:enn
1381           { \l__zrefclever_setup_language_tl }
1382           {#1} { seq }
1383         }
1384         \g_tmpa_seq
1385       }
1386       {
1387         \msg_info:nxxx { zref-clever }
1388         { refbounds-must-be-four }
1389         {#1} { \seq_count:N \g_tmpa_seq }
1390       }
1391     }
1392   }
1393   {
1394     \__zrefclever_opt_seq_if_set:cF
1395     {
1396       \__zrefclever_opt_varname_lang_type:eenn
1397       { \l__zrefclever_setup_language_tl }
1398       { \l__zrefclever_setup_type_tl } {#1} { seq }
1399     }
1400     {
1401       \seq_gclear:N \g_tmpa_seq
1402       \__zrefclever_opt_seq_gset_clist_split:Nn
1403       \g_tmpa_seq {##1}
1404       \bool_lazy_or:nnTF
1405       { \tl_if_empty_p:n {##1} }
1406       {
1407         \int_compare_p:nNn
1408         { \seq_count:N \g_tmpa_seq } = { 4 }
1409       }
1410       {
1411         \__zrefclever_opt_seq_gset_eq:cN

```

```

1412         {
1413             \__zrefclever_opt_varname_lang_type:eenn
1414             { \l__zrefclever_setup_language_tl }
1415             { \l__zrefclever_setup_type_tl }
1416             {#1} { seq }
1417         }
1418         \g_tmpa_seq
1419     }
1420     {
1421         \msg_info:nxxx { zref-clever }
1422         { refbounds-must-be-four }
1423         {#1} { \seq_count:N \g_tmpa_seq }
1424     }
1425 }
1426 }
1427 } ,
1428 }
1429 }
1430 \seq_map_inline:Nn
1431 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1432 {
1433     \keys_define:nn { zref-clever/langfile }
1434     {
1435         #1 .choice: ,
1436         #1 / true .code:n =
1437         {
1438             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1439             {
1440                 \__zrefclever_opt_bool_if_set:cF
1441                 {
1442                     \__zrefclever_opt_varname_lang_default:enn
1443                     { \l__zrefclever_setup_language_tl }
1444                     {#1} { bool }
1445                 }
1446                 {
1447                     \__zrefclever_opt_bool_gset_true:c
1448                     {
1449                         \__zrefclever_opt_varname_lang_default:enn
1450                         { \l__zrefclever_setup_language_tl }
1451                         {#1} { bool }
1452                     }
1453                 }
1454             }
1455         }
1456         \__zrefclever_opt_bool_if_set:cF
1457         {
1458             \__zrefclever_opt_varname_lang_type:eenn
1459             { \l__zrefclever_setup_language_tl }
1460             { \l__zrefclever_setup_type_tl }
1461             {#1} { bool }
1462         }
1463         {
1464             \__zrefclever_opt_bool_gset_true:c
1465             {

```



```

1466         \_zrefclever_opt_varname_lang_type:eenn
1467         { \l__zrefclever_setup_language_tl }
1468         { \l__zrefclever_setup_type_tl }
1469         {#1} { bool }
1470     }
1471 }
1472 }
1473 },
1474 #1 / false .code:n =
1475 {
1476     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1477     {
1478         \_zrefclever_opt_bool_if_set:cF
1479         {
1480             \_zrefclever_opt_varname_lang_default:enn
1481             { \l__zrefclever_setup_language_tl }
1482             {#1} { bool }
1483         }
1484         {
1485             \_zrefclever_opt_bool_gset_false:c
1486             {
1487                 \_zrefclever_opt_varname_lang_default:enn
1488                 { \l__zrefclever_setup_language_tl }
1489                 {#1} { bool }
1490             }
1491         }
1492     }
1493     {
1494         \_zrefclever_opt_bool_if_set:cF
1495         {
1496             \_zrefclever_opt_varname_lang_type:eenn
1497             { \l__zrefclever_setup_language_tl }
1498             { \l__zrefclever_setup_type_tl }
1499             {#1} { bool }
1500         }
1501         {
1502             \_zrefclever_opt_bool_gset_false:c
1503             {
1504                 \_zrefclever_opt_varname_lang_type:eenn
1505                 { \l__zrefclever_setup_language_tl }
1506                 { \l__zrefclever_setup_type_tl }
1507                 {#1} { bool }
1508             }
1509         }
1510     }
1511 },
1512 #1 .default:n = true ,
1513 no #1 .meta:n = { #1 = false } ,
1514 no #1 .value_forbidden:n = true ,
1515 }
1516 }

```

It is convenient for a number of language typesetting options (some basic separators) to have some “fallback” value available in case 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”. Other typesetting options, for which it is not a problem being empty, need not be catered for with a fallback value.

```

1517 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1518 {
1519   \tl_const:cn
1520   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1521 }
1522 \keyval_parse:nnn
1523 { }
1524 { \__zrefclever_opt_tl_cset_fallback:nn }
1525 {
1526   tpairsep = {,~} ,
1527   tlistsep = {,~} ,
1528   tlastsep = {,~} ,
1529   notesep  = {~} ,
1530   namesep  = {\nobreakspace} ,
1531   pairsep  = {,~} ,
1532   listsep  = {,~} ,
1533   lastsep  = {,~} ,
1534   rangesep = {\textendash} ,
1535 }

```

4.8 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 \langle property list \rangle { \langle key \rangle } { \langle value \rangle }

1536 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1537 {
1538   \tl_if_empty:nTF {#3}
1539   { \prop_remove:Nn #1 {#2} }
1540   { \prop_put:Nnn #1 {#2} {#3} }
1541 }

```

(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. Note that 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 (insightful comments by Ulrike Fischer at <https://github.com/ho-tex/zref/issues/13>). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door. We must also control for an empty value, since “empty” passes both `\zref@ifpropundefined` and `\zref@ifrefcontainsprop`.

```

1542 \tl_new:N \l__zrefclever_ref_property_tl
1543 \keys_define:nn { zref-clever/reference }
1544 {
1545   ref .code:n =
1546   {
1547     \tl_if_empty:nTF {#1}
1548     {
1549       \msg_warning:nnn { zref-clever }
1550       { zref-property-undefined } {#1}
1551       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1552     }
1553     {
1554       \zref@ifpropundefined {#1}
1555       {
1556         \msg_warning:nnn { zref-clever }
1557         { zref-property-undefined } {#1}
1558         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1559       }
1560       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1561     }
1562   } ,
1563   ref .initial:n = default ,
1564   ref .value_required:n = true ,
1565   page .meta:n = { ref = page } ,
1566   page .value_forbidden:n = true ,
1567 }

```

typeset option

```

1568 \bool_new:N \l__zrefclever_typeset_ref_bool
1569 \bool_new:N \l__zrefclever_typeset_name_bool
1570 \keys_define:nn { zref-clever/reference }
1571 {
1572   typeset .choice: ,
1573   typeset / both .code:n =
1574   {
1575     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1576     \bool_set_true:N \l__zrefclever_typeset_name_bool
1577   } ,
1578   typeset / ref .code:n =
1579   {
1580     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1581     \bool_set_false:N \l__zrefclever_typeset_name_bool
1582   } ,
1583   typeset / name .code:n =
1584   {
1585     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1586     \bool_set_true:N \l__zrefclever_typeset_name_bool
1587   } ,
1588   typeset .initial:n = both ,
1589   typeset .value_required:n = true ,
1590
1591   noname .meta:n = { typeset = ref } ,
1592   noname .value_forbidden:n = true ,

```

```

1593     noref .meta:n = { typeset = name } ,
1594     noref .value_forbidden:n = true ,
1595 }

```

sort option

```

1596 \bool_new:N \l__zrefclever_typeset_sort_bool
1597 \keys_define:nn { zref-clever/reference }
1598 {
1599     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1600     sort .initial:n = true ,
1601     sort .default:n = true ,
1602     nosort .meta:n = { sort = false },
1603     nosort .value_forbidden:n = true ,
1604 }

```

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`.

```

1605 \seq_new:N \l__zrefclever_typesort_seq
1606 \keys_define:nn { zref-clever/reference }
1607 {
1608     typesort .code:n =
1609     {
1610         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1611         \seq_reverse:N \l__zrefclever_typesort_seq
1612     } ,
1613     typesort .initial:n =
1614     { part , chapter , section , paragraph },
1615     typesort .value_required:n = true ,
1616     notypesort .code:n =
1617     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1618     notypesort .value_forbidden:n = true ,
1619 }

```

comp option

```

1620 \bool_new:N \l__zrefclever_typeset_compress_bool
1621 \keys_define:nn { zref-clever/reference }
1622 {
1623     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1624     comp .initial:n = true ,
1625     comp .default:n = true ,
1626     nocomp .meta:n = { comp = false },
1627     nocomp .value_forbidden:n = true ,
1628 }

```

endrange option

The working of `endrange` option depends on two underlying option values / variables: `endrangefunc` and `endrangeprop`. `endrangefunc` is the more general one,

and `endrangeprop` is used when the first is set to `_zrefclever_get_endrange_property:VVN`, which is the case when the user is setting `endrange` to an arbitrary `zref` property, instead of one of the `\str_case:nn` matches.

`endrangefunc` *must* receive three arguments and, more specifically, its signature *must* be `VVN`. For this reason, `endrangefunc` should be stored without the signature, which is added, and hard-coded, at the calling place. The first argument is *<begin range label>*, the second *<end range label>*, and the last *<tl var to set>*. Of course, *<tl var to set>* must be set to a proper value, and that’s the main task of the function. `endrangefunc` must also handle the case where `\zref@ifrefcontainsprop` is false, since `_zrefclever_get_ref_endrange:nnN` cannot take care of that. For this purpose, it may set *<tl var to set>* to the special value `zc@missingproperty`, to signal a missing property for `_zrefclever_get_ref_endrange:nnN`.

An empty `endrangefunc` signals that no processing is to be made to the end range reference, that is, that it should be treated like any other one, as defined by the `ref` option. This may happen either because `endrange` was never set for the reference type, and empty is the value “returned” by `_zrefclever_get_rf_opt_tl:nnnN` for options not set, or because `endrange` was set to `ref` at some scope which happens to get precedence.

One thing I was divided about in this functionality was whether to (x-)expand the references before processing them, when such processing is required. At first sight, it makes sense to do so, since we are aiming at “removing common parts” as close as possible to the printed representation of the references (`cleveref` does expand them in `\crefstripprefix`). On the other hand, this brings some new challenges: if a fragile command gets there, we are in trouble; also, if a protected one gets there, though things won’t break as badly, we may “strip” the macro and stay with different arguments, which will then end up in the input stream. I think `biblatex` is a good reference here, and it offers `\NumCheckSetup`, `\NumsCheckSetup`, and `\PagesCheckSetup` aimed at locally redefining some commands which may interfere with the processing. This is a good idea, thus we offer a similar hook for the same purpose: `endrange-setup`.

```

1629 \NewHook { zref-clever/endrange-setup }
1630 \keys_define:nn { zref-clever/reference }
1631 {
1632   endrange .code:n =
1633   {
1634     \str_case:nnF {#1}
1635     {
1636       { ref }
1637       {
1638         \_zrefclever_opt_tl_clear:c
1639         {
1640           \_zrefclever_opt_varname_general:nn
1641           { endrangefunc } { tl }
1642         }
1643         \_zrefclever_opt_tl_clear:c
1644         {
1645           \_zrefclever_opt_varname_general:nn
1646           { endrangeprop } { tl }
1647         }
1648       }
1649     }
1650     { stripprefix }
1651     {

```

```

1652 \_zrefclever_opt_tl_set:cn
1653 {
1654   \_zrefclever_opt_varname_general:nn
1655   { endrangefunc } { t1 }
1656 }
1657 { __zrefclever_get_endrange_stripprefix }
1658 \_zrefclever_opt_tl_clear:c
1659 {
1660   \_zrefclever_opt_varname_general:nn
1661   { endrangeprop } { t1 }
1662 }
1663 }
1664
1665 { pagecomp }
1666 {
1667   \_zrefclever_opt_tl_set:cn
1668   {
1669     \_zrefclever_opt_varname_general:nn
1670     { endrangefunc } { t1 }
1671   }
1672   { __zrefclever_get_endrange_pagecomp }
1673   \_zrefclever_opt_tl_clear:c
1674   {
1675     \_zrefclever_opt_varname_general:nn
1676     { endrangeprop } { t1 }
1677   }
1678 }
1679
1680 { pagecomp2 }
1681 {
1682   \_zrefclever_opt_tl_set:cn
1683   {
1684     \_zrefclever_opt_varname_general:nn
1685     { endrangefunc } { t1 }
1686   }
1687   { __zrefclever_get_endrange_pagecomptwo }
1688   \_zrefclever_opt_tl_clear:c
1689   {
1690     \_zrefclever_opt_varname_general:nn
1691     { endrangeprop } { t1 }
1692   }
1693 }
1694
1695 { unset }
1696 {
1697   \_zrefclever_opt_tl_unset:c
1698   {
1699     \_zrefclever_opt_varname_general:nn
1700     { endrangefunc } { t1 }
1701   }
1702   \_zrefclever_opt_tl_unset:c
1703   {
1704     \_zrefclever_opt_varname_general:nn
1705     { endrangeprop } { t1 }

```

```

1706         }
1707     }
1708 }
1709 {
1710     \tl_if_empty:nTF {#1}
1711     {
1712         \msg_warning:nnn { zref-clever }
1713         { endrange-property-undefined } {#1}
1714     }
1715     {
1716         \zref@ifpropundefined {#1}
1717         {
1718             \msg_warning:nnn { zref-clever }
1719             { endrange-property-undefined } {#1}
1720         }
1721         {
1722             \__zrefclever_opt_tl_set:cn
1723             {
1724                 \__zrefclever_opt_varname_general:nn
1725                 { endrangefunc } { tl }
1726             }
1727             { __zrefclever_get_endrange_property }
1728             \__zrefclever_opt_tl_set:cn
1729             {
1730                 \__zrefclever_opt_varname_general:nn
1731                 { endrangeprop } { tl }
1732             }
1733             {#1}
1734         }
1735     }
1736 } ,
1737 endrange .value_required:n = true ,
1738 }
1739
1740 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1741 {
1742     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1743     {
1744         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1745         {
1746             \__zrefclever_extract_default:Nnvn #3
1747             {#2} { \l__zrefclever_ref_property_tl } { }
1748         }
1749         { \tl_set:Nn #3 { zc@missingproperty } }
1750     }
1751     {
1752         \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1753         {

```

If the range came about by normal compression, we already know the beginning and the end references share the same “form” and “prefix” (this is ensured at `__zrefclever_labels_in_sequence:nn`), but the same is not true if the `range` option is being used, in which case, we have to check the replacement `\l__zrefclever_ref_property_tl` by `\l__zrefclever_endrangeprop_tl` is really granted.

```

1754 \bool_if:NTF \l__zrefclever_typeset_range_bool
1755 {
1756   \group_begin:
1757   \bool_set_false:N \l_tmpa_bool
1758   \exp_args:Nxx \tl_if_eq:nnT
1759   {
1760     \__zrefclever_extract_unexp:nnn
1761     {#1} { externaldocument } { }
1762   }
1763   {
1764     \__zrefclever_extract_unexp:nnn
1765     {#2} { externaldocument } { }
1766   }
1767   {
1768     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1769     {
1770       \exp_args:Nxx \tl_if_eq:nnT
1771       {
1772         \__zrefclever_extract_unexp:nnn
1773         {#1} { zc@pgfmt } { }
1774       }
1775       {
1776         \__zrefclever_extract_unexp:nnn
1777         {#2} { zc@pgfmt } { }
1778       }
1779       { \bool_set_true:N \l_tmpa_bool }
1780     }
1781     {
1782       \exp_args:Nxx \tl_if_eq:nnT
1783       {
1784         \__zrefclever_extract_unexp:nnn
1785         {#1} { zc@counter } { }
1786       }
1787       {
1788         \__zrefclever_extract_unexp:nnn
1789         {#2} { zc@counter } { }
1790       }
1791       {
1792         \exp_args:Nxx \tl_if_eq:nnT
1793         {
1794           \__zrefclever_extract_unexp:nnn
1795           {#1} { zc@enclval } { }
1796         }
1797         {
1798           \__zrefclever_extract_unexp:nnn
1799           {#2} { zc@enclval } { }
1800         }
1801         { \bool_set_true:N \l_tmpa_bool }
1802       }
1803     }
1804   }
1805   \bool_if:NTF \l_tmpa_bool
1806   {
1807     \__zrefclever_extract_default:Nnvn \l_tmpb_tl

```



```

1808         {#2} { l__zrefclever_endrangeprop_tl } { }
1809     }
1810     {
1811         \zref@ifrefcontainsprop
1812         {#2} { \l__zrefclever_ref_property_tl }
1813         {
1814             \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1815             {#2} { l__zrefclever_ref_property_tl } { }
1816         }
1817         { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1818     }
1819     \exp_args:NNNV
1820     \group_end:
1821     \tl_set:Nn #3 \l_tmpb_tl
1822 }
1823 {
1824     \__zrefclever_extract_default:Nnvn #3
1825     {#2} { l__zrefclever_endrangeprop_tl } { }
1826 }
1827 }
1828 {
1829     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1830     {
1831         \__zrefclever_extract_default:Nnvn #3
1832         {#2} { l__zrefclever_ref_property_tl } { }
1833     }
1834     { \tl_set:Nn #3 { zc@missingproperty } }
1835 }
1836 }
1837 }
1838 \cs_generate_variant:Nn \__zrefclever_get_endrange_property:nnN { VVN }

```

For the technique for smuggling the assignment out of the group, see Enrico Gregorio's answer at <https://tex.stackexchange.com/a/56314>.

```

1839 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1840 {
1841     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1842     {
1843         \group_begin:
1844         \UseHook { zref-clever/endrange-setup }
1845         \tl_set:Nx \l_tmpa_tl
1846         {
1847             \__zrefclever_extract:nnn
1848             {#1} { \l__zrefclever_ref_property_tl } { }
1849         }
1850         \tl_set:Nx \l_tmpb_tl
1851         {
1852             \__zrefclever_extract:nnn
1853             {#2} { \l__zrefclever_ref_property_tl } { }
1854         }
1855         \bool_set_false:N \l_tmpa_bool
1856         \bool_until_do:Nn \l_tmpa_bool
1857         {
1858             \exp_args:Nxx \tl_if_eq:nnTF

```

```

1859         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1860         {
1861             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1862             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1863             \tl_if_empty:NT \l_tmpb_tl
1864                 { \bool_set_true:N \l_tmpa_bool }
1865         }
1866         { \bool_set_true:N \l_tmpa_bool }
1867     }
1868     \exp_args:NNNV
1869     \group_end:
1870     \tl_set:Nn #3 \l_tmpb_tl
1871 }
1872 { \tl_set:Nn #3 { zc@missingproperty } }
1873 }
1874 \cs_generate_variant:Nn \__zrefclever_get_endrange_stripprefix:nnN { VVN }

```

`__zrefclever_is_integer_rgx:n` Test if argument is composed only of digits (adapted from <https://tex.stackexchange.com/a/427559>).

```

1875 \prg_new_protected_conditional:Npnn
1876   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1877 {
1878     \regex_match:nnTF { \A\d+\Z } {#1}
1879     { \prg_return_true: }
1880     { \prg_return_false: }
1881 }
1882 \prg_generate_conditional_variant:Nnn
1883   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End definition for `__zrefclever_is_integer_rgx:n`.)

```

1884 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1885 {
1886     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1887     {
1888         \group_begin:
1889         \UseHook { zref-clever/endrange-setup }
1890         \tl_set:Nx \l_tmpa_tl
1891         {
1892             \__zrefclever_extract:nnn
1893             {#1} { \l__zrefclever_ref_property_tl } { }
1894         }
1895         \tl_set:Nx \l_tmpb_tl
1896         {
1897             \__zrefclever_extract:nnn
1898             {#2} { \l__zrefclever_ref_property_tl } { }
1899         }
1900         \bool_set_false:N \l_tmpa_bool
1901         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1902         {
1903             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1904             { \bool_set_true:N \l_tmpa_bool }
1905         }
1906         { \bool_set_true:N \l_tmpa_bool }
1907         \bool_until_do:Nn \l_tmpa_bool

```

```

1908     {
1909         \exp_args:Nxx \tl_if_eq:nnTF
1910         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1911         {
1912             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1913             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1914             \tl_if_empty:NT \l_tmpb_tl
1915             { \bool_set_true:N \l_tmpa_bool }
1916         }
1917         { \bool_set_true:N \l_tmpa_bool }
1918     }
1919     \exp_args:NNNV
1920     \group_end:
1921     \tl_set:Nn #3 \l_tmpb_tl
1922 }
1923 { \tl_set:Nn #3 { zc@missingproperty } }
1924 }
1925 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1926 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1927 {
1928     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1929     {
1930         \group_begin:
1931         \UseHook { zref-clever/endrange-setup }
1932         \tl_set:Nx \l_tmpa_tl
1933         {
1934             \__zrefclever_extract:nnn
1935             {#1} { \l__zrefclever_ref_property_tl } { }
1936         }
1937         \tl_set:Nx \l_tmpb_tl
1938         {
1939             \__zrefclever_extract:nnn
1940             {#2} { \l__zrefclever_ref_property_tl } { }
1941         }
1942         \bool_set_false:N \l_tmpa_bool
1943         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1944         {
1945             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1946             { \bool_set_true:N \l_tmpa_bool }
1947         }
1948         { \bool_set_true:N \l_tmpa_bool }
1949         \bool_until_do:Nn \l_tmpa_bool
1950         {
1951             \exp_args:Nxx \tl_if_eq:nnTF
1952             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1953             {
1954                 \bool_lazy_or:nnTF
1955                 { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1956                 { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1957                 {
1958                     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1959                     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1960                 }
1961                 { \bool_set_true:N \l_tmpa_bool }

```

```

1962         }
1963         { \bool_set_true:N \l_tmpa_bool }
1964     }
1965     \exp_args:NNNV
1966     \group_end:
1967     \tl_set:Nn #3 \l_tmpb_tl
1968 }
1969 { \tl_set:Nn #3 { zc@missingproperty } }
1970 }
1971 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomptwo:nnN { VVN }

```

range and rangetopair options

The `rangetopair` option is being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```

1972 \bool_new:N \l__zrefclever_typeset_range_bool
1973 \keys_define:nn { zref-clever/reference }
1974 {
1975     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1976     range .initial:n = false ,
1977     range .default:n = true ,
1978 }

```

cap and capfirst options

The `cap` option is currently being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```

1979 \bool_new:N \l__zrefclever_capfirst_bool
1980 \keys_define:nn { zref-clever/reference }
1981 {
1982     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1983     capfirst .initial:n = false ,
1984     capfirst .default:n = true ,
1985 }

```

abbrev and noabbrevfirst options

The `abbrev` option is currently being handled with other reference format option booleans at `\g__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```

1986 \bool_new:N \l__zrefclever_noabbrev_first_bool
1987 \keys_define:nn { zref-clever/reference }
1988 {
1989     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1990     noabbrevfirst .initial:n = false ,
1991     noabbrevfirst .default:n = true ,
1992 }

```

S option

```

1993 \keys_define:nn { zref-clever/reference }
1994 {
1995     S .meta:n =

```

```

1996     { capfirst = {#1} , noabbrevfirst = {#1} },
1997     S .default:n = true ,
1998 }

```

hyperref option

```

1999 \bool_new:N \l__zrefclever_hyperlink_bool
2000 \bool_new:N \l__zrefclever_hyperref_warn_bool
2001 \keys_define:nn { zref-clever/reference }
2002 {
2003     hyperref .choice: ,
2004     hyperref / auto .code:n =
2005     {
2006         \bool_set_true:N \l__zrefclever_hyperlink_bool
2007         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2008     } ,
2009     hyperref / true .code:n =
2010     {
2011         \bool_set_true:N \l__zrefclever_hyperlink_bool
2012         \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2013     } ,
2014     hyperref / false .code:n =
2015     {
2016         \bool_set_false:N \l__zrefclever_hyperlink_bool
2017         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2018     } ,
2019     hyperref .initial:n = auto ,
2020     hyperref .default:n = true ,

```

`nohyperref` is provided mainly as a means to inhibit hyperlinking locally in `zref-vario`'s commands without the need to be setting `zref-clever`'s internal variables directly. What limits setting `hyperref` out of the preamble is that enabling hyperlinks requires loading packages. But `nohyperref` can only disable them, so we can use it in the document body too.

```

2021     nohyperref .meta:n = { hyperref = false } ,
2022     nohyperref .value_forbidden:n = true ,
2023 }
2024 \AddToHook { begindocument }
2025 {
2026     \__zrefclever_if_package_loaded:nTF { hyperref }
2027     {
2028         \bool_if:NT \l__zrefclever_hyperlink_bool
2029         { \RequirePackage { zref-hyperref } }
2030     }
2031     {
2032         \bool_if:NT \l__zrefclever_hyperref_warn_bool
2033         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2034         \bool_set_false:N \l__zrefclever_hyperlink_bool
2035     }
2036     \keys_define:nn { zref-clever/reference }
2037     {
2038         hyperref .code:n =
2039         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2040         nohyperref .code:n =
2041         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,

```

```

2042     }
2043 }

```

nameinlink option

```

2044 \str_new:N \l__zrefclever_nameinlink_str
2045 \keys_define:nn { zref-clever/reference }
2046 {
2047   nameinlink .choice: ,
2048   nameinlink / true .code:n =
2049     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2050   nameinlink / false .code:n =
2051     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2052   nameinlink / single .code:n =
2053     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2054   nameinlink / tsingle .code:n =
2055     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2056   nameinlink .initial:n = tsingle ,
2057   nameinlink .default:n = true ,
2058 }

```

preposinlink option (deprecated)

```

2059 \keys_define:nn { zref-clever/reference }
2060 {
2061   preposinlink .code:n =
2062   {
2063     % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2064     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2065     { preposinlink } { refbounds }
2066   } ,
2067 }

```

lang option

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 “current” and “main” document languages, this must be retrieved at a `begindocument` hook. The `begindocument` hook is responsible to get values for `\l__zrefclever_current_language_tl` and `\l__zrefclever_main_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 `current` language’s language file gets loaded, if it hadn’t been already.

For the `babel` and `polyglossia` variables which store the “current” and “main” 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`.

```

2068 \AddToHook { begindocument }
2069 {

```

```

2070 \__zrefclever_if_package_loaded:nTF { babel }
2071 {
2072   \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
2073   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2074 }
2075 {
2076   \__zrefclever_if_package_loaded:nTF { polyglossia }
2077   {
2078     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2079     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2080   }
2081   {
2082     \tl_set:Nn \l__zrefclever_current_language_tl { english }
2083     \tl_set:Nn \l__zrefclever_main_language_tl { english }
2084   }
2085 }
2086 }
2087 \keys_define:nn { zref-clever/reference }
2088 {
2089   lang .code:n =
2090   {
2091     \AddToHook { begindocument }
2092     {
2093       \str_case:nnF {#1}
2094       {
2095         { current }
2096         {
2097           \tl_set:Nn \l__zrefclever_ref_language_tl
2098             { \l__zrefclever_current_language_tl }
2099         }
2100
2101         { main }
2102         {
2103           \tl_set:Nn \l__zrefclever_ref_language_tl
2104             { \l__zrefclever_main_language_tl }
2105         }
2106       }
2107       {
2108         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2109         \__zrefclever_language_if_declared:nF {#1}
2110         {
2111           \msg_warning:nnn { zref-clever }
2112             { unknown-language-opt } {#1}
2113         }
2114       }
2115       \__zrefclever_provide_langfile:x
2116         { \l__zrefclever_ref_language_tl }
2117     }
2118   } ,
2119   lang .initial:n = current ,
2120   lang .value_required:n = true ,
2121 }
2122 \AddToHook { begindocument / before }

```

```

2123 {
2124   \AddToHook { begindocument }
2125   {

```

Redefinition of the `lang` key option for the document body. Also, drop the language file loading in the document body, it is somewhat redundant, since `__zrefclever-zcref:nnn` already ensures it.

```

2126     \keys_define:nn { zref-clever/reference }
2127     {
2128       lang .code:n =
2129       {
2130         \str_case:nnF {#1}
2131         {
2132           { current }
2133           {
2134             \tl_set:Nn \l__zrefclever_ref_language_tl
2135             { \l__zrefclever_current_language_tl }
2136           }
2137
2138           { main }
2139           {
2140             \tl_set:Nn \l__zrefclever_ref_language_tl
2141             { \l__zrefclever_main_language_tl }
2142           }
2143         }
2144         {
2145           \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2146           \__zrefclever_language_if_declared:nF {#1}
2147           {
2148             \msg_warning:nnn { zref-clever }
2149             { unknown-language-opt } {#1}
2150           }
2151         }
2152       } ,
2153     }
2154   }
2155 }

```

d option

For setting the declension case. Short for convenience and for not polluting the markup too much given that, for languages that need it, it may get to be used frequently.

‘samcarter’ and Alan Munn provided useful comments about declension on the TeX.SX chat. Also, Florent Rougon’s efforts in this area, with the `xcref` package (<https://github.com/frougon/xcref>), have been an insightful source to frame the problem in general terms.

```

2156 \tl_new:N \l__zrefclever_ref_decl_case_tl
2157 \keys_define:nn { zref-clever/reference }
2158 {
2159   d .code:n =
2160   { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2161 }
2162 \AddToHook { begindocument }

```



```

2163 {
2164   \keys_define:nn { zref-clever/reference }
2165   {

```

We just store the value at this point, which is validated by `__zrefclever_process_language_settings`: after `\keys_set:nn`.

```

2166     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2167     d .value_required:n = true ,
2168   }
2169 }

```

nudge & co. options

```

2170 \bool_new:N \l__zrefclever_nudge_enabled_bool
2171 \bool_new:N \l__zrefclever_nudge_multitype_bool
2172 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2173 \bool_new:N \l__zrefclever_nudge_singular_bool
2174 \bool_new:N \l__zrefclever_nudge_gender_bool
2175 \tl_new:N \l__zrefclever_ref_gender_tl
2176 \keys_define:nn { zref-clever/reference }
2177 {
2178   nudge .choice: ,
2179   nudge / true .code:n =
2180   { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2181   nudge / false .code:n =
2182   { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2183   nudge / ifdraft .code:n =
2184   {
2185     \ifdraft
2186       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2187       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2188     } ,
2189   nudge / iffinaal .code:n =
2190   {
2191     \ifoptionfinal
2192       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2193       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2194     } ,
2195   nudge .initial:n = false ,
2196   nudge .default:n = true ,
2197   nonnudge .meta:n = { nudge = false } ,
2198   nonnudge .value_forbidden:n = true ,
2199   nudgeif .code:n =
2200   {
2201     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2202     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2203     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2204     \clist_map_inline:nn {#1}
2205     {
2206       \str_case:nnF {##1}
2207       {
2208         { multitype }
2209         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2210         { comptosing }

```

```

2211         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2212         { gender }
2213         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2214         { all }
2215         {
2216             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2217             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2218             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2219         }
2220     }
2221     {
2222         \msg_warning:nnn { zref-clever }
2223         { nudgeif-unknown-value } {##1}
2224     }
2225 }
2226 },
2227 nudgeif .value_required:n = true ,
2228 nudgeif .initial:n = all ,
2229 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2230 sg .initial:n = false ,
2231 sg .default:n = true ,
2232 g .code:n =
2233 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2234 }
2235 \AddToHook { begindocument }
2236 {
2237     \keys_define:nn { zref-clever/reference }
2238     {

```

We just store the value at this point, which is validated by `__zrefclever_process_language_settings`: after `\keys_set:nn`.

```

2239         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2240         g .value_required:n = true ,
2241     }
2242 }

```

font option

```

2243 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2244 \keys_define:nn { zref-clever/reference }
2245 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2246 \keys_define:nn { zref-clever/reference }
2247 {
2248     titleref .code:n =
2249     {
2250         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2251         \msg_warning:nnxx { zref-clever } { option-deprecated } { titleref }
2252         { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2253     } ,
2254 }

```

vario option

```

2255 \keys_define:nn { zref-clever/reference }

```

```

2256 {
2257     vario .code:n =
2258     {
2259         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2260         \msg_warning:nnxx { zref-clever }{ option-deprecated } { vario }
2261         { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2262     } ,
2263 }

```

note option

```

2264 \tl_new:N \l__zrefclever_zcref_note_tl
2265 \keys_define:nn { zref-clever/reference }
2266 {
2267     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2268     note .value_required:n = true ,
2269 }

```

check option

Integration with zref-check.

```

2270 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2271 \bool_new:N \l__zrefclever_zcref_with_check_bool
2272 \keys_define:nn { zref-clever/reference }
2273 {
2274     check .code:n =
2275     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2276 }
2277 \AddToHook { begindocument }
2278 {
2279     \__zrefclever_if_package_loaded:nTF { zref-check }
2280     {
2281         \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2282         {
2283             \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2284             \keys_define:nn { zref-clever/reference }
2285             {
2286                 check .code:n =
2287                 {
2288                     \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2289                     \keys_set:nn { zref-check / zcheck } { #1 }
2290                 } ,
2291                 check .value_required:n = true ,
2292             }
2293         }
2294         {
2295             \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2296             \keys_define:nn { zref-clever/reference }
2297             {
2298                 check .code:n =
2299                 {
2300                     \msg_warning:nnn { zref-clever }
2301                     { zref-check-too-old } { 2021-09-16-v0.2.1 }
2302                 } ,
2303             }

```

```

2304     }
2305   }
2306   {
2307     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2308     \keys_define:nn { zref-clever/reference }
2309     {
2310       check .code:n =
2311         { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2312     }
2313   }
2314 }

```

reftype option

This allows one to manually specify the reference type. It is the equivalent of `cleveref`'s optional argument to `\label`.

```

2315 \tl_new:N \l__zrefclever_reftype_override_tl
2316 \keys_define:nn { zref-clever/label }
2317 {
2318   reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2319   reftype .default:n = {} ,
2320   reftype .initial:n = {} ,
2321 }

```

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`.

```

2322 \prop_new:N \l__zrefclever_counter_type_prop
2323 \keys_define:nn { zref-clever/label }
2324 {
2325   countertype .code:n =
2326   {
2327     \keyval_parse:nnn
2328     {
2329       \msg_warning:nnnn { zref-clever }
2330       { key-requires-value } { countertype }
2331     }
2332     {
2333       \__zrefclever_prop_put_non_empty:Nnn
2334       \l__zrefclever_counter_type_prop
2335     }
2336     {#1}
2337   } ,
2338   countertype .value_required:n = true ,
2339   countertype .initial:n =
2340   {
2341     subsection = section ,
2342     subsubsection = section ,
2343     subparagraph = paragraph ,
2344     enumi = item ,

```

```

2345         enumii      = item ,
2346         enumiii     = item ,
2347         enumiv       = item ,
2348         mpfootnote   = footnote ,
2349     } ,
2350 }

```

One interesting comment I received (by Denis Bitouzé, at issue [#1](#)) about the most appropriate type for `paragraph` and `subparagraph` counters was that the reader of the document does not care whether that particular document structure element has been introduced by `\paragraph` or, e.g. by the `\subsubsection` command. This is a difference the author knows, as they’re using L^AT_EX, but to the reader the difference between them is not really relevant, and it may be just confusing to refer to them by different names. In this case the type for `paragraph` and `subparagraph` should just be `section`. I don’t have a strong opinion about this, and the matter was not pursued further. Besides, I presume not many people would set `secnumdepth` so high to start with. But, for the time being, I left the `paragraph` type for them, since there is actually a visual difference to the reader between the `\subsubsection` and `\paragraph` in the standard classes: up to the former, the sectioning commands break a line before the following text, while, from the later on, the sectioning commands and the following text are part of the same line. So, `\paragraph` is actually different from “just a shorter way to write `\subsubsection`”.

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.

```

2351 \seq_new:N \l__zrefclever_counter_resetters_seq
2352 \keys_define:nn { zref-clever/label }
2353 {
2354     counterresetters .code:n =
2355     {
2356         \clist_map_inline:nn {#1}
2357         {
2358             \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2359             {
2360                 \seq_put_right:Nn
2361                 \l__zrefclever_counter_resetters_seq {##1}
2362             }
2363         }
2364     } ,
2365     counterresetters .initial:n =
2366     {
2367         part ,
2368         chapter ,
2369         section ,
2370         subsection ,
2371         subsubsection ,

```

```

2372     paragraph ,
2373     subparagraph ,
2374   },
2375   counterresetters .value_required:n = true ,
2376 }

```

counterresetby option

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

```

2377 \prop_new:N \l__zrefclever_counter_resetby_prop
2378 \keys_define:nn { zref-clever/label }
2379 {
2380   counterresetby .code:n =
2381   {
2382     \keyval_parse:nnn
2383     {
2384       \msg_warning:nnn { zref-clever }
2385       { key-requires-value } { counterresetby }
2386     }
2387     {
2388       \__zrefclever_prop_put_non_empty:Nnn
2389       \l__zrefclever_counter_resetby_prop
2390     }
2391     {#1}
2392   } ,
2393   counterresetby .value_required:n = true ,
2394   counterresetby .initial:n =
2395   {

```

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.

```

2396   enumii = enumi ,
2397   enumiii = enumii ,
2398   enumiv = enumiii ,
2399 } ,
2400 }

```

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.

```

2401 \tl_new:N \l__zrefclever_current_counter_tl
2402 \keys_define:nn { zref-clever/label }
2403 {
2404   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2405   currentcounter .default:n = \@currentcounter ,

```

```

2406     currentcounter .initial:n = \@currentcounter ,
2407 }

```

nocompat option

```

2408 \bool_new:N \g__zrefclever_nocompat_bool
2409 \seq_new:N \g__zrefclever_nocompat_modules_seq
2410 \keys_define:nn { zref-clever/reference }
2411 {
2412     nocompat .code:n =
2413     {
2414         \tl_if_empty:nTF {#1}
2415         { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2416         {
2417             \clist_map_inline:nn {#1}
2418             {
2419                 \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2420                 {
2421                     \seq_gput_right:Nn
2422                     \g__zrefclever_nocompat_modules_seq {##1}
2423                 }
2424             }
2425         }
2426     } ,
2427 }
2428 \AddToHook { begindocument }
2429 {
2430     \keys_define:nn { zref-clever/reference }
2431     {
2432         nocompat .code:n =
2433         {
2434             \msg_warning:nnn { zref-clever }
2435             { option-preamble-only } { nocompat }
2436         }
2437     }
2438 }
2439 \AtEndOfPackage
2440 {
2441     \AddToHook { begindocument }
2442     {
2443         \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2444         { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2445     }
2446 }

```

_zrefclever_compat_module:nn

Function to be used for compatibility modules loading. It should load the module as long as `\l__zrefclever_nocompat_bool` is false and `\l__zrefclever_nocompat_modules_seq` is not in `\l__zrefclever_nocompat_modules_seq`. The `begindocument` hook is needed so that we can have the option functional along the whole preamble, not just at package load time. This requirement might be relaxed if we made the option only available at load time, but this would not buy us much leeway anyway, since for most compatibility modules, we must test for the presence of packages at `begindocument`, only kernel features and document classes could be checked reliably before that. Besides, since we are using the new hook

management system, there is always its functionality to deal with potential loading order issues.

```

2447 \__zrefclever_compat_module:nn {<module>} {<code>}
2448 {
2449   \AddToHook { begindocument }
2450   {
2451     \bool_if:NF \g__zrefclever_nocompat_bool
2452     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2453     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2454   }
2455 }

```

(End definition for __zrefclever_compat_module:nn.)

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 \zceref or to \zcsetup or at load time, only “not necessarily type-specific” options are pertinent here.

```

2456 \seq_map_inline:Nn
2457 \g__zrefclever_rf_opts_tl_reference_seq
2458 {
2459   \keys_define:nn { zref-clever/reference }
2460   {
2461     #1 .default:o = \c_novalue_tl ,
2462     #1 .code:n =
2463     {
2464       \tl_if_novalue:nTF {##1}
2465       {
2466         \__zrefclever_opt_tl_unset:c
2467         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2468       }
2469       {
2470         \__zrefclever_opt_tl_set:cn
2471         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2472         {##1}
2473       }
2474     } ,
2475   }
2476 }
2477 \keys_define:nn { zref-clever/reference }
2478 {
2479   refpre .code:n =
2480   {
2481     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2482     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2483     { refpre } { refbounds }
2484   } ,
2485   refpos .code:n =
2486   {

```



```

2487         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2488         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2489         { refpos } { refbounds }
2490     } ,
2491     preref .code:n =
2492     {
2493         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2494         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2495         { preref } { refbounds }
2496     } ,
2497     postref .code:n =
2498     {
2499         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2500         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2501         { postref } { refbounds }
2502     } ,
2503 }
2504 \seq_map_inline:Nn
2505   \g__zrefclever_rf_opts_seq_refbounds_seq
2506   {
2507     \keys_define:nn { zref-clever/reference }
2508     {
2509       #1 .default:o = \c_novalue_tl ,
2510       #1 .code:n =
2511       {
2512         \tl_if_novalue:nTF {##1}
2513         {
2514           \__zrefclever_opt_seq_unset:c
2515           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2516         }
2517         {
2518           \seq_clear:N \l_tmpa_seq
2519           \__zrefclever_opt_seq_set_clist_split:Nn
2520           \l_tmpa_seq {##1}
2521           \bool_lazy_or:nnTF
2522           { \tl_if_empty_p:n {##1} }
2523           { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2524           {
2525             \__zrefclever_opt_seq_set_eq:cN
2526             { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2527             \l_tmpa_seq
2528           }
2529           {
2530             \msg_warning:nxxx { zref-clever }
2531             { refbounds-must-be-four }
2532             {#1} { \seq_count:N \l_tmpa_seq }
2533           }
2534         }
2535       } ,
2536     }
2537 }
2538 \seq_map_inline:Nn
2539   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2540   {

```

```

2541 \keys_define:nn { zref-clever/reference }
2542 {
2543   #1 .choice: ,
2544   #1 / true .code:n =
2545   {
2546     \__zrefclever_opt_bool_set_true:c
2547     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2548   } ,
2549   #1 / false .code:n =
2550   {
2551     \__zrefclever_opt_bool_set_false:c
2552     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2553   } ,
2554   #1 / unset .code:n =
2555   {
2556     \__zrefclever_opt_bool_unset:c
2557     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2558   } ,
2559   #1 .default:n = true ,
2560   no #1 .meta:n = { #1 = false } ,
2561   no #1 .value_forbidden:n = true ,
2562 }
2563 }

```

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 `\zceref`'s options. Anyway, for package options (`\zcsetup`) we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```

2564 \keys_define:nn { }
2565 {
2566   zref-clever/zcsetup .inherit:n =
2567   {
2568     zref-clever/label ,
2569     zref-clever/reference ,
2570   }
2571 }

```

`zref-clever` does not accept load-time options. Despite the tradition of so doing, Joseph Wright has a point in recommending otherwise at <https://chat.stackexchange.com/transcript/message/60360822#60360822>: separating “loading the package” from “configuring the package” grants less trouble with “option clashes” and with expansion of options at load-time.

```

2572 \bool_lazy_and:nnT
2573 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2574 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2575 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```
\zcsetup{<options>}

2576 \NewDocumentCommand \zcsetup { m }
2577 { \__zrefclever_zcsetup:n {#1} }
```

(End definition for `\zcsetup`.)

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

```
\__zrefclever_zcsetup:n{<options>}

2578 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2579 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2580 \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 language-specific setting, either done at `\zcLanguageSetup` or by the package’s language files. 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>}

2581 \NewDocumentCommand \zcRefTypeSetup { m m }
2582 {
2583   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2584   \keys_set:nn { zref-clever/typesetup } {#2}
2585   \tl_clear:N \l__zrefclever_setup_type_tl
2586 }

(End definition for \zcRefTypeSetup.)

2587 \seq_map_inline:Nn
2588 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2589 {
2590   \keys_define:nn { zref-clever/typesetup }
2591   {
2592     #1 .code:n =
2593     {
2594       \msg_warning:nnn { zref-clever }
2595       { option-not-type-specific } {#1}
2596     } ,
2597   }
2598 }
2599 \seq_map_inline:Nn
```

```

2600 \g__zrefclever_rf_opts_tl_typesetup_seq
2601 {
2602   \keys_define:nn { zref-clever/typesetup }
2603   {
2604     #1 .default:o = \c_novalue_tl ,
2605     #1 .code:n =
2606     {
2607       \tl_if_novalue:nTF {##1}
2608       {
2609         \__zrefclever_opt_tl_unset:c
2610         {
2611           \__zrefclever_opt_varname_type:enn
2612           { \l__zrefclever_setup_type_tl } {#1} { t1 }
2613         }
2614       }
2615       {
2616         \__zrefclever_opt_tl_set:cn
2617         {
2618           \__zrefclever_opt_varname_type:enn
2619           { \l__zrefclever_setup_type_tl } {#1} { t1 }
2620         }
2621         {##1}
2622       }
2623     } ,
2624   }
2625 }
2626 \keys_define:nn { zref-clever/typesetup }
2627 {
2628   endrange .code:n =
2629   {
2630     \str_case:nnF {#1}
2631     {
2632       { ref }
2633       {
2634         \__zrefclever_opt_tl_clear:c
2635         {
2636           \__zrefclever_opt_varname_type:enn
2637           { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2638         }
2639         \__zrefclever_opt_tl_clear:c
2640         {
2641           \__zrefclever_opt_varname_type:enn
2642           { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2643         }
2644       }
2645     }
2646     { stripprefix }
2647     {
2648       \__zrefclever_opt_tl_set:cn
2649       {
2650         \__zrefclever_opt_varname_type:enn
2651         { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2652       }
2653       { __zrefclever_get_endrange_stripprefix }

```

```

2654     \_zrefclever_opt_t1_clear:c
2655     {
2656         \_zrefclever_opt_varname_type:enn
2657         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2658     }
2659 }
2660
2661 { pagecomp }
2662 {
2663     \_zrefclever_opt_t1_set:cn
2664     {
2665         \_zrefclever_opt_varname_type:enn
2666         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2667     }
2668     { \_zrefclever_get_endrange_pagecomp }
2669     \_zrefclever_opt_t1_clear:c
2670     {
2671         \_zrefclever_opt_varname_type:enn
2672         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2673     }
2674 }
2675
2676 { pagecomp2 }
2677 {
2678     \_zrefclever_opt_t1_set:cn
2679     {
2680         \_zrefclever_opt_varname_type:enn
2681         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2682     }
2683     { \_zrefclever_get_endrange_pagecomptwo }
2684     \_zrefclever_opt_t1_clear:c
2685     {
2686         \_zrefclever_opt_varname_type:enn
2687         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2688     }
2689 }
2690
2691 { unset }
2692 {
2693     \_zrefclever_opt_t1_unset:c
2694     {
2695         \_zrefclever_opt_varname_type:enn
2696         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2697     }
2698     \_zrefclever_opt_t1_unset:c
2699     {
2700         \_zrefclever_opt_varname_type:enn
2701         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2702     }
2703 }
2704 }
2705 {
2706     \tl_if_empty:nTF {#1}
2707     {

```

```

2708         \msg_warning:nnn { zref-clever }
2709         { endrange-property-undefined } {#1}
2710     }
2711     {
2712         \zref@ifpropundefined {#1}
2713         {
2714             \msg_warning:nnn { zref-clever }
2715             { endrange-property-undefined } {#1}
2716         }
2717         {
2718             \__zrefclever_opt_tl_set:cn
2719             {
2720                 \__zrefclever_opt_varname_type:enn
2721                 { \l__zrefclever_setup_type_tl }
2722                 { endrangefunc } { tl }
2723             }
2724             { __zrefclever_get_endrange_property }
2725             \__zrefclever_opt_tl_set:cn
2726             {
2727                 \__zrefclever_opt_varname_type:enn
2728                 { \l__zrefclever_setup_type_tl }
2729                 { endrangeprop } { tl }
2730             }
2731             {#1}
2732         }
2733     }
2734 }
2735 },
2736 endrange .value_required:n = true ,
2737 }
2738 \keys_define:nn { zref-clever/typesetup }
2739 {
2740     refpre .code:n =
2741     {
2742         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2743         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2744         { refpre } { refbounds }
2745     } ,
2746     refpos .code:n =
2747     {
2748         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2749         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2750         { refpos } { refbounds }
2751     } ,
2752     preref .code:n =
2753     {
2754         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2755         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2756         { preref } { refbounds }
2757     } ,
2758     postref .code:n =
2759     {
2760         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2761         \msg_warning:nnnn { zref-clever }{ option-deprecated }

```

```

2762         { postref } { refbounds }
2763     } ,
2764 }
2765 \seq_map_inline:Nn
2766 \g__zrefclever_rf_opts_seq_refbounds_seq
2767 {
2768     \keys_define:nn { zref-clever/typesetup }
2769     {
2770         #1 .default:o = \c_novalue_tl ,
2771         #1 .code:n =
2772         {
2773             \tl_if_novalue:nTF {##1}
2774             {
2775                 \__zrefclever_opt_seq_unset:c
2776                 {
2777                     \__zrefclever_opt_varname_type:enn
2778                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2779                 }
2780             }
2781             {
2782                 \seq_clear:N \l_tmpa_seq
2783                 \__zrefclever_opt_seq_set_clist_split:Nn
2784                 \l_tmpa_seq {##1}
2785                 \bool_lazy_or:nnTF
2786                 { \tl_if_empty_p:n {##1} }
2787                 { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2788                 {
2789                     \__zrefclever_opt_seq_set_eq:cN
2790                     {
2791                         \__zrefclever_opt_varname_type:enn
2792                         { \l__zrefclever_setup_type_tl } {#1} { seq }
2793                     }
2794                     \l_tmpa_seq
2795                 }
2796                 {
2797                     \msg_warning:nxxx { zref-clever }
2798                     { refbounds-must-be-four }
2799                     {#1} { \seq_count:N \l_tmpa_seq }
2800                 }
2801             }
2802         } ,
2803     }
2804 }
2805 \seq_map_inline:Nn
2806 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2807 {
2808     \keys_define:nn { zref-clever/typesetup }
2809     {
2810         #1 .choice: ,
2811         #1 / true .code:n =
2812         {
2813             \__zrefclever_opt_bool_set_true:c
2814             {
2815                 \__zrefclever_opt_varname_type:enn

```

```

2816         { \l__zrefclever_setup_type_t1 }
2817         {#1} { bool }
2818     }
2819 },
2820 #1 / false .code:n =
2821 {
2822     \__zrefclever_opt_bool_set_false:c
2823     {
2824         \__zrefclever_opt_varname_type:enn
2825         { \l__zrefclever_setup_type_t1 }
2826         {#1} { bool }
2827     }
2828 },
2829 #1 / unset .code:n =
2830 {
2831     \__zrefclever_opt_bool_unset:c
2832     {
2833         \__zrefclever_opt_varname_type:enn
2834         { \l__zrefclever_setup_type_t1 }
2835         {#1} { bool }
2836     }
2837 },
2838 #1 .default:n = true ,
2839 no #1 .meta:n = { #1 = false } ,
2840 no #1 .value_forbidden:n = true ,
2841 }
2842 }

```

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) language options. When the `type` key is given with a value, the options following it will set “type-specific” language options for that type. The current type can be switched off by an empty `type` key. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
2843 \NewDocumentCommand \zcLanguageSetup { m m }
2844 {
2845     \group_begin:
2846     \__zrefclever_language_if_declared:nTF {#1}
2847     {
2848         \tl_clear:N \l__zrefclever_setup_type_t1
2849         \tl_set:Nn \l__zrefclever_setup_language_t1 {#1}
2850         \__zrefclever_opt_seq_get:cNF
2851         {
2852             \__zrefclever_opt_varname_language:nnn
2853             {#1} { declension } { seq }
2854         }
2855         \l__zrefclever_lang_declension_seq
2856         { \seq_clear:N \l__zrefclever_lang_declension_seq }

```



```

2857 \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2858 { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2859 {
2860   \seq_get_left:NN \l__zrefclever_lang_declension_seq
2861   \l__zrefclever_lang_decl_case_tl
2862 }
2863 \__zrefclever_opt_seq_get:cNF
2864 {
2865   \__zrefclever_opt_varname_language:nnn
2866   {#1} { gender } { seq }
2867 }
2868 \l__zrefclever_lang_gender_seq
2869 { \seq_clear:N \l__zrefclever_lang_gender_seq }
2870 \keys_set:nn { zref-clever/langsetup } {#2}
2871 }
2872 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2873 \group_end:
2874 }
2875 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2876 \keys_define:nn { zref-clever/langsetup }
2877 {
2878   type .code:n =
2879   {
2880     \tl_if_empty:NTF {#1}
2881     { \tl_clear:N \l__zrefclever_setup_type_tl }
2882     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2883   } ,
2884
2885   case .code:n =
2886   {
2887     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2888     {
2889       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2890       { \l__zrefclever_setup_language_tl } {#1}
2891     }
2892     {
2893       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2894       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2895       {
2896         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2897         {#1} { \l__zrefclever_setup_language_tl }
2898         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2899         \l__zrefclever_lang_decl_case_tl
2900       }
2901     }
2902   } ,
2903   case .value_required:n = true ,
2904
2905   gender .value_required:n = true ,
2906   gender .code:n =

```

```

2907 {
2908     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2909     {
2910         \msg_warning:nnxxx { zref-clever } { language-no-gender }
2911         { \l__zrefclever_setup_language_tl } { gender } {#1}
2912     }
2913     {
2914         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2915         {
2916             \msg_warning:nnn { zref-clever }
2917             { option-only-type-specific } { gender }
2918         }
2919         {
2920             \seq_clear:N \l_tmpa_seq
2921             \clist_map_inline:nn {#1}
2922             {
2923                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2924                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
2925                 {
2926                     \msg_warning:nnxx { zref-clever }
2927                     { gender-not-declared }
2928                     { \l__zrefclever_setup_language_tl } {##1}
2929                 }
2930             }
2931             \__zrefclever_opt_seq_gset_eq:cn
2932             {
2933                 \__zrefclever_opt_varname_lang_type:eenn
2934                 { \l__zrefclever_setup_language_tl }
2935                 { \l__zrefclever_setup_type_tl }
2936                 { gender }
2937                 { seq }
2938             }
2939             \l_tmpa_seq
2940         }
2941     } ,
2942 } ,
2943 }
2944 \seq_map_inline:Nn
2945 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2946 {
2947     \keys_define:nn { zref-clever/langsetup }
2948     {
2949         #1 .value_required:n = true ,
2950         #1 .code:n =
2951         {
2952             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2953             {
2954                 \__zrefclever_opt_tl_gset:cn
2955                 {
2956                     \__zrefclever_opt_varname_lang_default:enn
2957                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2958                 }
2959                 {##1}
2960             }

```

```

2961         {
2962             \msg_warning:nnn { zref-clever }
2963             { option-not-type-specific } {#1}
2964         }
2965     } ,
2966 }
2967 }
2968 \seq_map_inline:Nn
2969 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2970 {
2971     \keys_define:nn { zref-clever/langsetup }
2972     {
2973         #1 .value_required:n = true ,
2974         #1 .code:n =
2975         {
2976             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2977             {
2978                 \__zrefclever_opt_tl_gset:cn
2979                 {
2980                     \__zrefclever_opt_varname_lang_default:enn
2981                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2982                 }
2983                 {##1}
2984             }
2985             {
2986                 \__zrefclever_opt_tl_gset:cn
2987                 {
2988                     \__zrefclever_opt_varname_lang_type:eenn
2989                     { \l__zrefclever_setup_language_tl }
2990                     { \l__zrefclever_setup_type_tl }
2991                     {#1} { tl }
2992                 }
2993                 {##1}
2994             }
2995         } ,
2996     }
2997 }
2998 \keys_define:nn { zref-clever/langsetup }
2999 {
3000     endrange .value_required:n = true ,
3001     endrange .code:n =
3002     {
3003         \str_case:nnF {#1}
3004         {
3005             { ref }
3006             {
3007                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3008                 {
3009                     \__zrefclever_opt_tl_gclear:c
3010                     {
3011                         \__zrefclever_opt_varname_lang_default:enn
3012                         { \l__zrefclever_setup_language_tl }
3013                         { endrangefunc } { tl }
3014                     }

```

```

3015         \_zrefclever_opt_tl_gclear:c
3016         {
3017             \_zrefclever_opt_varname_lang_default:enn
3018             { \l_zrefclever_setup_language_tl }
3019             { endrangeprop } { tl }
3020         }
3021     }
3022     {
3023         \_zrefclever_opt_tl_gclear:c
3024         {
3025             \_zrefclever_opt_varname_lang_type:enn
3026             { \l_zrefclever_setup_language_tl }
3027             { \l_zrefclever_setup_type_tl }
3028             { endrangefunc } { tl }
3029         }
3030         \_zrefclever_opt_tl_gclear:c
3031         {
3032             \_zrefclever_opt_varname_lang_type:enn
3033             { \l_zrefclever_setup_language_tl }
3034             { \l_zrefclever_setup_type_tl }
3035             { endrangeprop } { tl }
3036         }
3037     }
3038 }
3039
3040 { stripprefix }
3041 {
3042     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3043     {
3044         \_zrefclever_opt_tl_gset:cn
3045         {
3046             \_zrefclever_opt_varname_lang_default:enn
3047             { \l_zrefclever_setup_language_tl }
3048             { endrangefunc } { tl }
3049         }
3050         { _zrefclever_get_endrange_stripprefix }
3051         \_zrefclever_opt_tl_gclear:c
3052         {
3053             \_zrefclever_opt_varname_lang_default:enn
3054             { \l_zrefclever_setup_language_tl }
3055             { endrangeprop } { tl }
3056         }
3057     }
3058     {
3059         \_zrefclever_opt_tl_gset:cn
3060         {
3061             \_zrefclever_opt_varname_lang_type:enn
3062             { \l_zrefclever_setup_language_tl }
3063             { \l_zrefclever_setup_type_tl }
3064             { endrangefunc } { tl }
3065         }
3066         { _zrefclever_get_endrange_stripprefix }
3067         \_zrefclever_opt_tl_gclear:c
3068         {

```

```

3069         \_zrefclever_opt_varname_lang_type:enn
3070         { \l_zrefclever_setup_language_tl }
3071         { \l_zrefclever_setup_type_tl }
3072         { endrangeprop } { tl }
3073     }
3074 }
3075 }
3076
3077 { pagecomp }
3078 {
3079     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3080     {
3081         \_zrefclever_opt_tl_gset:cn
3082         {
3083             \_zrefclever_opt_varname_lang_default:enn
3084             { \l_zrefclever_setup_language_tl }
3085             { endrangefunc } { tl }
3086         }
3087         { _zrefclever_get_endrange_pagecomp }
3088         \_zrefclever_opt_tl_gclear:c
3089         {
3090             \_zrefclever_opt_varname_lang_default:enn
3091             { \l_zrefclever_setup_language_tl }
3092             { endrangeprop } { tl }
3093         }
3094     }
3095     {
3096         \_zrefclever_opt_tl_gset:cn
3097         {
3098             \_zrefclever_opt_varname_lang_type:enn
3099             { \l_zrefclever_setup_language_tl }
3100             { \l_zrefclever_setup_type_tl }
3101             { endrangefunc } { tl }
3102         }
3103         { _zrefclever_get_endrange_pagecomp }
3104         \_zrefclever_opt_tl_gclear:c
3105         {
3106             \_zrefclever_opt_varname_lang_type:enn
3107             { \l_zrefclever_setup_language_tl }
3108             { \l_zrefclever_setup_type_tl }
3109             { endrangeprop } { tl }
3110         }
3111     }
3112 }
3113
3114 { pagecomp2 }
3115 {
3116     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3117     {
3118         \_zrefclever_opt_tl_gset:cn
3119         {
3120             \_zrefclever_opt_varname_lang_default:enn
3121             { \l_zrefclever_setup_language_tl }
3122             { endrangefunc } { tl }

```

```

3123     }
3124     { __zrefclever_get_endrange_pagecomptwo }
3125     \__zrefclever_opt_tl_gclear:c
3126     {
3127         \__zrefclever_opt_varname_lang_default:enn
3128         { \l__zrefclever_setup_language_tl }
3129         { endrangeprop } { tl }
3130     }
3131 }
3132 {
3133     \__zrefclever_opt_tl_gset:cn
3134     {
3135         \__zrefclever_opt_varname_lang_type:enn
3136         { \l__zrefclever_setup_language_tl }
3137         { \l__zrefclever_setup_type_tl }
3138         { endrangefunc } { tl }
3139     }
3140     { __zrefclever_get_endrange_pagecomptwo }
3141     \__zrefclever_opt_tl_gclear:c
3142     {
3143         \__zrefclever_opt_varname_lang_type:enn
3144         { \l__zrefclever_setup_language_tl }
3145         { \l__zrefclever_setup_type_tl }
3146         { endrangeprop } { tl }
3147     }
3148 }
3149 }
3150 }
3151 {
3152     \tl_if_empty:nTF {#1}
3153     {
3154         \msg_warning:nnn { zref-clever }
3155         { endrange-property-undefined } {#1}
3156     }
3157     {
3158         \zref@ifpropundefined {#1}
3159         {
3160             \msg_warning:nnn { zref-clever }
3161             { endrange-property-undefined } {#1}
3162         }
3163         {
3164             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3165             {
3166                 \__zrefclever_opt_tl_gset:cn
3167                 {
3168                     \__zrefclever_opt_varname_lang_default:enn
3169                     { \l__zrefclever_setup_language_tl }
3170                     { endrangefunc } { tl }
3171                 }
3172                 { __zrefclever_get_endrange_property }
3173                 \__zrefclever_opt_tl_gset:cn
3174                 {
3175                     \__zrefclever_opt_varname_lang_default:enn
3176                     { \l__zrefclever_setup_language_tl }

```

```

3177         { endrangeprop } { t1 }
3178     }
3179     {#1}
3180 }
3181 {
3182     \__zrefclever_opt_tl_gset:cn
3183     {
3184         \__zrefclever_opt_varname_lang_type:eenn
3185         { \l__zrefclever_setup_language_tl }
3186         { \l__zrefclever_setup_type_tl }
3187         { endrangefunc } { t1 }
3188     }
3189     { __zrefclever_get_endrange_property }
3190     \__zrefclever_opt_tl_gset:cn
3191     {
3192         \__zrefclever_opt_varname_lang_type:eenn
3193         { \l__zrefclever_setup_language_tl }
3194         { \l__zrefclever_setup_type_tl }
3195         { endrangeprop } { t1 }
3196     }
3197     {#1}
3198 }
3199 }
3200 }
3201 }
3202 } ,
3203 }
3204 \keys_define:nn { zref-clever/langsetup }
3205 {
3206     refpre .code:n =
3207     {
3208         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3209         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3210         { refpre } { refbounds }
3211     } ,
3212     refpos .code:n =
3213     {
3214         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3215         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3216         { refpos } { refbounds }
3217     } ,
3218     preref .code:n =
3219     {
3220         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3221         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3222         { preref } { refbounds }
3223     } ,
3224     postref .code:n =
3225     {
3226         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3227         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3228         { postref } { refbounds }
3229     } ,
3230 }

```

```

3231 \seq_map_inline:Nn
3232 \g__zrefclever_rf_opts_tl_type_names_seq
3233 {
3234   \keys_define:nn { zref-clever/langsetup }
3235   {
3236     #1 .value_required:n = true ,
3237     #1 .code:n =
3238     {
3239       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3240       {
3241         \msg_warning:nnn { zref-clever }
3242         { option-only-type-specific } {#1}
3243       }
3244       {
3245         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3246         {
3247           \__zrefclever_opt_tl_gset:cn
3248           {
3249             \__zrefclever_opt_varname_lang_type:eeenn
3250             { \l__zrefclever_setup_language_tl }
3251             { \l__zrefclever_setup_type_tl }
3252             {#1} { tl }
3253           }
3254           {##1}
3255         }
3256         {
3257           \__zrefclever_opt_tl_gset:cn
3258           {
3259             \__zrefclever_opt_varname_lang_type:eeenn
3260             { \l__zrefclever_setup_language_tl }
3261             { \l__zrefclever_setup_type_tl }
3262             { \l__zrefclever_lang_decl_case_tl - #1 }
3263             { tl }
3264           }
3265           {##1}
3266         }
3267       }
3268     } ,
3269   }
3270 }
3271 \seq_map_inline:Nn
3272 \g__zrefclever_rf_opts_seq_refbounds_seq
3273 {
3274   \keys_define:nn { zref-clever/langsetup }
3275   {
3276     #1 .value_required:n = true ,
3277     #1 .code:n =
3278     {
3279       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3280       {
3281         \seq_gclear:N \g_tmpa_seq
3282         \__zrefclever_opt_seq_gset_clist_split:Nn
3283         \g_tmpa_seq {##1}
3284         \bool_lazy_or:nnTF

```



```

3285         { \tl_if_empty_p:n {##1} }
3286     {
3287         \int_compare_p:nNn
3288             { \seq_count:N \g_tmpa_seq } = { 4 }
3289     }
3290     {
3291         \__zrefclever_opt_seq_gset_eq:cN
3292         {
3293             \__zrefclever_opt_varname_lang_default:enn
3294             { \l__zrefclever_setup_language_tl }
3295             {#1} { seq }
3296         }
3297         \g_tmpa_seq
3298     }
3299     {
3300         \msg_warning:nxxx { zref-clever }
3301         { refbounds-must-be-four }
3302         {#1} { \seq_count:N \g_tmpa_seq }
3303     }
3304 }
3305 {
3306     \seq_gclear:N \g_tmpa_seq
3307     \__zrefclever_opt_seq_gset_clist_split:Nn
3308         \g_tmpa_seq {##1}
3309     \bool_lazy_or:nnTF
3310         { \tl_if_empty_p:n {##1} }
3311         {
3312             \int_compare_p:nNn
3313                 { \seq_count:N \g_tmpa_seq } = { 4 }
3314         }
3315         {
3316             \__zrefclever_opt_seq_gset_eq:cN
3317             {
3318                 \__zrefclever_opt_varname_lang_type:eenn
3319                 { \l__zrefclever_setup_language_tl }
3320                 { \l__zrefclever_setup_type_tl } {#1} { seq }
3321             }
3322             \g_tmpa_seq
3323         }
3324         {
3325             \msg_warning:nxxx { zref-clever }
3326             { refbounds-must-be-four }
3327             {#1} { \seq_count:N \g_tmpa_seq }
3328         }
3329     }
3330 } ,
3331 }
3332 }
3333 \seq_map_inline:Nn
3334     \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3335     {
3336         \keys_define:nn { zref-clever/langsetup }
3337         {
3338             #1 .choice: ,

```

```

3339     #1 / true .code:n =
3340     {
3341         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3342         {
3343             \__zrefclever_opt_bool_gset_true:c
3344             {
3345                 \__zrefclever_opt_varname_lang_default:enn
3346                 { \l__zrefclever_setup_language_tl }
3347                 {#1} { bool }
3348             }
3349         }
3350         {
3351             \__zrefclever_opt_bool_gset_true:c
3352             {
3353                 \__zrefclever_opt_varname_lang_type:enn
3354                 { \l__zrefclever_setup_language_tl }
3355                 { \l__zrefclever_setup_type_tl }
3356                 {#1} { bool }
3357             }
3358         }
3359     } ,
3360     #1 / false .code:n =
3361     {
3362         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3363         {
3364             \__zrefclever_opt_bool_gset_false:c
3365             {
3366                 \__zrefclever_opt_varname_lang_default:enn
3367                 { \l__zrefclever_setup_language_tl }
3368                 {#1} { bool }
3369             }
3370         }
3371         {
3372             \__zrefclever_opt_bool_gset_false:c
3373             {
3374                 \__zrefclever_opt_varname_lang_type:enn
3375                 { \l__zrefclever_setup_language_tl }
3376                 { \l__zrefclever_setup_type_tl }
3377                 {#1} { bool }
3378             }
3379         }
3380     } ,
3381     #1 .default:n = true ,
3382     no #1 .meta:n = { #1 = false } ,
3383     no #1 .value_forbidden:n = true ,
3384 }
3385 }

```

6 User interface

6.1 \zcref

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

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

```
3386 \NewDocumentCommand \zcref { s O { } m }
3387 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }
```

(End definition for `\zcref`.)

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

`__zrefclever_zcref:nnnn {<labels>} {<*>} {<options>}`

```
3388 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3389 {
3390   \group_begin:
```

Set options.

```
3391   \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```
3392   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3393   \bool_set:Nn \l__zrefclever_link_star_bool {#2}
```

Ensure language file 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_langfile:x` does nothing if the language file is already loaded.

```
3394   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3395   \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3396   \bool_lazy_and:nnT
3397   { \l__zrefclever_zrefcheck_available_bool }
3398   { \l__zrefclever_zcref_with_check_bool }
3399   { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3400   \bool_lazy_or:nnT
3401   { \l__zrefclever_typeset_sort_bool }
3402   { \l__zrefclever_typeset_range_bool }
3403   { \__zrefclever_sort_labels: }
```

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

```
3404   \group_begin:
3405   \l__zrefclever_ref_typeset_font_tl
3406   \__zrefclever_typeset_refs:
3407   \group_end:
```

Typeset note.

```
3408   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3409   {
3410     \__zrefclever_get_rf_opt_tl:nxxN { noteseq }
3411     { \l__zrefclever_label_type_a_tl }
3412     { \l__zrefclever_ref_language_tl }
```

```

3413         \l_tmpa_tl
3414         \l_tmpa_tl
3415         \l__zrefclever_zcref_note_tl
3416     }

```

Integration with zref-check.

```

3417     \bool_lazy_and:nnT
3418     { \l__zrefclever_zrefcheck_available_bool }
3419     { \l__zrefclever_zcref_with_check_bool }
3420     {
3421         \zrefcheck_zcref_end_label_maybe:
3422         \zrefcheck_zcref_run_checks_on_labels:n
3423         { \l__zrefclever_zcref_labels_seq }
3424     }

```

Integration with mathtools.

```

3425     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3426     {
3427         \__zrefclever_mathtools_showonlyrefs:n
3428         { \l__zrefclever_zcref_labels_seq }
3429     }
3430     \group_end:
3431 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3432 \seq_new:N \l__zrefclever_zcref_labels_seq
3433 \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>}

```

```

3434 \NewDocumentCommand \zcpageref { s O { } m }
3435 {
3436     \group_begin:
3437     \IfBooleanT {#1}
3438     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3439     \zcref [#2, ref = page] {#3}
3440     \group_end:
3441 }

```

(End definition for \zcpageref.)

7 Sorting

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

`\l_zrefclever_label_type_a_tl` Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.

`\l_zrefclever_label_type_b_tl`

`\l_zrefclever_label_enclval_a_tl` 3442 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 3443 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 3444 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 3445 `\tl_new:N \l__zrefclever_label_enclval_b_tl`

3446 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`

3447 `\tl_new:N \l__zrefclever_label_extdoc_b_tl`

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

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

3448 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End definition for `\l__zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

3449 `\int_new:N \l__zrefclever_sort_prior_a_int`

3450 `\int_new:N \l__zrefclever_sort_prior_b_int`

(End definition for `\l__zrefclever_sort_prior_a_int` and `\l__zrefclever_sort_prior_b_int`.)

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

3451 `\seq_new:N \l__zrefclever_label_types_seq`

(End definition for `\l__zrefclever_label_types_seq`.)

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

3452 `\cs_new_protected:Npn __zrefclever_sort_labels:`
 3453 `{`

Store label types sequence.

```

3454 \seq_clear:N \l__zrefclever_label_types_seq
3455 \tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }
3456 {
3457   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3458   \__zrefclever_label_type_put_new_right:n
3459 }

```

Sort.

```

3460 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3461 {
3462   \zref@ifrefundefined {##1}
3463   {
3464     \zref@ifrefundefined {##2}
3465     {
3466       % Neither label is defined.
3467       \sort_return_same:
3468     }
3469     {
3470       % The second label is defined, but the first isn't, leave the
3471       % undefined first (to be more visible).
3472       \sort_return_same:
3473     }
3474   }
3475   {
3476     \zref@ifrefundefined {##2}
3477     {
3478       % The first label is defined, but the second isn't, bring the
3479       % second forward.
3480       \sort_return_swapped:
3481     }
3482     {
3483       % The interesting case: both labels are defined. References
3484       % to the "default" property or to the "page" are quite
3485       % different with regard to sorting, so we branch them here to
3486       % specialized functions.
3487       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3488       { \__zrefclever_sort_page:nn {##1} {##2} }
3489       { \__zrefclever_sort_default:nn {##1} {##2} }
3490     }
3491   }
3492 }
3493 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_label_type_put_new_right:n

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

```

    \_zrefclever_label_type_put_new_right:n {\label}}
3494 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
3495 {
3496   \_zrefclever_extract_default:Nnnn
3497   \l__zrefclever_label_type_a_tl {#1} {zc@type} { }
3498   \seq_if_in:NVF \l__zrefclever_label_types_seq
3499   \l__zrefclever_label_type_a_tl
3500   {
3501     \seq_put_right:NV \l__zrefclever_label_types_seq
3502     \l__zrefclever_label_type_a_tl
3503   }
3504 }

```

(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}
3505 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
3506 {
3507   \_zrefclever_extract_default:Nnnn
3508   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3509   \_zrefclever_extract_default:Nnnn
3510   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3511
3512   \tl_if_eq:NNTF
3513   \l__zrefclever_label_type_a_tl
3514   \l__zrefclever_label_type_b_tl
3515   { \_zrefclever_sort_default_same_type:nn {#1} {#2} }
3516   { \_zrefclever_sort_default_different_types:nn {#1} {#2} }
3517 }

```

(End definition for _zrefclever_sort_default:nn.)

```

\_zrefclever_sort_default_same_type:nn \_zrefclever_sort_default_same_type:nn {\label a} {\label b}
3518 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
3519 {
3520   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3521   {#1} {zc@enclval} { }
3522   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3523   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3524   {#2} {zc@enclval} { }
3525   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3526   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3527   {#1} {externaldocument} { }
3528   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3529   {#2} {externaldocument} { }
3530
3531   \bool_set_false:N \l__zrefclever_sort_decided_bool

```

```

3532
3533 % First we check if there's any "external document" difference (coming
3534 % from 'zref-xr') and, if so, sort based on that.
3535 \tl_if_eq:NNTF
3536   \l__zrefclever_label_extdoc_a_tl
3537   \l__zrefclever_label_extdoc_b_tl
3538   {
3539     \bool_if:nTF
3540     {
3541       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3542       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3543     }
3544     {
3545       \bool_set_true:N \l__zrefclever_sort_decided_bool
3546       \sort_return_same:
3547     }
3548     {
3549       \bool_if:nTF
3550       {
3551         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3552         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3553       }
3554       {
3555         \bool_set_true:N \l__zrefclever_sort_decided_bool
3556         \sort_return_swapped:
3557       }
3558       {
3559         \bool_set_true:N \l__zrefclever_sort_decided_bool
3560         % Two different "external documents": last resort, sort by the
3561         % document name itself.
3562         \str_compare:eNeTF
3563         { \l__zrefclever_label_extdoc_b_tl } <
3564         { \l__zrefclever_label_extdoc_a_tl }
3565         { \sort_return_swapped: }
3566         { \sort_return_same: }
3567       }
3568     }
3569   }
3570
3571 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3572 {
3573   \bool_if:nTF
3574   {
3575     % Both are empty: neither label has any (further) "enclosing
3576     % counters" (left).
3577     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3578     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3579   }
3580   {
3581     \bool_set_true:N \l__zrefclever_sort_decided_bool
3582     \int_compare:nNnTF
3583     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3584     >
3585     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }

```



```

3586     { \sort_return_swapped: }
3587     { \sort_return_same:   }
3588 }
3589 {
3590     \bool_if:nTF
3591     {
3592         % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
3593         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3594     }
3595     {
3596         \bool_set_true:N \l__zrefclever_sort_decided_bool
3597         \int_compare:nNnTF
3598         { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3599         >
3600         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3601         { \sort_return_swapped: }
3602         { \sort_return_same:   }
3603     }
3604 }
3605     \bool_if:nTF
3606     {
3607         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3608         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3609     }
3610     {
3611         \bool_set_true:N \l__zrefclever_sort_decided_bool
3612         \int_compare:nNnTF
3613         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3614         <
3615         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3616         { \sort_return_same:   }
3617         { \sort_return_swapped: }
3618     }
3619 }
3620 % Neither is empty: we can compare the values of the
3621 % current enclosing counter in the loop, if they are
3622 % equal, we are still in the loop, if they are not, a
3623 % sorting decision can be made directly.
3624 \int_compare:nNnTF
3625 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3626 =
3627 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3628 {
3629     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3630     { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3631     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3632     { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3633 }
3634 {
3635     \bool_set_true:N \l__zrefclever_sort_decided_bool
3636     \int_compare:nNnTF
3637     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3638     >
3639     { \tl_head:N \l__zrefclever_label_enclval_b_tl }

```

```

3640         { \sort_return_swapped: }
3641         { \sort_return_same: }
3642     }
3643 }
3644 }
3645 }
3646 }
3647 }

```

(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>}

```

```

3648 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
3649 {

```

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

```

3650     \int_zero:N \l__zrefclever_sort_prior_a_int
3651     \int_zero:N \l__zrefclever_sort_prior_b_int
3652     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3653     {
3654         \tl_if_eq:nnTF {##2} {{othertypes}}
3655         {
3656             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3657             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3658             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3659             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3660         }
3661         {
3662             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3663             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3664             {
3665                 \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
3666                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3667             }
3668         }
3669     }

```

Then do the actual sorting.

```

3670     \bool_if:nTF
3671     {
3672         \int_compare_p:nNn
3673         { \l__zrefclever_sort_prior_a_int } <
3674         { \l__zrefclever_sort_prior_b_int }
3675     }
3676     { \sort_return_same: }
3677     {
3678         \bool_if:nTF
3679         {
3680             \int_compare_p:nNn
3681             { \l__zrefclever_sort_prior_a_int } >
3682             { \l__zrefclever_sort_prior_b_int }
3683         }

```

```

3684         { \sort_return_swapped: }
3685     {
3686         % Sort priorities are equal: the type that occurs first in
3687         % ‘labels’, as given by the user, is kept (or brought) forward.
3688         \seq_map_inline:Nn \l__zrefclever_label_types_seq
3689         {
3690             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3691             { \seq_map_break:n { \sort_return_same: } }
3692             {
3693                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3694                 { \seq_map_break:n { \sort_return_swapped: } }
3695             }
3696         }
3697     }
3698 }
3699 }

```

(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}}

3700 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3701 {
3702     \int_compare:nNnTF
3703     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3704     >
3705     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3706     { \sort_return_swapped: }
3707     { \sort_return_same: }
3708 }

```

(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, suggested by Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes (and good ones at that) see <https://tex.stackexchange.com/q/611370>. 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

`\l_zrefclever_typeset_labels_seq`
`\l_zrefclever_typeset_last_bool`
`\l_zrefclever_last_of_type_bool`

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

3709 `\seq_new:N \l__zrefclever_typeset_labels_seq`
3710 `\bool_new:N \l__zrefclever_typeset_last_bool`
3711 `\bool_new:N \l__zrefclever_last_of_type_bool`

(End definition for `\l_zrefclever_typeset_labels_seq`, `\l_zrefclever_typeset_last_bool`, and `\l__zrefclever_last_of_type_bool`.)

`\l_zrefclever_type_count_int`
`\l_zrefclever_label_count_int`
`\l__zrefclever_ref_count_int`

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

3712 `\int_new:N \l__zrefclever_type_count_int`
3713 `\int_new:N \l__zrefclever_label_count_int`
3714 `\int_new:N \l__zrefclever_ref_count_int`

(End definition for `\l_zrefclever_type_count_int`, `\l_zrefclever_label_count_int`, and `\l__zrefclever_ref_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.

3715 `\tl_new:N \l__zrefclever_label_a_tl`
3716 `\tl_new:N \l__zrefclever_label_b_tl`
3717 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
3718 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
3719 `\tl_new:N \l__zrefclever_type_first_label_tl`
3720 `\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_type_name_missing_bool`
`\l_zrefclever_name_format_tl`
`\l_zrefclever_name_format_fallback_tl`
`\l_zrefclever_type_name_gender_seq`

Auxiliary variables for `__zrefclever_typeset_refs`: type name handling.

3721 `\tl_new:N \l__zrefclever_type_name_tl`
3722 `\bool_new:N \l_zrefclever_name_in_link_bool`
3723 `\bool_new:N \l_zrefclever_type_name_missing_bool`
3724 `\tl_new:N \l__zrefclever_name_format_tl`
3725 `\tl_new:N \l__zrefclever_name_format_fallback_tl`
3726 `\seq_new:N \l__zrefclever_type_name_gender_seq`

(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_range_beg_is_first_bool`
`\l_zrefclever_range_end_ref_tl`
`\l_zrefclever_next_maybe_range_bool`
`\l_zrefclever_next_is_same_bool`

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

3727 `\int_new:N \l__zrefclever_range_count_int`
3728 `\int_new:N \l__zrefclever_range_same_count_int`
3729 `\tl_new:N \l__zrefclever_range_beg_label_tl`
3730 `\bool_new:N \l__zrefclever_range_beg_is_first_bool`
3731 `\tl_new:N \l__zrefclever_range_end_ref_tl`
3732 `\bool_new:N \l__zrefclever_next_maybe_range_bool`
3733 `\bool_new:N \l__zrefclever_next_is_same_bool`

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

`\l__zrefclever_tpairsep_tl` Auxiliary variables for `__zrefclever_typeset_refs`: separators, and font and other options.

```

\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_rangeseq_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_tl
\l__zrefclever_endrangefunc_tl
\l__zrefclever_endrangeprop_tl
\l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool
\l__zrefclever_rangetopair_bool

```

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

`\l__zrefclever_refbounds_first_seq` Auxiliary variables for `__zrefclever_typeset_refs::` advanced reference format options.

```

\l__zrefclever_refbounds_first_sg_seq
\l__zrefclever_refbounds_first_pb_seq
\l__zrefclever_refbounds_first_rb_seq
\l__zrefclever_refbounds_mid_seq
\l__zrefclever_refbounds_mid_rb_seq
\l__zrefclever_refbounds_mid_re_seq
\l__zrefclever_refbounds_last_seq
\l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_seq
\l__zrefclever_type_first_refbounds_set_bool

```

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

`\l__zrefclever_verbose_testing_bool` Internal variable which enables extra log messaging at points of interest in the code for purposes of regression testing. Particularly relevant to keep track of expansion control in `\l__zrefclever_typeset_queue_curr_tl`.

```

\bool_new:N \l__zrefclever_verbose_testing_bool

```

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

`__zrefclever_typeset_refs:` Main typesetting function for `\zceref`.

```

\cs_new_protected:Npn \__zrefclever_typeset_refs:
{
\seq_set_eq:NN \l__zrefclever_typeset_labels_seq
\l__zrefclever_zceref_labels_seq
\l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_typeset_queue_curr_tl
\l__zrefclever_type_first_label_tl

```

```

3769 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3770 \tl_clear:N \l__zrefclever_range_beg_label_tl
3771 \tl_clear:N \l__zrefclever_range_end_ref_tl
3772 \int_zero:N \l__zrefclever_label_count_int
3773 \int_zero:N \l__zrefclever_type_count_int
3774 \int_zero:N \l__zrefclever_ref_count_int
3775 \int_zero:N \l__zrefclever_range_count_int
3776 \int_zero:N \l__zrefclever_range_same_count_int
3777 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3778 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3779
3780 % Get type block options (not type-specific).
3781 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3782   { \l__zrefclever_label_type_a_tl }
3783   { \l__zrefclever_ref_language_tl }
3784   \l__zrefclever_tpairsep_tl
3785 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3786   { \l__zrefclever_label_type_a_tl }
3787   { \l__zrefclever_ref_language_tl }
3788   \l__zrefclever_tlistsep_tl
3789 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3790   { \l__zrefclever_label_type_a_tl }
3791   { \l__zrefclever_ref_language_tl }
3792   \l__zrefclever_tlastsep_tl
3793
3794 % Process label stack.
3795 \bool_set_false:N \l__zrefclever_typeset_last_bool
3796 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3797   {
3798     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3799     \l__zrefclever_label_a_tl
3800     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3801       {
3802         \tl_clear:N \l__zrefclever_label_b_tl
3803         \bool_set_true:N \l__zrefclever_typeset_last_bool
3804       }
3805       {
3806         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3807         \l__zrefclever_label_b_tl
3808       }
3809
3810     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3811       {
3812         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3813         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3814       }
3815       {
3816         \__zrefclever_extract_default:NVnn
3817         \l__zrefclever_label_type_a_tl
3818         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3819         \__zrefclever_extract_default:NVnn
3820         \l__zrefclever_label_type_b_tl
3821         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3822       }

```

```

3823
3824 % First, we establish whether the "current label" (i.e. 'a') is the
3825 % last one of its type. This can happen because the "next label"
3826 % (i.e. 'b') is of a different type (or different definition status),
3827 % or because we are at the end of the list.
3828 \bool_if:NTF \l__zrefclever_typeset_last_bool
3829 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3830 {
3831   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3832   {
3833     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3834     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3835     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3836   }
3837   {
3838     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3839     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3840     {
3841       % Neither is undefined, we must check the types.
3842       \tl_if_eq:NNTF
3843         \l__zrefclever_label_type_a_tl
3844         \l__zrefclever_label_type_b_tl
3845         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3846         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3847     }
3848   }
3849 }
3850
3851 % Handle warnings in case of reference or type undefined.
3852 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3853 \zref@refused { \l__zrefclever_label_a_tl }
3854 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3855 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3856 {}
3857 {
3858   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3859   {
3860     \msg_warning:nxx { zref-clever } { missing-type }
3861     { \l__zrefclever_label_a_tl }
3862   }
3863   \zref@ifrefcontainsprop
3864     { \l__zrefclever_label_a_tl }
3865     { \l__zrefclever_ref_property_tl }
3866     { }
3867     {
3868       \msg_warning:nxxx { zref-clever } { missing-property }
3869       { \l__zrefclever_ref_property_tl }
3870       { \l__zrefclever_label_a_tl }
3871     }
3872 }
3873
3874 % Get possibly type-specific separators, refbounds, font and other
3875 % options, once per type.
3876 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```



```

3877 {
3878   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3879   { \l__zrefclever_label_type_a_tl }
3880   { \l__zrefclever_ref_language_tl }
3881   \l__zrefclever_namesep_tl
3882   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3883   { \l__zrefclever_label_type_a_tl }
3884   { \l__zrefclever_ref_language_tl }
3885   \l__zrefclever_pairsep_tl
3886   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3887   { \l__zrefclever_label_type_a_tl }
3888   { \l__zrefclever_ref_language_tl }
3889   \l__zrefclever_listsep_tl
3890   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3891   { \l__zrefclever_label_type_a_tl }
3892   { \l__zrefclever_ref_language_tl }
3893   \l__zrefclever_lastsep_tl
3894   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3895   { \l__zrefclever_label_type_a_tl }
3896   { \l__zrefclever_ref_language_tl }
3897   \l__zrefclever_rangesep_tl
3898   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3899   { \l__zrefclever_label_type_a_tl }
3900   { \l__zrefclever_ref_language_tl }
3901   \l__zrefclever_namefont_tl
3902   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3903   { \l__zrefclever_label_type_a_tl }
3904   { \l__zrefclever_ref_language_tl }
3905   \l__zrefclever_reffont_tl
3906   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3907   { \l__zrefclever_label_type_a_tl }
3908   { \l__zrefclever_ref_language_tl }
3909   \l__zrefclever_endrangefunc_tl
3910   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3911   { \l__zrefclever_label_type_a_tl }
3912   { \l__zrefclever_ref_language_tl }
3913   \l__zrefclever_endrangeprop_tl
3914   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3915   { \l__zrefclever_label_type_a_tl }
3916   { \l__zrefclever_ref_language_tl }
3917   \l__zrefclever_cap_bool
3918   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3919   { \l__zrefclever_label_type_a_tl }
3920   { \l__zrefclever_ref_language_tl }
3921   \l__zrefclever_abbrev_bool
3922   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3923   { \l__zrefclever_label_type_a_tl }
3924   { \l__zrefclever_ref_language_tl }
3925   \l__zrefclever_rangetopair_bool
3926   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3927   { \l__zrefclever_label_type_a_tl }
3928   { \l__zrefclever_ref_language_tl }
3929   \l__zrefclever_refbounds_first_seq
3930   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3931         { \l__zrefclever_label_type_a_tl }
3932         { \l__zrefclever_ref_language_tl }
3933         \l__zrefclever_refbounds_first_sg_seq
3934     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3935         { \l__zrefclever_label_type_a_tl }
3936         { \l__zrefclever_ref_language_tl }
3937         \l__zrefclever_refbounds_first_pb_seq
3938     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3939         { \l__zrefclever_label_type_a_tl }
3940         { \l__zrefclever_ref_language_tl }
3941         \l__zrefclever_refbounds_first_rb_seq
3942     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3943         { \l__zrefclever_label_type_a_tl }
3944         { \l__zrefclever_ref_language_tl }
3945         \l__zrefclever_refbounds_mid_seq
3946     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3947         { \l__zrefclever_label_type_a_tl }
3948         { \l__zrefclever_ref_language_tl }
3949         \l__zrefclever_refbounds_mid_rb_seq
3950     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3951         { \l__zrefclever_label_type_a_tl }
3952         { \l__zrefclever_ref_language_tl }
3953         \l__zrefclever_refbounds_mid_re_seq
3954     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3955         { \l__zrefclever_label_type_a_tl }
3956         { \l__zrefclever_ref_language_tl }
3957         \l__zrefclever_refbounds_last_seq
3958     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3959         { \l__zrefclever_label_type_a_tl }
3960         { \l__zrefclever_ref_language_tl }
3961         \l__zrefclever_refbounds_last_pe_seq
3962     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3963         { \l__zrefclever_label_type_a_tl }
3964         { \l__zrefclever_ref_language_tl }
3965         \l__zrefclever_refbounds_last_re_seq
3966 }
3967
3968 % Here we send this to a couple of auxiliary functions.
3969 \bool_if:NTF \l__zrefclever_last_of_type_bool
3970 % There exists no next label of the same type as the current.
3971 { \__zrefclever_typeset_refs_last_of_type: }
3972 % There exists a next label of the same type as the current.
3973 { \__zrefclever_typeset_refs_not_last_of_type: }
3974 }
3975 }

```

(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.

```

3976 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3977 {
3978   % Process the current label to the current queue.
3979   \int_case:nnF { \l__zrefclever_label_count_int }
3980   {
3981     % It is the last label of its type, but also the first one, and that's
3982     % what matters here: just store it.
3983     % Test: 'zc-typeset01.lvt': "Last of type: single"
3984     { 0 }
3985     {
3986       \tl_set:NV \l__zrefclever_type_first_label_tl
3987       \l__zrefclever_label_a_tl
3988       \tl_set:NV \l__zrefclever_type_first_label_type_tl
3989       \l__zrefclever_label_type_a_tl
3990       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3991       \l__zrefclever_refbounds_first_sg_seq
3992       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3993     }
3994
3995     % The last is the second: we have a pair (if not repeated).
3996     % Test: 'zc-typeset01.lvt': "Last of type: pair"
3997     { 1 }
3998     {
3999       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4000       {
4001         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4002         \l__zrefclever_refbounds_first_sg_seq
4003         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4004       }
4005       {
4006         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4007         {
4008           \exp_not:V \l__zrefclever_pairsep_tl
4009           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4010           \l__zrefclever_refbounds_last_pe_seq
4011         }
4012         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4013         \l__zrefclever_refbounds_first_pb_seq
4014         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4015       }
4016     }
4017   }
4018   % Last is third or more of its type: without repetition, we'd have the
4019   % last element on a list, but control for possible repetition.
4020   {
4021     \int_case:nnF { \l__zrefclever_range_count_int }
4022     {
4023       % There was no range going on.
4024       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4025       { 0 }

```

```

4026 {
4027   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4028   {
4029     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4030     {
4031       \exp_not:V \l__zrefclever_pairsep_tl
4032       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4033       \l__zrefclever_refbounds_last_pe_seq
4034     }
4035   }
4036   {
4037     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4038     {
4039       \exp_not:V \l__zrefclever_lastsep_tl
4040       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4041       \l__zrefclever_refbounds_last_seq
4042     }
4043   }
4044 }
4045 % Last in the range is also the second in it.
4046 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4047 { 1 }
4048 {
4049   \int_compare:nNnTF
4050   { \l__zrefclever_range_same_count_int } = { 1 }
4051   {
4052     % We know 'range_beg_is_first_bool' is false, since this is
4053     % the second element in the range, but the third or more in
4054     % the type list.
4055     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4056     {
4057       \exp_not:V \l__zrefclever_pairsep_tl
4058       \__zrefclever_get_ref:VN
4059       \l__zrefclever_range_beg_label_tl
4060       \l__zrefclever_refbounds_last_pe_seq
4061     }
4062     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4063     \l__zrefclever_refbounds_first_pb_seq
4064     \bool_set_true:N
4065     \l__zrefclever_type_first_refbounds_set_bool
4066   }
4067   {
4068     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4069     {
4070       \exp_not:V \l__zrefclever_listsep_tl
4071       \__zrefclever_get_ref:VN
4072       \l__zrefclever_range_beg_label_tl
4073       \l__zrefclever_refbounds_mid_seq
4074       \exp_not:V \l__zrefclever_lastsep_tl
4075       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4076       \l__zrefclever_refbounds_last_seq
4077     }
4078   }
4079 }

```

```

4080 }
4081 % Last in the range is third or more in it.
4082 {
4083   \int_case:nnF
4084   {
4085     \l__zrefclever_range_count_int -
4086     \l__zrefclever_range_same_count_int
4087   }
4088   {
4089     % Repetition, not a range.
4090     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4091     { 0 }
4092     {
4093       % If 'range_beg_is_first_bool' is true, it means it was also
4094       % the first of the type, and hence its typesetting was
4095       % already handled, and we just have to set rebounds.
4096       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4097       {
4098         \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4099         \l__zrefclever_rebounds_first_sg_seq
4100         \bool_set_true:N
4101         \l__zrefclever_type_first_rebounds_set_bool
4102       }
4103       {
4104         \int_compare:nNnTF
4105         { \l__zrefclever_ref_count_int } < { 2 }
4106         {
4107           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4108           {
4109             \exp_not:V \l__zrefclever_pairsep_tl
4110             \__zrefclever_get_ref:VN
4111             \l__zrefclever_range_beg_label_tl
4112             \l__zrefclever_rebounds_last_pe_seq
4113           }
4114         }
4115         {
4116           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4117           {
4118             \exp_not:V \l__zrefclever_lastsep_tl
4119             \__zrefclever_get_ref:VN
4120             \l__zrefclever_range_beg_label_tl
4121             \l__zrefclever_rebounds_last_seq
4122           }
4123         }
4124       }
4125     }
4126     % A 'range', but with no skipped value, treat as pair if range
4127     % started with first of type, otherwise as list.
4128     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4129     { 1 }
4130     {
4131       % Ditto.
4132       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4133       {

```

```

4134         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4135         \l__zrefclever_refbounds_first_pb_seq
4136     \bool_set_true:N
4137         \l__zrefclever_type_first_refbounds_set_bool
4138     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4139     {
4140         \exp_not:V \l__zrefclever_pairsep_tl
4141         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4142         \l__zrefclever_refbounds_last_pe_seq
4143     }
4144 }
4145 {
4146     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4147     {
4148         \exp_not:V \l__zrefclever_listsep_tl
4149         \__zrefclever_get_ref:VN
4150         \l__zrefclever_range_beg_label_tl
4151         \l__zrefclever_refbounds_mid_seq
4152     }
4153     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4154     {
4155         \exp_not:V \l__zrefclever_lastsep_tl
4156         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4157         \l__zrefclever_refbounds_last_seq
4158     }
4159 }
4160 }
4161 }
4162 {
4163     % An actual range.
4164     % Test: 'zc-typeset01.lvt': "Last of type: range"
4165     % Ditto.
4166     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4167     {
4168         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4169         \l__zrefclever_refbounds_first_rb_seq
4170         \bool_set_true:N
4171         \l__zrefclever_type_first_refbounds_set_bool
4172     }
4173     {
4174         \int_compare:nNnTF
4175         { \l__zrefclever_ref_count_int } < { 2 }
4176         {
4177             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4178             {
4179                 \exp_not:V \l__zrefclever_pairsep_tl
4180                 \__zrefclever_get_ref:VN
4181                 \l__zrefclever_range_beg_label_tl
4182                 \l__zrefclever_refbounds_mid_rb_seq
4183             }
4184             \seq_set_eq:NN
4185             \l__zrefclever_type_first_refbounds_seq
4186             \l__zrefclever_refbounds_first_pb_seq
4187             \bool_set_true:N

```

```

4188         \l__zrefclever_type_first_refbounds_set_bool
4189     }
4190     {
4191         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4192     {
4193         \exp_not:V \l__zrefclever_lastsep_tl
4194         \__zrefclever_get_ref:VN
4195         \l__zrefclever_range_beg_label_tl
4196         \l__zrefclever_refbounds_mid_rb_seq
4197     }
4198     }
4199 }
4200 \bool_lazy_and:nnTF
4201 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4202 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4203 {
4204     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4205     \l__zrefclever_range_beg_label_tl
4206     \l__zrefclever_label_a_tl
4207     \l__zrefclever_range_end_ref_tl
4208     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4209     {
4210         \exp_not:V \l__zrefclever_rangesep_tl
4211         \__zrefclever_get_ref_endrange:VVN
4212         \l__zrefclever_label_a_tl
4213         \l__zrefclever_range_end_ref_tl
4214         \l__zrefclever_refbounds_last_re_seq
4215     }
4216 }
4217 {
4218     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4219     {
4220         \exp_not:V \l__zrefclever_rangesep_tl
4221         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4222         \l__zrefclever_refbounds_last_re_seq
4223     }
4224 }
4225 }
4226 }
4227 }
4228
4229 % Handle "range" option. The idea is simple: if the queue is not empty,
4230 % we replace it with the end of the range (or pair). We can still
4231 % retrieve the end of the range from 'label_a' since we know to be
4232 % processing the last label of its type at this point.
4233 \bool_if:NT \l__zrefclever_typeset_range_bool
4234 {
4235     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4236     {
4237         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4238         { }
4239         {
4240             \msg_warning:nxx { zref-clever } { single-element-range }
4241             { \l__zrefclever_type_first_label_type_tl }

```

```

4242     }
4243   }
4244   {
4245     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4246     \bool_if:NT \l__zrefclever_rangetopair_bool
4247     {
4248       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4249       { }
4250       {
4251         \__zrefclever_labels_in_sequence:nn
4252         { \l__zrefclever_type_first_label_tl }
4253         { \l__zrefclever_label_a_tl }
4254       }
4255     }
4256     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4257     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4258     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4259     {
4260       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4261       {
4262         \exp_not:V \l__zrefclever_pairsep_tl
4263         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4264         \l__zrefclever_refbounds_last_pe_seq
4265       }
4266       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4267       \l__zrefclever_refbounds_first_pb_seq
4268       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4269     }
4270     {
4271       \bool_lazy_and:nnTF
4272       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4273       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4274       {
4275         % We must get 'type_first_label_tl' instead of
4276         % 'range_beg_label_tl' here, since it is not necessary
4277         % that the first of type was actually starting a range for
4278         % the 'range' option to be used.
4279         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4280         \l__zrefclever_type_first_label_tl
4281         \l__zrefclever_label_a_tl
4282         \l__zrefclever_range_end_ref_tl
4283         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4284         {
4285           \exp_not:V \l__zrefclever_rangesep_tl
4286           \__zrefclever_get_ref_endrange:VVN
4287           \l__zrefclever_label_a_tl
4288           \l__zrefclever_range_end_ref_tl
4289           \l__zrefclever_refbounds_last_re_seq
4290         }
4291       }
4292     }
4293     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4294     {
4295       \exp_not:V \l__zrefclever_rangesep_tl

```



```

4296         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4297         \l__zrefclever_refbounds_last_re_seq
4298     }
4299 }
4300 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4301 \l__zrefclever_refbounds_first_rb_seq
4302 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4303 }
4304 }
4305 }
4306
4307 % If none of the special cases for the first of type refbounds have been
4308 % set, do it.
4309 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4310 {
4311     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4312     \l__zrefclever_refbounds_first_seq
4313 }
4314
4315 % Now that the type block is finished, we can add the name and the first
4316 % ref to the queue. Also, if "typeset" option is not "both", handle it
4317 % here as well.
4318 \_zrefclever_type_name_setup:
4319 \bool_if:nTF
4320 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4321 {
4322     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4323     { \_zrefclever_get_ref_first: }
4324 }
4325 {
4326     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4327     {
4328         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4329         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4330         {
4331             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4332             \l__zrefclever_type_first_refbounds_seq
4333         }
4334     }
4335     {
4336         \bool_if:NTF \l__zrefclever_typeset_name_bool
4337         {
4338             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4339             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4340             {
4341                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4342                 {
4343                     \exp_not:N \group_begin:
4344                     \exp_not:V \l__zrefclever_namefont_tl
4345                     \_zrefclever_hyperlink:nnn
4346                     {
4347                         \_zrefclever_extract_url_unexp:V
4348                         \l__zrefclever_type_first_label_tl
4349                     }

```

```

4350         {
4351             \__zrefclever_extract_unexp:Vnn
4352             \l__zrefclever_type_first_label_tl
4353             { anchor } { }
4354         }
4355         { \exp_not:V \l__zrefclever_type_name_tl }
4356         \exp_not:N \group_end:
4357     }
4358     {
4359         \exp_not:N \group_begin:
4360         \exp_not:V \l__zrefclever_namefont_tl
4361         \exp_not:V \l__zrefclever_type_name_tl
4362         \exp_not:N \group_end:
4363     }
4364 }
4365 }
4366 {
4367     % Logically, this case would correspond to "typeset=none", but
4368     % it should not occur, given that the options are set up to
4369     % typeset either "ref" or "name". Still, leave here a
4370     % sensible fallback, equal to the behavior of "both".
4371     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4372     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4373     { \__zrefclever_get_ref_first: }
4374 }
4375 }
4376 }
4377
4378 % Typeset the previous type block, if there is one.
4379 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4380 {
4381     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4382     { \l__zrefclever_tlistsep_tl }
4383     \l__zrefclever_typeset_queue_prev_tl
4384 }
4385
4386 % Extra log for testing.
4387 \bool_if:NT \l__zrefclever_verbose_testing_bool
4388 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4389
4390 % Wrap up loop, or prepare for next iteration.
4391 \bool_if:NTF \l__zrefclever_typeset_last_bool
4392 {
4393     % We are finishing, typeset the current queue.
4394     \int_case:nnF { \l__zrefclever_type_count_int }
4395     {
4396         % Single type.
4397         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4398         { 0 }
4399         { \l__zrefclever_typeset_queue_curr_tl }
4400         % Pair of types.
4401         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4402         { 1 }
4403         {

```

```

4404         \l__zrefclever_tpairsep_tl
4405         \l__zrefclever_typeset_queue_curr_tl
4406     }
4407 }
4408 {
4409     % Last in list of types.
4410     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4411     \l__zrefclever_tlastsep_tl
4412     \l__zrefclever_typeset_queue_curr_tl
4413 }
4414 % And nudge in case of multitype reference.
4415 \bool_lazy_all:nT
4416 {
4417     { \l__zrefclever_nudge_enabled_bool }
4418     { \l__zrefclever_nudge_multitype_bool }
4419     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4420 }
4421 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4422 }
4423 {
4424     % There are further labels, set variables for next iteration.
4425     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4426         \l__zrefclever_typeset_queue_curr_tl
4427     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4428     \tl_clear:N \l__zrefclever_type_first_label_tl
4429     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4430     \tl_clear:N \l__zrefclever_range_beg_label_tl
4431     \tl_clear:N \l__zrefclever_range_end_ref_tl
4432     \int_zero:N \l__zrefclever_label_count_int
4433     \int_zero:N \l__zrefclever_ref_count_int
4434     \int_incr:N \l__zrefclever_type_count_int
4435     \int_zero:N \l__zrefclever_range_count_int
4436     \int_zero:N \l__zrefclever_range_same_count_int
4437     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4438     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4439 }
4440 }

```

(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.

```

4441 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
4442 {
4443     % Signal if next label may form a range with the current one (only
4444     % considered if compression is enabled in the first place).
4445     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4446     \bool_set_false:N \l__zrefclever_next_is_same_bool
4447     \bool_if:NT \l__zrefclever_typeset_compress_bool
4448     {
4449         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4450         { }
4451         {
4452             \_zrefclever_labels_in_sequence:nn
4453             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4454     }
4455 }
4456
4457 % Process the current label to the current queue.
4458 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4459 {
4460     % Current label is the first of its type (also not the last, but it
4461     % doesn't matter here): just store the label.
4462     \tl_set:NV \l__zrefclever_type_first_label_tl
4463         \l__zrefclever_label_a_tl
4464     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4465         \l__zrefclever_label_type_a_tl
4466     \int_incr:N \l__zrefclever_ref_count_int
4467
4468     % If the next label may be part of a range, signal it (we deal with it
4469     % as the "first", and must do it there, to handle hyperlinking), but
4470     % also step the range counters.
4471     % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4472     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4473     {
4474         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4475         \tl_set:NV \l__zrefclever_range_beg_label_tl
4476             \l__zrefclever_label_a_tl
4477         \tl_clear:N \l__zrefclever_range_end_ref_tl
4478         \int_incr:N \l__zrefclever_range_count_int
4479         \bool_if:NT \l__zrefclever_next_is_same_bool
4480             { \int_incr:N \l__zrefclever_range_same_count_int }
4481     }
4482 }
4483 {
4484     % Current label is neither the first (nor the last) of its type.
4485     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4486     {
4487         % Starting, or continuing a range.
4488         \int_compare:nNnTF
4489             { \l__zrefclever_range_count_int } = { 0 }
4490         {
4491             % There was no range going, we are starting one.
4492             \tl_set:NV \l__zrefclever_range_beg_label_tl
4493                 \l__zrefclever_label_a_tl
4494             \tl_clear:N \l__zrefclever_range_end_ref_tl
4495             \int_incr:N \l__zrefclever_range_count_int
4496             \bool_if:NT \l__zrefclever_next_is_same_bool
4497                 { \int_incr:N \l__zrefclever_range_same_count_int }
4498         }
4499         {
4500             % Second or more in the range, but not the last.
4501             \int_incr:N \l__zrefclever_range_count_int
4502             \bool_if:NT \l__zrefclever_next_is_same_bool
4503                 { \int_incr:N \l__zrefclever_range_same_count_int }
4504         }
4505     }
4506 }
4507 % Next element is not in sequence: there was no range, or we are

```

```

4508 % closing one.
4509 \int_case:nnF { \l__zrefclever_range_count_int }
4510 {
4511 % There was no range going on.
4512 % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4513 { 0 }
4514 {
4515 \int_incr:N \l__zrefclever_ref_count_int
4516 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4517 {
4518 \exp_not:V \l__zrefclever_listsep_tl
4519 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4520 \l__zrefclever_refbounds_mid_seq
4521 }
4522 }
4523 % Last is second in the range: if 'range_same_count' is also
4524 % '1', it's a repetition (drop it), otherwise, it's a "pair
4525 % within a list", treat as list.
4526 % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4527 % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4528 { 1 }
4529 {
4530 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4531 {
4532 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4533 \l__zrefclever_refbounds_first_seq
4534 \bool_set_true:N
4535 \l__zrefclever_type_first_refbounds_set_bool
4536 }
4537 {
4538 \int_incr:N \l__zrefclever_ref_count_int
4539 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4540 {
4541 \exp_not:V \l__zrefclever_listsep_tl
4542 \__zrefclever_get_ref:VN
4543 \l__zrefclever_range_beg_label_tl
4544 \l__zrefclever_refbounds_mid_seq
4545 }
4546 }
4547 \int_compare:nNnF
4548 { \l__zrefclever_range_same_count_int } = { 1 }
4549 {
4550 \int_incr:N \l__zrefclever_ref_count_int
4551 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4552 {
4553 \exp_not:V \l__zrefclever_listsep_tl
4554 \__zrefclever_get_ref:VN
4555 \l__zrefclever_label_a_tl
4556 \l__zrefclever_refbounds_mid_seq
4557 }
4558 }
4559 }
4560 }
4561 {

```

```

4562 % Last is third or more in the range: if 'range_count' and
4563 % 'range_same_count' are the same, its a repetition (drop it),
4564 % if they differ by '1', its a list, if they differ by more,
4565 % it is a real range.
4566 \int_case:nnF
4567 {
4568   \l__zrefclever_range_count_int -
4569   \l__zrefclever_range_same_count_int
4570 }
4571 {
4572   % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4573   { 0 }
4574   {
4575     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4576     {
4577       \seq_set_eq:NN
4578       \l__zrefclever_type_first_refbounds_seq
4579       \l__zrefclever_refbounds_first_seq
4580       \bool_set_true:N
4581       \l__zrefclever_type_first_refbounds_set_bool
4582     }
4583     {
4584       \int_incr:N \l__zrefclever_ref_count_int
4585       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4586       {
4587         \exp_not:V \l__zrefclever_listsep_tl
4588         \__zrefclever_get_ref:VN
4589         \l__zrefclever_range_beg_label_tl
4590         \l__zrefclever_refbounds_mid_seq
4591       }
4592     }
4593   }
4594   % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4595   { 1 }
4596   {
4597     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4598     {
4599       \seq_set_eq:NN
4600       \l__zrefclever_type_first_refbounds_seq
4601       \l__zrefclever_refbounds_first_seq
4602       \bool_set_true:N
4603       \l__zrefclever_type_first_refbounds_set_bool
4604     }
4605     {
4606       \int_incr:N \l__zrefclever_ref_count_int
4607       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4608       {
4609         \exp_not:V \l__zrefclever_listsep_tl
4610         \__zrefclever_get_ref:VN
4611         \l__zrefclever_range_beg_label_tl
4612         \l__zrefclever_refbounds_mid_seq
4613       }
4614     }
4615     \int_incr:N \l__zrefclever_ref_count_int

```

```

4616 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4617 {
4618   \exp_not:V \l__zrefclever_listsep_tl
4619   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4620   \l__zrefclever_refbounds_mid_seq
4621 }
4622 }
4623 }
4624 {
4625   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4626   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4627   {
4628     \seq_set_eq:NN
4629     \l__zrefclever_type_first_refbounds_seq
4630     \l__zrefclever_refbounds_first_rb_seq
4631     \bool_set_true:N
4632     \l__zrefclever_type_first_refbounds_set_bool
4633   }
4634   {
4635     \int_incr:N \l__zrefclever_ref_count_int
4636     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4637     {
4638       \exp_not:V \l__zrefclever_listsep_tl
4639       \__zrefclever_get_ref:VN
4640       \l__zrefclever_range_beg_label_tl
4641       \l__zrefclever_refbounds_mid_rb_seq
4642     }
4643   }
4644   % For the purposes of the serial comma, and thus for the
4645   % distinction of 'lastsep' and 'pairsep', a "range" counts
4646   % as one. Since 'range_beg' has already been counted
4647   % (here or with the first of type), we refrain from
4648   % incrementing 'ref_count_int'.
4649   \bool_lazy_and:nnTF
4650   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4651   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4652   {
4653     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4654     \l__zrefclever_range_beg_label_tl
4655     \l__zrefclever_label_a_tl
4656     \l__zrefclever_range_end_ref_tl
4657     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4658     {
4659       \exp_not:V \l__zrefclever_rangesep_tl
4660       \__zrefclever_get_ref_endrange:VVN
4661       \l__zrefclever_label_a_tl
4662       \l__zrefclever_range_end_ref_tl
4663       \l__zrefclever_refbounds_mid_re_seq
4664     }
4665   }
4666   {
4667     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4668     {
4669       \exp_not:V \l__zrefclever_rangesep_tl

```

```

4670             \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4671             \l__zrefclever_refbounds_mid_re_seq
4672         }
4673     }
4674 }
4675 }
4676 % We just closed a range, reset 'range_beg_is_first' in case a
4677 % second range for the same type occurs, in which case its
4678 % 'range_beg' will no longer be 'first'.
4679 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4680 % Reset counters.
4681 \int_zero:N \l__zrefclever_range_count_int
4682 \int_zero:N \l__zrefclever_range_same_count_int
4683 }
4684 }
4685 % Step label counter for next iteration.
4686 \int_incr:N \l__zrefclever_label_count_int
4687 }

```

(End definition for `_zrefclever_typeset_refs_not_last_of_type:.`)

Auxiliary functions

`_zrefclever_get_ref:nN` and `_zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `_zrefclever_get_ref:nN` 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:nN` and `_zrefclever_get_ref_first:` get called, as they must, in the context of `x` type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

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

```

4688 \cs_new_protected:Npn \_zrefclever_ref_default:
4689 { \zref@default }
4690 \cs_new_protected:Npn \_zrefclever_name_default:
4691 { \zref@default }

```


(End definition for _zrefclever_ref_default: and _zrefclever_name_default:.)

_zrefclever_get_ref:nN Handles a complete reference block to be accumulated in the “queue”, including ref-bounds, and hyperlinking. For use with all labels, except the first of its type, which is done by _zrefclever_get_ref_first:, and the last of a range, which is done by _zrefclever_get_ref_endrange:nnN.

```

\_zrefclever_get_ref:nN {\label}} {\refbounds}}

4692 \cs_new:Npn \_zrefclever_get_ref:nN #1#2
4693 {
4694   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4695   {
4696     \bool_if:nTF
4697     {
4698       \l__zrefclever_hyperlink_bool &&
4699       ! \l__zrefclever_link_star_bool
4700     }
4701     {
4702       \seq_item:Nn #2 { 1 }
4703       \_zrefclever_hyperlink:nnn
4704       { \_zrefclever_extract_url_unexp:n {#1} }
4705       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4706       {
4707         \seq_item:Nn #2 { 2 }
4708         \exp_not:N \group_begin:
4709         \exp_not:V \l__zrefclever_reffont_tl
4710         \_zrefclever_extract_unexp:nvn {#1}
4711         { \l__zrefclever_ref_property_tl } { }
4712         \exp_not:N \group_end:
4713         \seq_item:Nn #2 { 3 }
4714       }
4715       \seq_item:Nn #2 { 4 }
4716     }
4717     {
4718       \seq_item:Nn #2 { 1 }
4719       \seq_item:Nn #2 { 2 }
4720       \exp_not:N \group_begin:
4721       \exp_not:V \l__zrefclever_reffont_tl
4722       \_zrefclever_extract_unexp:nvn {#1}
4723       { \l__zrefclever_ref_property_tl } { }
4724       \exp_not:N \group_end:
4725       \seq_item:Nn #2 { 3 }
4726       \seq_item:Nn #2 { 4 }
4727     }
4728   }
4729   { \_zrefclever_ref_default: }
4730 }
4731 \cs_generate_variant:Nn \_zrefclever_get_ref:nN { VN }

```

(End definition for _zrefclever_get_ref:nN.)

```

\_zrefclever_get_ref_endrange:nnN \_zrefclever_get_ref_endrange:nnN {\label}} {\reference}} {\refbounds}}
4732 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4733 {
4734   \str_if_eq:nnTF {#2} { zc@missingproperty }
4735   { \__zrefclever_ref_default: }
4736   {
4737     \bool_if:nTF
4738     {
4739       \l__zrefclever_hyperlink_bool &&
4740       ! \l__zrefclever_link_star_bool
4741     }
4742     {
4743       \seq_item:Nn #3 { 1 }
4744       \__zrefclever_hyperlink:nnn
4745       { \__zrefclever_extract_url_unexp:n {#1} }
4746       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4747       {
4748         \seq_item:Nn #3 { 2 }
4749         \exp_not:N \group_begin:
4750         \exp_not:V \l__zrefclever_reffont_tl
4751         \exp_not:n {#2}
4752         \exp_not:N \group_end:
4753         \seq_item:Nn #3 { 3 }
4754       }
4755       \seq_item:Nn #3 { 4 }
4756     }
4757     {
4758       \seq_item:Nn #3 { 1 }
4759       \seq_item:Nn #3 { 2 }
4760       \exp_not:N \group_begin:
4761       \exp_not:V \l__zrefclever_reffont_tl
4762       \exp_not:n {#2}
4763       \exp_not:N \group_end:
4764       \seq_item:Nn #3 { 3 }
4765       \seq_item:Nn #3 { 4 }
4766     }
4767   }
4768 }
4769 \cs_generate_variant:Nn \__zrefclever_get_ref_endrange:nnN { VVN }

```

(End definition for __zrefclever_get_ref_endrange:nnN.)

`__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.

```

4770 \cs_new:Npn \__zrefclever_get_ref_first:
4771 {
4772   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4773   { \__zrefclever_ref_default: }
4774   {
4775     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4776 {
4777   \zref@ifrefcontainsprop
4778   { \l__zrefclever_type_first_label_tl }
4779   { \l__zrefclever_ref_property_tl }
4780   {
4781     \__zrefclever_hyperlink:nnn
4782     {
4783       \__zrefclever_extract_url_unexp:V
4784       \l__zrefclever_type_first_label_tl
4785     }
4786     {
4787       \__zrefclever_extract_unexp:Vnn
4788       \l__zrefclever_type_first_label_tl { anchor } { }
4789     }
4790     {
4791       \exp_not:N \group_begin:
4792       \exp_not:V \l__zrefclever_namefont_tl
4793       \exp_not:V \l__zrefclever_type_name_tl
4794       \exp_not:N \group_end:
4795       \exp_not:V \l__zrefclever_namesep_tl
4796       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4797       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4798       \exp_not:N \group_begin:
4799       \exp_not:V \l__zrefclever_reffont_tl
4800       \__zrefclever_extract_unexp:Vnn
4801       \l__zrefclever_type_first_label_tl
4802       { \l__zrefclever_ref_property_tl } { }
4803       \exp_not:N \group_end:
4804       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4805     }
4806     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4807   }
4808   {
4809     \exp_not:N \group_begin:
4810     \exp_not:V \l__zrefclever_namefont_tl
4811     \exp_not:V \l__zrefclever_type_name_tl
4812     \exp_not:N \group_end:
4813     \exp_not:V \l__zrefclever_namesep_tl
4814     \__zrefclever_ref_default:
4815   }
4816 }
4817 {
4818   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4819   {
4820     \__zrefclever_name_default:
4821     \exp_not:V \l__zrefclever_namesep_tl
4822   }
4823   {
4824     \exp_not:N \group_begin:
4825     \exp_not:V \l__zrefclever_namefont_tl
4826     \exp_not:V \l__zrefclever_type_name_tl
4827     \exp_not:N \group_end:
4828     \tl_if_empty:NF \l__zrefclever_type_name_tl
4829     { \exp_not:V \l__zrefclever_namesep_tl }

```

```

4830     }
4831 \zref@ifrefcontainsprop
4832 { \l__zrefclever_type_first_label_tl }
4833 { \l__zrefclever_ref_property_tl }
4834 {
4835     \bool_if:nTF
4836     {
4837         \l__zrefclever_hyperlink_bool &&
4838         ! \l__zrefclever_link_star_bool
4839     }
4840     {
4841         \seq_item:Nn
4842         \l__zrefclever_type_first_refbounds_seq { 1 }
4843         \__zrefclever_hyperlink:nnn
4844         {
4845             \__zrefclever_extract_url_unexp:V
4846             \l__zrefclever_type_first_label_tl
4847         }
4848         {
4849             \__zrefclever_extract_unexp:Vnn
4850             \l__zrefclever_type_first_label_tl { anchor } { }
4851         }
4852         {
4853             \seq_item:Nn
4854             \l__zrefclever_type_first_refbounds_seq { 2 }
4855             \exp_not:N \group_begin:
4856             \exp_not:V \l__zrefclever_reffont_tl
4857             \__zrefclever_extract_unexp:Vnn
4858             \l__zrefclever_type_first_label_tl
4859             { \l__zrefclever_ref_property_tl } { }
4860             \exp_not:N \group_end:
4861             \seq_item:Nn
4862             \l__zrefclever_type_first_refbounds_seq { 3 }
4863         }
4864         \seq_item:Nn
4865         \l__zrefclever_type_first_refbounds_seq { 4 }
4866     }
4867     {
4868         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4869         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4870         \exp_not:N \group_begin:
4871         \exp_not:V \l__zrefclever_reffont_tl
4872         \__zrefclever_extract_unexp:Vnn
4873         \l__zrefclever_type_first_label_tl
4874         { \l__zrefclever_ref_property_tl } { }
4875         \exp_not:N \group_end:
4876         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4877         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4878     }
4879 }
4880 { \__zrefclever_ref_default: }
4881 }
4882 }
4883 }

```

(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.

```

4884 \cs_new_protected:Npn \_zrefclever_type_name_setup:
4885 {
4886   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
4887   {
4888     \tl_clear:N \l\_zrefclever_type_name_tl
4889     \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4890   }
4891   {
4892     \tl_if_eq:NnTF
4893       \l\_zrefclever_type_first_label_type_tl { zc@missingtype }
4894       {
4895         \tl_clear:N \l\_zrefclever_type_name_tl
4896         \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4897       }
4898       {
4899         % Determine whether we should use capitalization, abbreviation,
4900         % and plural.
4901         \bool_lazy_or:nnTF
4902           { \l\_zrefclever_cap_bool }
4903           {
4904             \l\_zrefclever_capfirst_bool &&
4905             \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 }
4906           }
4907           { \tl_set:Nn \l\_zrefclever_name_format_tl {Name} }
4908           { \tl_set:Nn \l\_zrefclever_name_format_tl {name} }
4909         % If the queue is empty, we have a singular, otherwise, plural.
4910         \tl_if_empty:NTF \l\_zrefclever_typeset_queue_curr_tl
4911           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -sg } }
4912           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -pl } }
4913         \bool_lazy_and:nnTF
4914           { \l\_zrefclever_abbrev_bool }
4915           {
4916             ! \int_compare_p:nNn
4917               { \l\_zrefclever_type_count_int } = { 0 } ||
4918             ! \l\_zrefclever_noabbrev_first_bool
4919           }
4920           {
4921             \tl_set:NV \l\_zrefclever_name_format_fallback_tl
4922               \l\_zrefclever_name_format_tl
4923             \tl_put_right:Nn \l\_zrefclever_name_format_tl { -ab }

```

```

4924     }
4925     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4926
4927 % Handle number and gender nudges.
4928 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4929 {
4930     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4931     {
4932         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4933         {
4934             \msg_warning:nxx { zref-clever }
4935             { nudge-plural-when-sg }
4936             { \l__zrefclever_type_first_label_type_tl }
4937         }
4938     }
4939     {
4940         \bool_lazy_all:nT
4941         {
4942             { \l__zrefclever_nudge_comptosing_bool }
4943             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4944             {
4945                 \int_compare_p:nNn
4946                 { \l__zrefclever_label_count_int } > { 0 }
4947             }
4948         }
4949         {
4950             \msg_warning:nxx { zref-clever }
4951             { nudge-comptosing }
4952             { \l__zrefclever_type_first_label_type_tl }
4953         }
4954     }
4955     \bool_lazy_and:nnT
4956     { \l__zrefclever_nudge_gender_bool }
4957     { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4958     {
4959         \__zrefclever_get_rf_opt_seq:nxxN { gender }
4960         { \l__zrefclever_type_first_label_type_tl }
4961         { \l__zrefclever_ref_language_tl }
4962         \l__zrefclever_type_name_gender_seq
4963         \seq_if_in:NVF
4964         \l__zrefclever_type_name_gender_seq
4965         \l__zrefclever_ref_gender_tl
4966         {
4967             \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4968             {
4969                 \msg_warning:nxxxx { zref-clever }
4970                 { nudge-gender-not-declared-for-type }
4971                 { \l__zrefclever_ref_gender_tl }
4972                 { \l__zrefclever_type_first_label_type_tl }
4973                 { \l__zrefclever_ref_language_tl }
4974             }
4975             {
4976                 \msg_warning:nxxxxx { zref-clever }
4977                 { nudge-gender-mismatch }

```

```

4978         { \l__zrefclever_type_first_label_type_tl }
4979         { \l__zrefclever_ref_gender_tl }
4980         {
4981             \seq_use:Nn
4982             \l__zrefclever_type_name_gender_seq { , ~ }
4983         }
4984         { \l__zrefclever_ref_language_tl }
4985     }
4986 }
4987 }
4988 }
4989
4990 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4991 {
4992     \__zrefclever_opt_tl_get:cNF
4993     {
4994         \__zrefclever_opt_varname_type:een
4995         { \l__zrefclever_type_first_label_type_tl }
4996         { \l__zrefclever_name_format_tl }
4997         { tl }
4998     }
4999     \l__zrefclever_type_name_tl
5000     {
5001         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5002         {
5003             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5004             \tl_put_left:NV \l__zrefclever_name_format_tl
5005             \l__zrefclever_ref_decl_case_tl
5006         }
5007         \__zrefclever_opt_tl_get:cNF
5008         {
5009             \__zrefclever_opt_varname_lang_type:een
5010             { \l__zrefclever_ref_language_tl }
5011             { \l__zrefclever_type_first_label_type_tl }
5012             { \l__zrefclever_name_format_tl }
5013             { tl }
5014         }
5015         \l__zrefclever_type_name_tl
5016         {
5017             \tl_clear:N \l__zrefclever_type_name_tl
5018             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5019             \msg_warning:nxxx { zref-clever } { missing-name }
5020             { \l__zrefclever_name_format_tl }
5021             { \l__zrefclever_type_first_label_type_tl }
5022         }
5023     }
5024 }
5025 {
5026     \__zrefclever_opt_tl_get:cNF
5027     {
5028         \__zrefclever_opt_varname_type:een
5029         { \l__zrefclever_type_first_label_type_tl }
5030         { \l__zrefclever_name_format_tl }
5031         { tl }

```

```

5032 }
5033 \l__zrefclever_type_name_tl
5034 {
5035   \__zrefclever_opt_tl_get:cNF
5036   {
5037     \__zrefclever_opt_varname_type:een
5038     { \l__zrefclever_type_first_label_type_tl }
5039     { \l__zrefclever_name_format_fallback_tl }
5040     { tl }
5041   }
5042   \l__zrefclever_type_name_tl
5043   {
5044     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5045     {
5046       \tl_put_left:Nn
5047         \l__zrefclever_name_format_tl { - }
5048       \tl_put_left:NV \l__zrefclever_name_format_tl
5049         \l__zrefclever_ref_decl_case_tl
5050       \tl_put_left:Nn
5051         \l__zrefclever_name_format_fallback_tl { - }
5052       \tl_put_left:NV
5053         \l__zrefclever_name_format_fallback_tl
5054         \l__zrefclever_ref_decl_case_tl
5055     }
5056     \__zrefclever_opt_tl_get:cNF
5057     {
5058       \__zrefclever_opt_varname_lang_type:eeen
5059       { \l__zrefclever_ref_language_tl }
5060       { \l__zrefclever_type_first_label_type_tl }
5061       { \l__zrefclever_name_format_tl }
5062       { tl }
5063     }
5064     \l__zrefclever_type_name_tl
5065     {
5066       \__zrefclever_opt_tl_get:cNF
5067       {
5068         \__zrefclever_opt_varname_lang_type:eeen
5069         { \l__zrefclever_ref_language_tl }
5070         { \l__zrefclever_type_first_label_type_tl }
5071         { \l__zrefclever_name_format_fallback_tl }
5072         { tl }
5073       }
5074       \l__zrefclever_type_name_tl
5075       {
5076         \tl_clear:N \l__zrefclever_type_name_tl
5077         \bool_set_true:N
5078           \l__zrefclever_type_name_missing_bool
5079         \msg_warning:nxxx { zref-clever }
5080           { missing-name }
5081         { \l__zrefclever_name_format_tl }
5082         { \l__zrefclever_type_first_label_type_tl }
5083       }
5084     }
5085   }

```



```

5086     }
5087   }
5088 }
5089 }
5090
5091 % Signal whether the type name is to be included in the hyperlink or not.
5092 \bool_lazy_any:nTF
5093 {
5094   { ! \l__zrefclever_hyperlink_bool }
5095   { \l__zrefclever_link_star_bool }
5096   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5097   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5098 }
5099 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5100 {
5101   \bool_lazy_any:nTF
5102   {
5103     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5104     {
5105       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5106       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5107     }
5108     {
5109       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5110       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5111       \l__zrefclever_typeset_last_bool &&
5112       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5113     }
5114   }
5115   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5116   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5117 }
5118 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_hyperlink:nnn This avoids using the internal \hyper@@link, using only public hyperref commands (see <https://github.com/latex3/hyperref/issues/229#issuecomment-1093870142>, thanks Ulrike Fisher).

```

\__zrefclever_hyperlink:nnn {\url/file}\{\anchor}\{\text}
5119 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5120 {
5121   \tl_if_empty:nTF {#1}
5122   { \hyperlink {#2} {#3} }
5123   { \hyper@linkfile {#3} {#1} {#2} }
5124 }

```

(End definition for __zrefclever_hyperlink:nnn.)

__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.

```

5125 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5126 {
5127   \zref@ifpropundefined { urluse }
5128   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5129   {
5130     \zref@ifrefcontainsprop {#1} { urluse }
5131     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5132     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5133   }
5134 }
5135 \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}

5136 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5137 {
5138   \exp_args:Nxx \tl_if_eq:nnT
5139   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5140   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5141   {
5142     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5143     {
5144       \exp_args:Nxx \tl_if_eq:nnT
5145       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5146       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5147       {
5148         \int_compare:nNnTF
5149         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5150         =
5151         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5152         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5153         {
5154           \int_compare:nNnTF
5155           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5156           =
5157           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5158           {
5159             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5160             \bool_set_true:N \l__zrefclever_next_is_same_bool
5161           }
5162         }
5163       }
5164     }
5165   }
5166   \exp_args:Nxx \tl_if_eq:nnT

```

```

5167 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5168 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5169 {
5170   \exp_args:Nxx \tl_if_eq:nnT
5171   { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5172   { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5173   {
5174     \int_compare:nNnTF
5175     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5176     =
5177     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5178     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5179     {
5180       \int_compare:nNnT
5181       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5182       =
5183       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5184       {

```

If `zc@counters` are equal, `zc@enclvals` are equal, and `zc@enclvals` are equal, but the references themselves are different, this means that `\@currentlabel` has somehow been set manually (e.g. by an `amsmath`'s `\tag`), in which case we have no idea what's in there, and we should not even consider this is still a range. If they are equal, though, of course it is a range, and it is the same.

```

5185   \exp_args:Nxx \tl_if_eq:nnT
5186   {
5187     \_zrefclever_extract_unexp:nvn {#1}
5188     { l__zrefclever_ref_property_tl } { }
5189   }
5190   {
5191     \_zrefclever_extract_unexp:nvn {#2}
5192     { l__zrefclever_ref_property_tl } { }
5193   }
5194   {
5195     \bool_set_true:N
5196     \l__zrefclever_next_maybe_range_bool
5197     \bool_set_true:N
5198     \l__zrefclever_next_is_same_bool
5199   }
5200 }
5201 }
5202 }
5203 }
5204 }
5205 }
5206 }

```

(End definition for `_zrefclever_labels_in_sequence:nn`.)

Finally, some functions for retrieving reference options values, according to the relevant precedence rules. They receive an *option* as argument, and store the retrieved value in an appropriate *variable*. The difference between each of these functions is the data type of the option each should be used for.

```

\__zrefclever_get_rf_opt_tl:nnnN
\__zrefclever_get_rf_opt_tl:nnnN {<option>}
  {<ref type>} {<language>} {<tl variable>}
5207 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5208 {
5209   % First attempt: general options.
5210   \__zrefclever_opt_tl_get:cNF
5211   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5212   #4
5213   {
5214     % If not found, try type specific options.
5215     \__zrefclever_opt_tl_get:cNF
5216     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5217     #4
5218     {
5219       % If not found, try type- and language-specific.
5220       \__zrefclever_opt_tl_get:cNF
5221       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5222       #4
5223       {
5224         % If not found, try language-specific default.
5225         \__zrefclever_opt_tl_get:cNF
5226         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5227         #4
5228         {
5229           % If not found, try fallback.
5230           \__zrefclever_opt_tl_get:cNF
5231           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5232           #4
5233           { \tl_clear:N #4 }
5234         }
5235       }
5236     }
5237   }
5238 }
5239 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { nxxN }
(End definition for \__zrefclever_get_rf_opt_tl:nnnN.)

```

```

\__zrefclever_get_rf_opt_seq:nnnN
\__zrefclever_get_rf_opt_seq:nnnN {<option>}
  {<ref type>} {<language>} {<seq variable>}
5240 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5241 {
5242   % First attempt: general options.
5243   \__zrefclever_opt_seq_get:cNF
5244   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5245   #4
5246   {
5247     % If not found, try type specific options.
5248     \__zrefclever_opt_seq_get:cNF
5249     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5250     #4
5251     {
5252       % If not found, try type- and language-specific.
5253       \__zrefclever_opt_seq_get:cNF

```

```

5254         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5255         #4
5256         {
5257             % If not found, try language-specific default.
5258             \__zrefclever_opt_seq_get:cNF
5259             { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5260             #4
5261             {
5262                 % If not found, try fallback.
5263                 \__zrefclever_opt_seq_get:cNF
5264                 { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5265                 #4
5266                 { \seq_clear:N #4 }
5267             }
5268         }
5269     }
5270 }
5271 }
5272 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { nxxN }

```

(End definition for __zrefclever_get_rf_opt_seq:nnnN.)

```

\__zrefclever_get_rf_opt_bool:nnnnN

\__zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
{<ref type>} {<language>} {<bool variable>}

5273 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5274 {
5275     % First attempt: general options.
5276     \__zrefclever_opt_bool_get:cNF
5277     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5278     #5
5279     {
5280         % If not found, try type specific options.
5281         \__zrefclever_opt_bool_get:cNF
5282         { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5283         #5
5284         {
5285             % If not found, try type- and language-specific.
5286             \__zrefclever_opt_bool_get:cNF
5287             { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5288             #5
5289             {
5290                 % If not found, try language-specific default.
5291                 \__zrefclever_opt_bool_get:cNF
5292                 { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5293                 #5
5294                 {
5295                     % If not found, try fallback.
5296                     \__zrefclever_opt_bool_get:cNF
5297                     { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5298                     #5
5299                     { \use:c { bool_set_ #2 :N } #5 }
5300                 }
5301             }
5302         }

```

```

5303     }
5304   }
5305   \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nxxxN }

```

(End definition for __zrefclever_get_rf_opt_bool:nnnnN.)

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.

9.1 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.

```

5306 \__zrefclever_compat_module:nn { appendix }
5307 {
5308   \AddToHook { cmd / appendix / before }
5309   {
5310     \__zrefclever_zcsetup:n
5311     {
5312       countertype =
5313       {
5314         chapter      = appendix ,
5315         section      = appendix ,
5316         subsection   = appendix ,
5317         subsubsection = appendix ,
5318         paragraph    = appendix ,
5319         subparagraph = appendix ,
5320       }
5321     }
5322   }
5323 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmds` (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>, with a detailed explanation and possible workaround by Phelype Oleinik). The 2021-11-15 kernel release already handles this gracefully, thanks to fix by Phelype Oleinik at <https://github.com/latex3/latex2e/pull/699>.

9.2 appendices

This module applies both to the appendix package, and to the memoir class, since it “emulates” the package.

```

5324 \__zrefclever_compat_module:nn { appendices }
5325 {
5326   \__zrefclever_if_package_loaded:nT { appendix }
5327   {
5328     \newcounter { zc@appendix }
5329     \newcounter { zc@save@appendix }
5330     \setcounter { zc@appendix } { 0 }
5331     \setcounter { zc@save@appendix } { 0 }
5332     \cs_if_exist:cTF { chapter }
5333     {
5334       \__zrefclever_zcsetup:n
5335       { counterresetby = { chapter = zc@appendix } }
5336     }
5337     {
5338       \cs_if_exist:cT { section }
5339       {
5340         \__zrefclever_zcsetup:n
5341         { counterresetby = { section = zc@appendix } }
5342       }
5343     }
5344     \AddToHook { env / appendices / begin }
5345     {
5346       \stepcounter { zc@save@appendix }
5347       \setcounter { zc@appendix } { \value { zc@save@appendix } }
5348       \__zrefclever_zcsetup:n
5349       {
5350         countertype =
5351         {
5352           chapter      = appendix ,
5353           section      = appendix ,
5354           subsection   = appendix ,
5355           subsubsection = appendix ,
5356           paragraph    = appendix ,
5357           subparagraph = appendix ,
5358         }
5359       }
5360     }
5361     \AddToHook { env / appendices / end }
5362     { \setcounter { zc@appendix } { 0 } }
5363     \AddToHook { cmd / appendix / before }
5364     {
5365       \stepcounter { zc@save@appendix }
5366       \setcounter { zc@appendix } { \value { zc@save@appendix } }
5367     }

```

```

5368 \AddToHook { env / subappendices / begin }
5369 {
5370     \__zrefclever_zcsetup:n
5371     {
5372         countertype =
5373         {
5374             section      = appendix ,
5375             subsection   = appendix ,
5376             subsubsection = appendix ,
5377             paragraph    = appendix ,
5378             subparagraph = appendix ,
5379         } ,
5380     }
5381 }
5382 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5383 }
5384 }

```

9.3 memoir

The `memoir` document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. Some of them are implemented in ways which make difficult the use of `zref`, particularly `\zlabel`, short of redefining the whole stuff ourselves. Hopefully, these features are specialized enough to make `zref-clever` useful enough with `memoir` without much friction, but unless some support is added upstream, it is difficult not to be a little intrusive here.

1. Caption functionality which receives $\langle label \rangle$ as optional argument, namely:

- (a) The `sidecaption` and `sidecontcaption` environments. These environments *store* the label in an internal macro, `\m@mscaplabel`, at the begin environment code (more precisely in `\@@sidecaption`), but both the call to `\refstepcounter` and the expansion of `\m@mscaplabel` take place at `\endsidecaption`. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the $\langle label \rangle$ argument to start with. I can see two ways to deal with these environments, none of which I really like. First, map through `\m@mscaplabel` until `\label` is found, then grab the next token which is the $\langle label \rangle$. This can be used to set a `\zlabel` either with a kernel environment hook, or with `\@mem@scap@afterhook` (the former requires running `\refstepcounter` on our own, since the `env/.../end` hook comes before this is done by `\endsidecaption`). Second, locally redefine `\label` to set both labels inside the environments.
- (b) The bilingual caption commands: `\bitwonumcaption`, `\bionenumcaption`, and `\bicaption`. These commands do not support setting the label in their arguments (the labels do get set, but they end up included in the `title` property of the label too). So we do the same for them as for `sidecaption`, just taking care of grouping, since we can't count on the convenience of the environment hook (luckily for us, they are scoped themselves, so we can add an extra group there).

2. The `\subcaptionref` command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside

the $\langle subtitle \rangle$ argument of the subcaptioning commands, namely: `\subcaption`, `\contsubcaption`, `\subbottom`, `\contsubbottom`, `\subtop`. This functionality is implemented by `memoir` by setting a *second label* with prefix `sub@ $\langle label \rangle$` , and storing there just that part of interest. With `zref` this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find a place to hook into to add the property to the `main` list, since `memoir` does not really consider the possibility of some other command setting labels. `\@memsubcaption` is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.

3. `memoir`'s `\footnote`, `\verbfootnote`, `\sidefootnote` and `\pagenote`, just as the regular `\footnote` until recently in the kernel, do not set `\@currentcounter` alongside `\@currentlabel`, proper referencing to them requires setting the type for it.
4. Note that `memoir`'s appendix features “emulates” the `appendix` package, hence the corresponding compatibility module is loaded for `memoir` even if that package is not itself loaded. The same is true for the `\appendix` command module, since it is also defined.

```

5385 \__zrefclever_compat_module:n { memoir }
5386 {
5387   \__zrefclever_if_class_loaded:nT { memoir }
5388   {

```

Add subfigure and subtable support out of the box. Technically, this is not “default” behavior for `memoir`, users have to enable it with `\newsfloat`, but let this be smooth. Still, this does not cover any other floats created with `\newfloat`. Also include setup for `verse`.

```

5389   \__zrefclever_zcsetup:n
5390   {
5391     countertype =
5392     {
5393       subfigure = figure ,
5394       subtable  = table ,
5395       poemline  = line ,
5396     } ,
5397     counterresetby =
5398     {
5399       subfigure = figure ,
5400       subtable  = table ,
5401     } ,
5402   }

```

Support for caption `memoir` features that require that $\langle label \rangle$ be supplied as an optional argument.

```

5403   \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5404   {
5405     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5406     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5407     {
5408       \__zrefclever_memoir_orig_label:n {##1}
5409       \zlabel{##1}
5410     }
5411     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n

```

```

5412     }
5413     \AddToHook { env / sidecaption / begin }
5414     { \_zrefclever_memoir_both_labels: }
5415     \AddToHook { env / sidecontcaption / begin }
5416     { \_zrefclever_memoir_both_labels: }
5417     \AddToHook{ cmd / bitwonumcaption / before }
5418     { \group_begin: \_zrefclever_memoir_both_labels: }
5419     \AddToHook{ cmd / bitwonumcaption / after }
5420     { \group_end: }
5421     \AddToHook{ cmd / bionenumcaption / before }
5422     { \group_begin: \_zrefclever_memoir_both_labels: }
5423     \AddToHook{ cmd / bionenumcaption / after }
5424     { \group_end: }
5425     \AddToHook{ cmd / bicaption / before }
5426     { \group_begin: \_zrefclever_memoir_both_labels: }
5427     \AddToHook{ cmd / bicaption / after }
5428     { \group_end: }

```

Support for subcaption reference.

```

5429     \zref@newprop { subcaption }
5430     { \cs_if_exist_use:c { @@thesub \@capttype } }
5431     \AddToHook{ cmd / @memsubcaption / before }
5432     { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

Support for \footnote, \verbfootnote, \sidefootnote, and \pagenote.

```

5433     \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5434     \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5435     \AddToHook { env / minipage / begin }
5436     { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5437     \AddToHook { cmd / @makefntext / before }
5438     {
5439         \__zrefclever_zcsetup:x
5440         { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5441     }
5442     \AddToHook { cmd / @makesidefntext / before }
5443     { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5444     \__zrefclever_zcsetup:n
5445     {
5446         countertype =
5447         {
5448             sidefootnote = footnote ,
5449             pagenote = endnote ,
5450         } ,
5451     }
5452     \AddToHook { file / \jobname.ent / before }
5453     { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5454     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5455 }
5456 }

```

9.4 amsmath

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

```

5457 \__zrefclever_compat_module:nn { amsmath }

```

```

5458 {
5459   \__zrefclever_if_package_loaded:nT { amsmath }
5460   {

```

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...). See also <https://github.com/ho-tex/zref/issues/4> and <https://github.com/latex3/latex2e/issues/272>.

```

5461     \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5462     {
5463       \__zrefclever_orig_ltxlabel:n {#1}
5464       \zref@wrapper@babel \zref@label {#1}
5465     }

```

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` (though this has changed recently 2022-05-16, see <https://github.com/latex3/hyperref/commit/a011ba9308a1b047dc151796de557da0bb22abaa>), which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. Other classes packages also redefine `\ltx@label`, which may cause some trouble. A `grep` on `texmf-dist` returns hits for: `thm-restate.sty`, `smartref.sty`, `jmlrbook.cls`, `cleveref.sty`, `cryptocode.sty`, `nameref.sty`, `easyeqn.sty`, `empheq.sty`, `ntheorem.sty`, `nccmath.sty`, `nwejm.cls`, `nwejmart.cls`, `aguplus.sty`, `aguplus.cls`, `agupp.sty`, `amsmath.hyp`, `amsmath.sty` (surprise!), `amsmath.4ht`, `nameref.4ht`, `frenchle.sty`, `french.sty`, plus corresponding documentations and different versions of the same packages. That’s not too many, but not “just a few” either. The critical ones are explicitly handled here (`amsmath` itself, and `nameref`). A number of those I’m really not acquainted with. For `cleveref`, in particular, this procedure is not compatible with it. If we happen to come later than it and override its definition, this may be a substantial problem for `cleveref`, since it will find the label, but it won’t contain the data it is expecting. However, this should normally not occur, if the user has followed the documented recommendation for `cleveref` to load it last, or at least very late, and besides I don’t see much of an use case for using both `cleveref` and `zref-clever` together. I have documented in the user manual that this module may cause potential issues, and how to work around them. And I have made an upstream feature request for a hook, so that this could be made more cleanly at <https://github.com/latex3/hyperref/issues/212>.

```

5466   \__zrefclever_if_package_loaded:nTF { hyperref }
5467   {
5468     \AddToHook { package / nameref / after }
5469     {
5470       \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5471       \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5472     }
5473   }
5474   {
5475     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label

```

```

5476         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5477     }

```

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`. We cannot even set `\@currentcounter` at `env/.../begin`, since the call to `\refstepcounter{equation}` done by `subequations` will override that in sequence. Unfortunately, the suggestion to set `\@currentcounter` to `parentequation` here was not accepted, see <https://github.com/latex3/latex2e/issues/687#issuecomment-951451024> and subsequent discussion. So, for `subequations`, 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).

```

5478     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5479     \AddToHook { env / subequations / begin }
5480     {
5481         \__zrefclever_zcsetup:x
5482         {
5483             counterresetby =
5484             {
5485                 parentequation =
5486                 \__zrefclever_counter_reset_by:n { equation } ,
5487                 equation = parentequation ,
5488             } ,
5489             currentcounter = parentequation ,
5490             countertype = { parentequation = equation } ,
5491         }
5492     \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5493 }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout and does set `\@currentcounter` for `\tags`. But we still have to manually reset `currentcounter` to default because, 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. We also arrange, at this point, for the provision of the `subeq` property, for the convenience of referring to them directly or to build terse ranges with the `endrange` option.

```

5494     \zref@newprop { subeq } { \alph { equation } }
5495     \clist_map_inline:nn
5496     {
5497         equation ,
5498         equation* ,
5499         align ,
5500         align* ,
5501         alignat ,
5502         alignat* ,
5503         flalign ,

```

```

5504         flalign* ,
5505         xalignat ,
5506         xalignat* ,
5507         gather ,
5508         gather* ,
5509         multiline ,
5510         multiline* ,
5511     }
5512     {
5513         \AddToHook { env / #1 / begin }
5514         {
5515             \__zrefclever_zcsetup:n { currentcounter = equation }
5516             \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5517             { \zref@localaddprop \ZREF@mainlist { subeq } }
5518         }
5519     }
5520     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5521 }
5522 }

```

9.5 mathtools

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.

```

5523 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5524 \__zrefclever_compat_module:nn { mathtools }
5525 {
5526     \__zrefclever_if_package_loaded:nT { mathtools }
5527     {
5528         \MH_if_boolean:nT { show_only_refs }
5529         {
5530             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5531             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5532             {
5533                 \@bsphack
5534                 \seq_map_inline:Nn #1
5535                 {
5536                     \exp_args:Nx \tl_if_eq:nnTF
5537                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5538                     { equation }
5539                     {
5540                         \protected@write \@auxout { }
5541                         { \string \MT@newlabel {##1} }
5542                     }
5543                 }
5544                 \exp_args:Nx \tl_if_eq:nnT

```

```

5545         { \_zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5546     { parentequation }
5547     {
5548         \protected@write \@auxout { }
5549         { \string \MT@newlabel {##1} }
5550     }
5551 }
5552 }
5553 \@esphack
5554 }
5555 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5556 }
5557 }
5558 }

```

9.6 breqn

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for `\zlabel` just as well. However, if it happens not to work, there was no easy alternative handle I could find. In particular, it does not seem viable to leverage the `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

```

5559 \_zrefclever_compat_module:nn { breqn }
5560 {
5561     \_zrefclever_if_package_loaded:nT { breqn }
5562     {

```

Contrary to the practice in `amsmath`, which prints `\tag` even in unnumbered environments, the starred environments from `breqn` don’t typeset any tag/number at all, even for a manually given `number=` as an option. So, even if one can actually set a label in them, it is not really meaningful to make a reference to them. Also contrary to `amsmath`’s practice, `breqn` uses `\stepcounter` instead of `\refstepcounter` for incrementing the equation counters (see <https://tex.stackexchange.com/a/241150>).

```

5563     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5564     \AddToHook { env / dgroup / begin }
5565     {
5566         \_zrefclever_zcsetup:x
5567         {
5568             counterresetby =
5569             {
5570                 parentequation =
5571                 \_zrefclever_counter_reset_by:n { equation } ,
5572                 equation = parentequation ,
5573             } ,
5574             currentcounter = parentequation ,
5575             countertype = { parentequation = equation } ,
5576         }
5577         \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5578     }
5579     \zref@ifpropundefined { subeq }
5580     { \zref@newprop { subeq } { \alph { equation } } }
5581     { }

```

```

5582 \clist_map_inline:nn
5583 {
5584     dmath ,
5585     dseries ,
5586     darray ,
5587 }
5588 {
5589     \AddToHook { env / #1 / begin }
5590     {
5591         \__zrefclever_zcsetup:n { currentcounter = equation }
5592         \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5593         { \zref@localaddprop \ZREF@mainlist { subeq } }
5594     }
5595 }
5596 \msg_info:nnn { zref-clever } { compat-package } { breqn }
5597 }
5598 }

```

9.7 listings

```

5599 \__zrefclever_compat_module:nn { listings }
5600 {
5601     \__zrefclever_if_package_loaded:nT { listings }
5602     {
5603         \__zrefclever_zcsetup:n
5604         {
5605             countertype =
5606             {
5607                 lstlisting = listing ,
5608                 lstnumber = line ,
5609             } ,
5610             counterresetby = { lstnumber = lstlisting } ,
5611         }
5612     }
5613 }

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment. The *only* place to set this label is the `PreInit` hook. This hook, comes right after `\lst@MakeCaption` in `\lst@Init`, which runs `\refstepcounter` on `lstlisting`, so we must come after it. Also `listings` itself sets `\@currentlabel` to `\thelstnumber` in the `Init` hook, which comes right after the `PreInit` one in `\lst@Init`. Since, if we add to `Init` here, we go to the end of it, we'd be seeing the wrong `\@currentlabel` at that point.

```

5612 \lst@AddToHook { PreInit }
5613 { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }

```

Set `currentcounter` to `lstnumber` in the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` here. Note that `listings` *does use* `\refstepcounter` on `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. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Indeed, the fact that `listings` manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```

5614 \lst@AddToHook { Init }
5615 { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }

```

```

5616         \msg_info:nnn { zref-clever } { compat-package } { listings }
5617     }
5618 }

```

9.8 enumitem

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.

```

5619 \__zrefclever_compat_module:nn { enumitem }
5620 {
5621     \__zrefclever_if_package_loaded:nT { enumitem }
5622     {
5623         \int_set:Nn \l_tmpa_int { 5 }
5624         \bool_while_do:nn
5625         {
5626             \cs_if_exist_p:c
5627             { c@ enum \int_to_roman:n { \l_tmpa_int } }
5628         }
5629         {
5630             \__zrefclever_zcsetup:x
5631             {
5632                 counterresetby =
5633                 {
5634                     enum \int_to_roman:n { \l_tmpa_int } =
5635                     enum \int_to_roman:n { \l_tmpa_int - 1 }
5636                 } ,
5637                 countertype =
5638                 { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5639             }
5640             \int_incr:N \l_tmpa_int
5641         }
5642         \int_compare:nNnT { \l_tmpa_int } > { 5 }
5643         { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5644     }
5645 }

```

9.9 subcaption

```

5646 \__zrefclever_compat_module:nn { subcaption }
5647 {
5648     \__zrefclever_if_package_loaded:nT { subcaption }
5649     {

```



```

5650 \__zrefclever_zcsetup:n
5651 {
5652     countertype =
5653     {
5654         subfigure = figure ,
5655         subtable = table ,
5656     } ,
5657     counterresetby =
5658     {
5659         subfigure = figure ,
5660         subtable = table ,
5661     } ,
5662 }

```

Support for `subref` reference.

```

5663 \zref@newprop { subref }
5664 { \cs_if_exist_use:c { thesub \@captype } }
5665 \tl_put_right:Nn \caption@subtypetoolhook
5666 { \zref@localaddprop \ZREF@mainlist { subref } }
5667 }
5668 }

```

9.10 subfig

Though `subfig` offers `\subref` (as `subcaption`), I could not find any reasonable place to add the `subref` property to `zref`'s main list.

```

5669 \__zrefclever_compat_module:nn { subfig }
5670 {
5671     \__zrefclever_if_package_loaded:nT { subfig }
5672     {
5673         \__zrefclever_zcsetup:n
5674         {
5675             countertype =
5676             {
5677                 subfigure = figure ,
5678                 subtable = table ,
5679             } ,
5680             counterresetby =
5681             {
5682                 subfigure = figure ,
5683                 subtable = table ,
5684             } ,
5685         }
5686     }
5687 }
5688 </package>

```

10 Language files

Initial values for the English, German, French, Portuguese, and Spanish language files have been provided by the author. Translations available for document elements' names in other packages have been an useful reference for the purpose, namely: `babel`, `cleveref`, `translator`, and `translations`.

10.1 Localization guidelines

Since the task of localizing `zref-clever` to work in different languages depends on the generous work of contributors, it is a good idea to set some guidelines not only to ease the task itself but also to document what the package expects in this regard.

The first general observation is that, contrary to a common initial reaction of those faced with the task of localizing the reference types, is that the job is not quite one of “translation”. The reference type names are just the internal names used by the package to refer to them, technically, they could just as well be foobars. Of course, for practical reasons, they were chosen to be semantic. However, what we are searching for is not really the translation to the reference type name itself, but rather for the word / term / expression which is typically used to refer to the document object that the reference type is meant to represent. And terms that should work well in the contexts which cross-references are commonly used.

That said, some comments about the reference types and common pitfalls.

Sectioning: A number of reference types are provided to support referencing to document sectioning commands. Obviously, `part`, `chapter`, `section`, and `paragraph` are meant to refer to the sectioning commands of the standard classes and elsewhere, which anyone reading this is certainly acquainted with. Note that `zref-clever` uses – by default at least, which is what the language files cater for – the `section` reference type to refer to `\subsections` and `\subsubsections` as well, similarly, `paragraph` also refers to `\subparagraph`. The `appendix` reference type is meant to refer to any sectioning command – be them chapters, sections, or paragraphs – issued after `\appendix`, which corresponds to how the standard classes, the KOMA Script classes, and `memoir` deal with appendices. The `book` reference type deserves some explanation. The word “book” has a good number of meanings, and the most common one is not the one which is intended here. The Webster dictionary gives us a couple of definitions of interest: “1. A collection of sheets of paper, or similar material, blank, written, or printed, bound together; commonly, many folded and bound sheets containing continuous printing or writing.” and “3. A part or subdivision of a treatise or literary work; as, the tenth book of ‘Paradise Lost.’” It is this third meaning which the `book` reference type is meant to support: a major subdivision of a work, much like `\part`. Even if it does not exist in the standard classes, it may exist elsewhere, in particular, it is provided by `memoir`.

Common numbered objects: Nothing surprising here, just being explicit. `table` and `figure` refer to the document’s respective floats objects. `page` to the page number. `item` to the item number in `enumerate` environments. Similarly, `line` is meant to refer to line numbers.

Notes: `zref-clever` provides three reference types in this area: `footnote`, `endnote`, and `note`. The first two refer to footnotes and end notes, respectively. The third is meant as a convenience for a general “note” object, either the other two, or something else. By experience, here is one place where that initial observation of not simply translating the reference types names is particularly relevant. There’s a natural temptation, because three different types exist and are somewhat close to each other, to distinguish them clearly. Duty would compel us to do so. But that may lead to less than ideal results. Different terms work well for some languages, like English and German, which have compound words for the purpose. But less so for other languages, like Portuguese, French, or Italian. For example, in a document in French which only contains footnotes, arguably a very common use case, would it be better to refer to a footnote as just “note”, or be very precise with “note infrapaginale”? Of course, in a document which contains both footnotes and end notes, we may need the distinction. But is it really the better default?

True, possibly the inclusion of the `note` reference type, with no clear object to refer to, creates more noise than convenience here. If I recall correctly, my intention was to provide an easy way out for users from possible contentious localizations for `footnote` and `endnote`, but I'm not sure if it's been working like this in practice, and I should probably have refrained from adding it in the first place.

Math & Co.: A good number of reference types provided by the package are meant to cater for document objects commonly used in Mathematics and related areas. They are either straight math environments, defined by the kernel, `amsmath` or other packages, or environments which are normally not pre-defined by the kernel or the standard classes, but are traditionally defined by users with the kernel's `\newtheorem` or similar constructs available in the `LATEX` package ecosystem. For most of them, localization should strive as much as possible to use the formal terms, jargon really, typically employed by mathematicians, logicians, and friends. Namely for the reference types: `equation`, `theorem`, `lemma`, `corollary`, `proposition`, `definition`, `proof`, `result`, and `remark`. Regarding `example`, `exercise`, and `solution` being somewhat less formal is admissible. But the chosen terms should still be fit for use in Math related contexts, and should be assumed were created by `\newtheorem` or similar, even if users may well find other uses for these types.

Code: A couple of reference types are provided for code related environments: `algorithm` and `listing`. By experience, the `listing` type has already proven to be a particularly challenging one. Formally, it should be a good default term to encompass anything which may regularly be included in a `lstlisting` environment as provided by the `listings` package. However, it seems that in different languages it is quite difficult to find a satisfying term for it. Though my English is decent, I'm not a native speaker, still I'm not even sure how common the term is used for the purpose even in English. It seems to be traditional enough in the `LATEX` community at least. In doubt, pend to the jargon side, anglicism if need be. Since we are bound to displease mostly everyone anyway, at least we do so in a consistent manner.

Completeness and abbreviated forms: Ideally, the language file should be as complete as possible. "Complete" meaning it contains: i) the defaults for all basic separators, `namesep`, `pairsep`, `listsep`, `lastsep`, `tpairsep`, `tlistsep`, `tlastsep`, `notesep`, and `rangesep`; ii) the non-abbreviated forms of names for all the supported reference types, according to the language definitions, that is, usually for `Name-sg`, `name-sg`, `Name-pl`, `name-pl`, but only for the capitalized forms if the language was declared with `allcaps` option, and names for each declension case, if the language was declared with `declension`; iii) genders for each reference type, if the language was declared with `gender`. The language file may include some other things, like some type specific settings for separators or rebounds, and also some abbreviated name forms. In the case of abbreviated name forms, it is usual and desirable to provide some, but they should be used sparingly, only for cases where the abbreviation is a common and well established tradition for the language. The reason is that `abbrev=true` is quite a common use case, and it is easier to provide an occasional wanted abbreviated form, if the language file didn't include it, than it is to disable several unwanted ones, if the language file includes too many of them. What should be aimed at is to provide a good default abbreviations set. Unusual or disputable abbreviations should be avoided. In particular, there is no need at all to provide the same set of abbreviations for each language. It is not because English has them for a given type that some other language has to have them, and it is not because English lacks them for another type, that other languages shouldn't have them. Still, with regard to abbreviated forms, it is better to be conservative than opinionated.

babel names: As is known, `babel` defines a set of captions for different document objects for each supported language. In some cases, they intersect with the objects referred to with cross-references, in which case consistency with `babel` should be maintained as much as possible. This is specially the case for prominent and traditional objects, such as `\chaptername`, `\figurename`, `\tablename`, `\pagename`, `\partname`, and `\appendixname`. This is not set in stone, but there should be good reason to diverge from it. In particular, if a certain term is contentious in a given language, `babel`’s default should be preferred. For example, “table” vs. “tableau” in French, or “cuadro” vs. “tabla” in Spanish.

Input encoding of language files: When `zref-clever` was released, the \LaTeX kernel already used UTF-8 as default input encoding. Indeed, `zref-clever` requires a kernel even newer than the one where the default input encoding was changed. That given, UTF-8 input encoding was made a requirement of the package, and hence the language files should be in UTF-8, since it makes them easier to read and maintain than LICR.

Precedence rule for options in the language files: Any option given twice or more times has to have some precedence rule. Normally, the language files should not contain options in duplicity, but they may happen when setting some “group” `refbounds` options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that’s the point where we know from `babel` or `polyglossia` the `\language`. But we also don’t want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

zref-vario: If you are interested in the localization of `zref-clever` to your language, and willing to contribute to it, you may also want to consider doing the same for the companion package `zref-vario`. It is actually a much simpler task than localizing `zref-clever`.

10.2 English

English language file has been initially provided by the author.

```

5689 <*package>
5690 \zcDeclareLanguage { english }
5691 \zcDeclareLanguageAlias { american } { english }
5692 \zcDeclareLanguageAlias { australian } { english }
5693 \zcDeclareLanguageAlias { british } { english }
5694 \zcDeclareLanguageAlias { canadian } { english }
5695 \zcDeclareLanguageAlias { newzealand } { english }
5696 \zcDeclareLanguageAlias { UKenglish } { english }
5697 \zcDeclareLanguageAlias { USenglish } { english }
5698 </package>

5699 <*lang-english>

5700 namesep = {\nobreakspace} ,
5701 pairsep = {\~and\nobreakspace} ,
5702 listsep = {\~,~} ,
5703 lastsep = {\~and\nobreakspace} ,
5704 tpairsep = {\~and\nobreakspace} ,
5705 tlistsep = {\~,~} ,
5706 tlastsep = {\~,~and\nobreakspace} ,

```

```

5707 notesep    = {\~} ,
5708 rangesep    = {\~to\nobreakspace} ,
5709
5710 type = book ,
5711   Name-sg = Book ,
5712   name-sg = book ,
5713   Name-pl = Books ,
5714   name-pl = books ,
5715
5716 type = part ,
5717   Name-sg = Part ,
5718   name-sg = part ,
5719   Name-pl = Parts ,
5720   name-pl = parts ,
5721
5722 type = chapter ,
5723   Name-sg = Chapter ,
5724   name-sg = chapter ,
5725   Name-pl = Chapters ,
5726   name-pl = chapters ,
5727
5728 type = section ,
5729   Name-sg = Section ,
5730   name-sg = section ,
5731   Name-pl = Sections ,
5732   name-pl = sections ,
5733
5734 type = paragraph ,
5735   Name-sg = Paragraph ,
5736   name-sg = paragraph ,
5737   Name-pl = Paragraphs ,
5738   name-pl = paragraphs ,
5739   Name-sg-ab = Par. ,
5740   name-sg-ab = par. ,
5741   Name-pl-ab = Par. ,
5742   name-pl-ab = par. ,
5743
5744 type = appendix ,
5745   Name-sg = Appendix ,
5746   name-sg = appendix ,
5747   Name-pl = Appendices ,
5748   name-pl = appendices ,
5749
5750 type = page ,
5751   Name-sg = Page ,
5752   name-sg = page ,
5753   Name-pl = Pages ,
5754   name-pl = pages ,
5755   rangesep = {\textendash} ,
5756   rangetopair = false ,
5757
5758 type = line ,
5759   Name-sg = Line ,
5760   name-sg = line ,

```

```

5761     Name-pl = Lines ,
5762     name-pl = lines ,
5763
5764 type = figure ,
5765     Name-sg = Figure ,
5766     name-sg = figure ,
5767     Name-pl = Figures ,
5768     name-pl = figures ,
5769     Name-sg-ab = Fig. ,
5770     name-sg-ab = fig. ,
5771     Name-pl-ab = Figs. ,
5772     name-pl-ab = figs. ,
5773
5774 type = table ,
5775     Name-sg = Table ,
5776     name-sg = table ,
5777     Name-pl = Tables ,
5778     name-pl = tables ,
5779
5780 type = item ,
5781     Name-sg = Item ,
5782     name-sg = item ,
5783     Name-pl = Items ,
5784     name-pl = items ,
5785
5786 type = footnote ,
5787     Name-sg = Footnote ,
5788     name-sg = footnote ,
5789     Name-pl = Footnotes ,
5790     name-pl = footnotes ,
5791
5792 type = endnote ,
5793     Name-sg = Note ,
5794     name-sg = note ,
5795     Name-pl = Notes ,
5796     name-pl = notes ,
5797
5798 type = note ,
5799     Name-sg = Note ,
5800     name-sg = note ,
5801     Name-pl = Notes ,
5802     name-pl = notes ,
5803
5804 type = equation ,
5805     Name-sg = Equation ,
5806     name-sg = equation ,
5807     Name-pl = Equations ,
5808     name-pl = equations ,
5809     Name-sg-ab = Eq. ,
5810     name-sg-ab = eq. ,
5811     Name-pl-ab = Eqs. ,
5812     name-pl-ab = eqs. ,
5813     refbounds-first-sg = {(,),(,)} ,
5814     refbounds = {(,,)} ,

```

```

5815
5816 type = theorem ,
5817     Name-sg = Theorem ,
5818     name-sg = theorem ,
5819     Name-pl = Theorems ,
5820     name-pl = theorems ,
5821
5822 type = lemma ,
5823     Name-sg = Lemma ,
5824     name-sg = lemma ,
5825     Name-pl = Lemmas ,
5826     name-pl = lemmas ,
5827
5828 type = corollary ,
5829     Name-sg = Corollary ,
5830     name-sg = corollary ,
5831     Name-pl = Corollaries ,
5832     name-pl = corollaries ,
5833
5834 type = proposition ,
5835     Name-sg = Proposition ,
5836     name-sg = proposition ,
5837     Name-pl = Propositions ,
5838     name-pl = propositions ,
5839
5840 type = definition ,
5841     Name-sg = Definition ,
5842     name-sg = definition ,
5843     Name-pl = Definitions ,
5844     name-pl = definitions ,
5845
5846 type = proof ,
5847     Name-sg = Proof ,
5848     name-sg = proof ,
5849     Name-pl = Proofs ,
5850     name-pl = proofs ,
5851
5852 type = result ,
5853     Name-sg = Result ,
5854     name-sg = result ,
5855     Name-pl = Results ,
5856     name-pl = results ,
5857
5858 type = remark ,
5859     Name-sg = Remark ,
5860     name-sg = remark ,
5861     Name-pl = Remarks ,
5862     name-pl = remarks ,
5863
5864 type = example ,
5865     Name-sg = Example ,
5866     name-sg = example ,
5867     Name-pl = Examples ,
5868     name-pl = examples ,

```

```

5869
5870 type = algorithm ,
5871   Name-sg = Algorithm ,
5872   name-sg = algorithm ,
5873   Name-pl = Algorithms ,
5874   name-pl = algorithms ,
5875
5876 type = listing ,
5877   Name-sg = Listing ,
5878   name-sg = listing ,
5879   Name-pl = Listings ,
5880   name-pl = listings ,
5881
5882 type = exercise ,
5883   Name-sg = Exercise ,
5884   name-sg = exercise ,
5885   Name-pl = Exercises ,
5886   name-pl = exercises ,
5887
5888 type = solution ,
5889   Name-sg = Solution ,
5890   name-sg = solution ,
5891   Name-pl = Solutions ,
5892   name-pl = solutions ,
5893  $\langle$ /lang-english $\rangle$ 

```

10.3 German

German language file has been initially provided by the author.

`babel-german` also has `.ldfs` for `germanb` and `ngermanb`, but they are deprecated as options and, if used, they fall back respectively to `german` and `ngerman`.

```

5894  $\langle$ *package $\rangle$ 
5895 \zcDeclareLanguage
5896 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5897 { german }
5898 \zcDeclareLanguageAlias { ngerman      } { german }
5899 \zcDeclareLanguageAlias { austrian     } { german }
5900 \zcDeclareLanguageAlias { naustrian    } { german }
5901 \zcDeclareLanguageAlias { swissgerman  } { german }
5902 \zcDeclareLanguageAlias { nswissgerman } { german }
5903  $\langle$ /package $\rangle$ 
5904  $\langle$ *lang-german $\rangle$ 
5905 namesep = {\nobreakspace} ,
5906 pairsep  = {\~und\nobreakspace} ,
5907 listsep  = { , ~ } ,
5908 lastsep  = {\~und\nobreakspace} ,
5909 tpairsep = {\~und\nobreakspace} ,
5910 tlistsep = { , ~ } ,
5911 tlastsep = {\~und\nobreakspace} ,
5912 notesep  = { ~ } ,
5913 rangesep = {\~bis\nobreakspace} ,
5914

```



```

5915 type = book ,
5916     gender = n ,
5917     case = N ,
5918         Name-sg = Buch ,
5919         Name-pl = Bücher ,
5920     case = A ,
5921         Name-sg = Buch ,
5922         Name-pl = Bücher ,
5923     case = D ,
5924         Name-sg = Buch ,
5925         Name-pl = Büchern ,
5926     case = G ,
5927         Name-sg = Buches ,
5928         Name-pl = Bücher ,
5929
5930 type = part ,
5931     gender = m ,
5932     case = N ,
5933         Name-sg = Teil ,
5934         Name-pl = Teile ,
5935     case = A ,
5936         Name-sg = Teil ,
5937         Name-pl = Teile ,
5938     case = D ,
5939         Name-sg = Teil ,
5940         Name-pl = Teilen ,
5941     case = G ,
5942         Name-sg = Teiles ,
5943         Name-pl = Teile ,
5944
5945 type = chapter ,
5946     gender = n ,
5947     case = N ,
5948         Name-sg = Kapitel ,
5949         Name-pl = Kapitel ,
5950     case = A ,
5951         Name-sg = Kapitel ,
5952         Name-pl = Kapitel ,
5953     case = D ,
5954         Name-sg = Kapitel ,
5955         Name-pl = Kapiteln ,
5956     case = G ,
5957         Name-sg = Kapitels ,
5958         Name-pl = Kapitel ,
5959
5960 type = section ,
5961     gender = m ,
5962     case = N ,
5963         Name-sg = Abschnitt ,
5964         Name-pl = Abschnitte ,
5965     case = A ,
5966         Name-sg = Abschnitt ,
5967         Name-pl = Abschnitte ,
5968     case = D ,

```

```

5969     Name-sg = Abschnitt ,
5970     Name-pl = Abschnitten ,
5971     case = G ,
5972     Name-sg = Abschnitts ,
5973     Name-pl = Abschnitte ,
5974
5975 type = paragraph ,
5976     gender = m ,
5977     case = N ,
5978     Name-sg = Absatz ,
5979     Name-pl = Absätze ,
5980     case = A ,
5981     Name-sg = Absatz ,
5982     Name-pl = Absätze ,
5983     case = D ,
5984     Name-sg = Absatz ,
5985     Name-pl = Absätzen ,
5986     case = G ,
5987     Name-sg = Absatzes ,
5988     Name-pl = Absätze ,
5989
5990 type = appendix ,
5991     gender = m ,
5992     case = N ,
5993     Name-sg = Anhang ,
5994     Name-pl = Anhänge ,
5995     case = A ,
5996     Name-sg = Anhang ,
5997     Name-pl = Anhänge ,
5998     case = D ,
5999     Name-sg = Anhang ,
6000     Name-pl = Anhängen ,
6001     case = G ,
6002     Name-sg = Anhangs ,
6003     Name-pl = Anhänge ,
6004
6005 type = page ,
6006     gender = f ,
6007     case = N ,
6008     Name-sg = Seite ,
6009     Name-pl = Seiten ,
6010     case = A ,
6011     Name-sg = Seite ,
6012     Name-pl = Seiten ,
6013     case = D ,
6014     Name-sg = Seite ,
6015     Name-pl = Seiten ,
6016     case = G ,
6017     Name-sg = Seite ,
6018     Name-pl = Seiten ,
6019     rangsep = {\textendash} ,
6020     rangetopair = false ,
6021
6022 type = line ,

```

```

6023 gender = f ,
6024 case = N ,
6025     Name-sg = Zeile ,
6026     Name-pl = Zeilen ,
6027 case = A ,
6028     Name-sg = Zeile ,
6029     Name-pl = Zeilen ,
6030 case = D ,
6031     Name-sg = Zeile ,
6032     Name-pl = Zeilen ,
6033 case = G ,
6034     Name-sg = Zeile ,
6035     Name-pl = Zeilen ,
6036
6037 type = figure ,
6038 gender = f ,
6039 case = N ,
6040     Name-sg = Abbildung ,
6041     Name-pl = Abbildungen ,
6042     Name-sg-ab = Abb. ,
6043     Name-pl-ab = Abb. ,
6044 case = A ,
6045     Name-sg = Abbildung ,
6046     Name-pl = Abbildungen ,
6047     Name-sg-ab = Abb. ,
6048     Name-pl-ab = Abb. ,
6049 case = D ,
6050     Name-sg = Abbildung ,
6051     Name-pl = Abbildungen ,
6052     Name-sg-ab = Abb. ,
6053     Name-pl-ab = Abb. ,
6054 case = G ,
6055     Name-sg = Abbildung ,
6056     Name-pl = Abbildungen ,
6057     Name-sg-ab = Abb. ,
6058     Name-pl-ab = Abb. ,
6059
6060 type = table ,
6061 gender = f ,
6062 case = N ,
6063     Name-sg = Tabelle ,
6064     Name-pl = Tabellen ,
6065 case = A ,
6066     Name-sg = Tabelle ,
6067     Name-pl = Tabellen ,
6068 case = D ,
6069     Name-sg = Tabelle ,
6070     Name-pl = Tabellen ,
6071 case = G ,
6072     Name-sg = Tabelle ,
6073     Name-pl = Tabellen ,
6074
6075 type = item ,
6076 gender = m ,

```

```

6077 case = N ,
6078     Name-sg = Punkt ,
6079     Name-pl = Punkte ,
6080 case = A ,
6081     Name-sg = Punkt ,
6082     Name-pl = Punkte ,
6083 case = D ,
6084     Name-sg = Punkt ,
6085     Name-pl = Punkten ,
6086 case = G ,
6087     Name-sg = Punktes ,
6088     Name-pl = Punkte ,
6089
6090 type = footnote ,
6091     gender = f ,
6092     case = N ,
6093         Name-sg = Fußnote ,
6094         Name-pl = Fußnoten ,
6095     case = A ,
6096         Name-sg = Fußnote ,
6097         Name-pl = Fußnoten ,
6098     case = D ,
6099         Name-sg = Fußnote ,
6100         Name-pl = Fußnoten ,
6101     case = G ,
6102         Name-sg = Fußnote ,
6103         Name-pl = Fußnoten ,
6104
6105 type = endnote ,
6106     gender = f ,
6107     case = N ,
6108         Name-sg = Endnote ,
6109         Name-pl = Endnoten ,
6110     case = A ,
6111         Name-sg = Endnote ,
6112         Name-pl = Endnoten ,
6113     case = D ,
6114         Name-sg = Endnote ,
6115         Name-pl = Endnoten ,
6116     case = G ,
6117         Name-sg = Endnote ,
6118         Name-pl = Endnoten ,
6119
6120 type = note ,
6121     gender = f ,
6122     case = N ,
6123         Name-sg = Anmerkung ,
6124         Name-pl = Anmerkungen ,
6125     case = A ,
6126         Name-sg = Anmerkung ,
6127         Name-pl = Anmerkungen ,
6128     case = D ,
6129         Name-sg = Anmerkung ,
6130         Name-pl = Anmerkungen ,

```

```

6131 case = G ,
6132     Name-sg = Anmerkung ,
6133     Name-pl = Anmerkungen ,
6134
6135 type = equation ,
6136     gender = f ,
6137     case = N ,
6138     Name-sg = Gleichung ,
6139     Name-pl = Gleichungen ,
6140     case = A ,
6141     Name-sg = Gleichung ,
6142     Name-pl = Gleichungen ,
6143     case = D ,
6144     Name-sg = Gleichung ,
6145     Name-pl = Gleichungen ,
6146     case = G ,
6147     Name-sg = Gleichung ,
6148     Name-pl = Gleichungen ,
6149     refbounds-first-sg = {,(,)},
6150     refbounds = {(,,)} ,
6151
6152 type = theorem ,
6153     gender = n ,
6154     case = N ,
6155     Name-sg = Theorem ,
6156     Name-pl = Theoreme ,
6157     case = A ,
6158     Name-sg = Theorem ,
6159     Name-pl = Theoreme ,
6160     case = D ,
6161     Name-sg = Theorem ,
6162     Name-pl = Theoremen ,
6163     case = G ,
6164     Name-sg = Theorems ,
6165     Name-pl = Theoreme ,
6166
6167 type = lemma ,
6168     gender = n ,
6169     case = N ,
6170     Name-sg = Lemma ,
6171     Name-pl = Lemmata ,
6172     case = A ,
6173     Name-sg = Lemma ,
6174     Name-pl = Lemmata ,
6175     case = D ,
6176     Name-sg = Lemma ,
6177     Name-pl = Lemmata ,
6178     case = G ,
6179     Name-sg = Lemmas ,
6180     Name-pl = Lemmata ,
6181
6182 type = corollary ,
6183     gender = n ,
6184     case = N ,

```

```

6185     Name-sg = Korollar ,
6186     Name-pl = Korollare ,
6187     case = A ,
6188     Name-sg = Korollar ,
6189     Name-pl = Korollare ,
6190     case = D ,
6191     Name-sg = Korollar ,
6192     Name-pl = Korollaren ,
6193     case = G ,
6194     Name-sg = Korollars ,
6195     Name-pl = Korollare ,
6196
6197 type = proposition ,
6198     gender = m ,
6199     case = N ,
6200     Name-sg = Satz ,
6201     Name-pl = Sätze ,
6202     case = A ,
6203     Name-sg = Satz ,
6204     Name-pl = Sätze ,
6205     case = D ,
6206     Name-sg = Satz ,
6207     Name-pl = Sätzen ,
6208     case = G ,
6209     Name-sg = Satzes ,
6210     Name-pl = Sätze ,
6211
6212 type = definition ,
6213     gender = f ,
6214     case = N ,
6215     Name-sg = Definition ,
6216     Name-pl = Definitionen ,
6217     case = A ,
6218     Name-sg = Definition ,
6219     Name-pl = Definitionen ,
6220     case = D ,
6221     Name-sg = Definition ,
6222     Name-pl = Definitionen ,
6223     case = G ,
6224     Name-sg = Definition ,
6225     Name-pl = Definitionen ,
6226
6227 type = proof ,
6228     gender = m ,
6229     case = N ,
6230     Name-sg = Beweis ,
6231     Name-pl = Beweise ,
6232     case = A ,
6233     Name-sg = Beweis ,
6234     Name-pl = Beweise ,
6235     case = D ,
6236     Name-sg = Beweis ,
6237     Name-pl = Beweisen ,
6238     case = G ,

```

```

6239     Name-sg = Beweises ,
6240     Name-pl = Beweise ,
6241
6242 type = result ,
6243     gender = n ,
6244     case = N ,
6245     Name-sg = Ergebnis ,
6246     Name-pl = Ergebnisse ,
6247     case = A ,
6248     Name-sg = Ergebnis ,
6249     Name-pl = Ergebnisse ,
6250     case = D ,
6251     Name-sg = Ergebnis ,
6252     Name-pl = Ergebnissen ,
6253     case = G ,
6254     Name-sg = Ergebnisses ,
6255     Name-pl = Ergebnisse ,
6256
6257 type = remark ,
6258     gender = f ,
6259     case = N ,
6260     Name-sg = Bemerkung ,
6261     Name-pl = Bemerkungen ,
6262     case = A ,
6263     Name-sg = Bemerkung ,
6264     Name-pl = Bemerkungen ,
6265     case = D ,
6266     Name-sg = Bemerkung ,
6267     Name-pl = Bemerkungen ,
6268     case = G ,
6269     Name-sg = Bemerkung ,
6270     Name-pl = Bemerkungen ,
6271
6272 type = example ,
6273     gender = n ,
6274     case = N ,
6275     Name-sg = Beispiel ,
6276     Name-pl = Beispiele ,
6277     case = A ,
6278     Name-sg = Beispiel ,
6279     Name-pl = Beispiele ,
6280     case = D ,
6281     Name-sg = Beispiel ,
6282     Name-pl = Beispielen ,
6283     case = G ,
6284     Name-sg = Beispiels ,
6285     Name-pl = Beispiele ,
6286
6287 type = algorithm ,
6288     gender = m ,
6289     case = N ,
6290     Name-sg = Algorithmus ,
6291     Name-pl = Algorithmen ,
6292     case = A ,

```

```

6293     Name-sg = Algorithmus ,
6294     Name-pl = Algorithmen ,
6295     case = D ,
6296     Name-sg = Algorithmus ,
6297     Name-pl = Algorithmen ,
6298     case = G ,
6299     Name-sg = Algorithmus ,
6300     Name-pl = Algorithmen ,
6301
6302     type = listing ,
6303     gender = n ,
6304     case = N ,
6305     Name-sg = Listing ,
6306     Name-pl = Listings ,
6307     case = A ,
6308     Name-sg = Listing ,
6309     Name-pl = Listings ,
6310     case = D ,
6311     Name-sg = Listing ,
6312     Name-pl = Listings ,
6313     case = G ,
6314     Name-sg = Listings ,
6315     Name-pl = Listings ,
6316
6317     type = exercise ,
6318     gender = f ,
6319     case = N ,
6320     Name-sg = Übungsaufgabe ,
6321     Name-pl = Übungsaufgaben ,
6322     case = A ,
6323     Name-sg = Übungsaufgabe ,
6324     Name-pl = Übungsaufgaben ,
6325     case = D ,
6326     Name-sg = Übungsaufgabe ,
6327     Name-pl = Übungsaufgaben ,
6328     case = G ,
6329     Name-sg = Übungsaufgabe ,
6330     Name-pl = Übungsaufgaben ,
6331
6332     type = solution ,
6333     gender = f ,
6334     case = N ,
6335     Name-sg = Lösung ,
6336     Name-pl = Lösungen ,
6337     case = A ,
6338     Name-sg = Lösung ,
6339     Name-pl = Lösungen ,
6340     case = D ,
6341     Name-sg = Lösung ,
6342     Name-pl = Lösungen ,
6343     case = G ,
6344     Name-sg = Lösung ,
6345     Name-pl = Lösungen ,
6346 </lang-german>

```


10.4 French

French language file has been initially provided by the author, and has been improved thanks to Denis Bitouzé and François Lagarde (at issue [#1](#)) and participants of the Groupe francophone des Utilisateurs de T_EX (GUTenberg) (at https://groups.google.com/g/gut_fr/c/rNLm6weGcyg) and the fr.comp.text.tex (at <https://groups.google.com/g/fr.comp.text.tex/c/Fa11Tf6MFFs>) mailing lists.

babel-french also has .ldfs for `francais`, `frenchb`, and `canadien`, but they are deprecated as options and, if used, they fall back to either `french` or `acadian`.

```
6347 <*package>
6348 \zcDeclareLanguage [ gender = { f , m } ] { french }
6349 \zcDeclareLanguageAlias { acadian } { french }
6350 </package>
6351 <*lang-french>
6352 namesep = {\nobreakspace} ,
6353 pairsep = {\~et\nobreakspace} ,
6354 listsep = { , ~ } ,
6355 lastsep = {\~et\nobreakspace} ,
6356 tpairsep = {\~et\nobreakspace} ,
6357 tlistsep = { , ~ } ,
6358 tlastsep = {\~et\nobreakspace} ,
6359 notesep = { ~ } ,
6360 rangesep = {\~à\nobreakspace} ,
6361
6362 type = book ,
6363   gender = m ,
6364   Name-sg = Livre ,
6365   name-sg = livre ,
6366   Name-pl = Livres ,
6367   name-pl = livres ,
6368
6369 type = part ,
6370   gender = f ,
6371   Name-sg = Partie ,
6372   name-sg = partie ,
6373   Name-pl = Parties ,
6374   name-pl = parties ,
6375
6376 type = chapter ,
6377   gender = m ,
6378   Name-sg = Chapitre ,
6379   name-sg = chapitre ,
6380   Name-pl = Chapitres ,
6381   name-pl = chapitres ,
6382
6383 type = section ,
6384   gender = f ,
6385   Name-sg = Section ,
6386   name-sg = section ,
6387   Name-pl = Sections ,
6388   name-pl = sections ,
6389
6390 type = paragraph ,
```

```

6391     gender = m ,
6392     Name-sg = Paragraphe ,
6393     name-sg = paragraphe ,
6394     Name-pl = Paragraphes ,
6395     name-pl = paragraphes ,
6396
6397     type = appendix ,
6398     gender = f ,
6399     Name-sg = Annexe ,
6400     name-sg = annexe ,
6401     Name-pl = Annexes ,
6402     name-pl = annexes ,
6403
6404     type = page ,
6405     gender = f ,
6406     Name-sg = Page ,
6407     name-sg = page ,
6408     Name-pl = Pages ,
6409     name-pl = pages ,
6410     rangesep = {-} ,
6411     rangetopair = false ,
6412
6413     type = line ,
6414     gender = f ,
6415     Name-sg = Ligne ,
6416     name-sg = ligne ,
6417     Name-pl = Lignes ,
6418     name-pl = lignes ,
6419
6420     type = figure ,
6421     gender = f ,
6422     Name-sg = Figure ,
6423     name-sg = figure ,
6424     Name-pl = Figures ,
6425     name-pl = figures ,
6426
6427     type = table ,
6428     gender = f ,
6429     Name-sg = Table ,
6430     name-sg = table ,
6431     Name-pl = Tables ,
6432     name-pl = tables ,
6433
6434     type = item ,
6435     gender = m ,
6436     Name-sg = Point ,
6437     name-sg = point ,
6438     Name-pl = Points ,
6439     name-pl = points ,
6440
6441     type = footnote ,
6442     gender = f ,
6443     Name-sg = Note ,
6444     name-sg = note ,

```

```

6445     Name-pl = Notes ,
6446     name-pl = notes ,
6447
6448 type = endnote ,
6449     gender = f ,
6450     Name-sg = Note ,
6451     name-sg = note ,
6452     Name-pl = Notes ,
6453     name-pl = notes ,
6454
6455 type = note ,
6456     gender = f ,
6457     Name-sg = Note ,
6458     name-sg = note ,
6459     Name-pl = Notes ,
6460     name-pl = notes ,
6461
6462 type = equation ,
6463     gender = f ,
6464     Name-sg = Équation ,
6465     name-sg = équation ,
6466     Name-pl = Équations ,
6467     name-pl = équations ,
6468     refbounds-first-sg = {(,),(,)} ,
6469     refbounds = {(,),(,)} ,
6470
6471 type = theorem ,
6472     gender = m ,
6473     Name-sg = Théorème ,
6474     name-sg = théorème ,
6475     Name-pl = Théorèmes ,
6476     name-pl = théorèmes ,
6477
6478 type = lemma ,
6479     gender = m ,
6480     Name-sg = Lemme ,
6481     name-sg = lemme ,
6482     Name-pl = Lemmes ,
6483     name-pl = lemmes ,
6484
6485 type = corollary ,
6486     gender = m ,
6487     Name-sg = Corollaire ,
6488     name-sg = corollaire ,
6489     Name-pl = Corollaires ,
6490     name-pl = corollaires ,
6491
6492 type = proposition ,
6493     gender = f ,
6494     Name-sg = Proposition ,
6495     name-sg = proposition ,
6496     Name-pl = Propositions ,
6497     name-pl = propositions ,
6498

```

```

6499 type = definition ,
6500     gender = f ,
6501     Name-sg = Définition ,
6502     name-sg = définition ,
6503     Name-pl = Définitions ,
6504     name-pl = définitions ,
6505
6506 type = proof ,
6507     gender = f ,
6508     Name-sg = Démonstration ,
6509     name-sg = démonstration ,
6510     Name-pl = Démonstrations ,
6511     name-pl = démonstrations ,
6512
6513 type = result ,
6514     gender = m ,
6515     Name-sg = Résultat ,
6516     name-sg = résultat ,
6517     Name-pl = Résultats ,
6518     name-pl = résultats ,
6519
6520 type = remark ,
6521     gender = f ,
6522     Name-sg = Remarque ,
6523     name-sg = remarque ,
6524     Name-pl = Remarques ,
6525     name-pl = remarques ,
6526
6527 type = example ,
6528     gender = m ,
6529     Name-sg = Exemple ,
6530     name-sg = exemple ,
6531     Name-pl = Exemples ,
6532     name-pl = exemples ,
6533
6534 type = algorithm ,
6535     gender = m ,
6536     Name-sg = Algorithme ,
6537     name-sg = algorithme ,
6538     Name-pl = Algorithmes ,
6539     name-pl = algorithmes ,
6540
6541 type = listing ,
6542     gender = m ,
6543     Name-sg = Listing ,
6544     name-sg = listing ,
6545     Name-pl = Listings ,
6546     name-pl = listings ,
6547
6548 type = exercise ,
6549     gender = m ,
6550     Name-sg = Exercice ,
6551     name-sg = exercice ,
6552     Name-pl = Exercices ,

```

```

6553   name-pl = exercices ,
6554
6555   type = solution ,
6556   gender = f ,
6557   Name-sg = Solution ,
6558   name-sg = solution ,
6559   Name-pl = Solutions ,
6560   name-pl = solutions ,
6561 </lang-french>

```

10.5 Portuguese

Portuguese language file provided by the author, who's a native speaker of (Brazilian) Portuguese. I do expect this to be sufficiently general, but if Portuguese speakers from other places feel the need for a Portuguese variant, please let me know.

```

6562 <*package>
6563 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6564 \zcDeclareLanguageAlias { brazilian } { portuguese }
6565 \zcDeclareLanguageAlias { brazil } { portuguese }
6566 \zcDeclareLanguageAlias { portuges } { portuguese }
6567 </package>
6568 <*lang-portuguese>
6569 namesep = {\nobreakspace} ,
6570 pairsep = {\nobreakspace} ,
6571 listsep = {,~} ,
6572 lastsep = {\nobreakspace} ,
6573 tpairsep = {\nobreakspace} ,
6574 tlistsep = {,~} ,
6575 tlastsep = {\nobreakspace} ,
6576 notesep = {~} ,
6577 rangesep = {\nobreakspace} ,
6578
6579 type = book ,
6580 gender = m ,
6581 Name-sg = Livro ,
6582 name-sg = livro ,
6583 Name-pl = Livros ,
6584 name-pl = livros ,
6585
6586 type = part ,
6587 gender = f ,
6588 Name-sg = Parte ,
6589 name-sg = parte ,
6590 Name-pl = Partes ,
6591 name-pl = partes ,
6592
6593 type = chapter ,
6594 gender = m ,
6595 Name-sg = Capítulo ,
6596 name-sg = capítulo ,
6597 Name-pl = Capítulos ,
6598 name-pl = capítulos ,

```

```

6599
6600 type = section ,
6601     gender = f ,
6602     Name-sg = Seção ,
6603     name-sg = seção ,
6604     Name-pl = Seções ,
6605     name-pl = seções ,
6606
6607 type = paragraph ,
6608     gender = m ,
6609     Name-sg = Parágrafo ,
6610     name-sg = parágrafo ,
6611     Name-pl = Parágrafos ,
6612     name-pl = parágrafos ,
6613     Name-sg-ab = Par. ,
6614     name-sg-ab = par. ,
6615     Name-pl-ab = Par. ,
6616     name-pl-ab = par. ,
6617
6618 type = appendix ,
6619     gender = m ,
6620     Name-sg = Apêndice ,
6621     name-sg = apêndice ,
6622     Name-pl = Apêndices ,
6623     name-pl = apêndices ,
6624
6625 type = page ,
6626     gender = f ,
6627     Name-sg = Página ,
6628     name-sg = página ,
6629     Name-pl = Páginas ,
6630     name-pl = páginas ,
6631     rangesep = {\textendash} ,
6632     rangetopair = false ,
6633
6634 type = line ,
6635     gender = f ,
6636     Name-sg = Linha ,
6637     name-sg = linha ,
6638     Name-pl = Linhas ,
6639     name-pl = linhas ,
6640
6641 type = figure ,
6642     gender = f ,
6643     Name-sg = Figura ,
6644     name-sg = figura ,
6645     Name-pl = Figuras ,
6646     name-pl = figuras ,
6647     Name-sg-ab = Fig. ,
6648     name-sg-ab = fig. ,
6649     Name-pl-ab = Figs. ,
6650     name-pl-ab = figs. ,
6651
6652 type = table ,

```

```

6653 gender = f ,
6654 Name-sg = Tabela ,
6655 name-sg = tabela ,
6656 Name-pl = Tabelas ,
6657 name-pl = tabelas ,
6658
6659 type = item ,
6660 gender = m ,
6661 Name-sg = Item ,
6662 name-sg = item ,
6663 Name-pl = Itens ,
6664 name-pl = itens ,
6665
6666 type = footnote ,
6667 gender = f ,
6668 Name-sg = Nota ,
6669 name-sg = nota ,
6670 Name-pl = Notas ,
6671 name-pl = notas ,
6672
6673 type = endnote ,
6674 gender = f ,
6675 Name-sg = Nota ,
6676 name-sg = nota ,
6677 Name-pl = Notas ,
6678 name-pl = notas ,
6679
6680 type = note ,
6681 gender = f ,
6682 Name-sg = Nota ,
6683 name-sg = nota ,
6684 Name-pl = Notas ,
6685 name-pl = notas ,
6686
6687 type = equation ,
6688 gender = f ,
6689 Name-sg = Equação ,
6690 name-sg = equação ,
6691 Name-pl = Equações ,
6692 name-pl = equações ,
6693 Name-sg-ab = Eq. ,
6694 name-sg-ab = eq. ,
6695 Name-pl-ab = Eqs. ,
6696 name-pl-ab = eqs. ,
6697 refbounds-first-sg = {,(,),} ,
6698 refbounds = {(,,)} ,
6699
6700 type = theorem ,
6701 gender = m ,
6702 Name-sg = Teorema ,
6703 name-sg = teorema ,
6704 Name-pl = Teoremas ,
6705 name-pl = teoremas ,
6706

```

```

6707 type = lemma ,
6708     gender = m ,
6709     Name-sg = Lema ,
6710     name-sg = lema ,
6711     Name-pl = Lemas ,
6712     name-pl = lemas ,
6713
6714 type = corollary ,
6715     gender = m ,
6716     Name-sg = Corolário ,
6717     name-sg = corolário ,
6718     Name-pl = Corolários ,
6719     name-pl = corolários ,
6720
6721 type = proposition ,
6722     gender = f ,
6723     Name-sg = Proposição ,
6724     name-sg = proposição ,
6725     Name-pl = Proposições ,
6726     name-pl = proposições ,
6727
6728 type = definition ,
6729     gender = f ,
6730     Name-sg = Definição ,
6731     name-sg = definição ,
6732     Name-pl = Definições ,
6733     name-pl = definições ,
6734
6735 type = proof ,
6736     gender = f ,
6737     Name-sg = Demonstração ,
6738     name-sg = demonstração ,
6739     Name-pl = Demonstrações ,
6740     name-pl = demonstrações ,
6741
6742 type = result ,
6743     gender = m ,
6744     Name-sg = Resultado ,
6745     name-sg = resultado ,
6746     Name-pl = Resultados ,
6747     name-pl = resultados ,
6748
6749 type = remark ,
6750     gender = f ,
6751     Name-sg = Observação ,
6752     name-sg = observação ,
6753     Name-pl = Observações ,
6754     name-pl = observações ,
6755
6756 type = example ,
6757     gender = m ,
6758     Name-sg = Exemplo ,
6759     name-sg = exemplo ,
6760     Name-pl = Exemplos ,

```



```

6761   name-pl = exemplos ,
6762
6763   type = algorithm ,
6764   gender = m ,
6765   Name-sg = Algoritmo ,
6766   name-sg = algoritmo ,
6767   Name-pl = Algoritmos ,
6768   name-pl = algoritmos ,
6769
6770   type = listing ,
6771   gender = f ,
6772   Name-sg = Listagem ,
6773   name-sg = listagem ,
6774   Name-pl = Listagens ,
6775   name-pl = listagens ,
6776
6777   type = exercise ,
6778   gender = m ,
6779   Name-sg = Exercício ,
6780   name-sg = exercício ,
6781   Name-pl = Exercícios ,
6782   name-pl = exercícios ,
6783
6784   type = solution ,
6785   gender = f ,
6786   Name-sg = Solução ,
6787   name-sg = solução ,
6788   Name-pl = Soluções ,
6789   name-pl = soluções ,
6790 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```

6791 <*package>
6792 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6793 </package>
6794 <*lang-spanish>
6795 namesep = {\nobreakspace} ,
6796 pairsep = {\~y\nobreakspace} ,
6797 listsep = {,~} ,
6798 lastsep = {\~y\nobreakspace} ,
6799 tpairsep = {\~y\nobreakspace} ,
6800 tlistsep = {,~} ,
6801 tlastsep = {\~y\nobreakspace} ,
6802 notesep = {\~} ,
6803 rangesep = {\~a\nobreakspace} ,
6804
6805 type = book ,
6806 gender = m ,
6807 Name-sg = Libro ,
6808 name-sg = libro ,

```

```

6809     Name-pl = Libros ,
6810     name-pl = libros ,
6811
6812     type = part ,
6813     gender = f ,
6814     Name-sg = Parte ,
6815     name-sg = parte ,
6816     Name-pl = Partes ,
6817     name-pl = partes ,
6818
6819     type = chapter ,
6820     gender = m ,
6821     Name-sg = Capítulo ,
6822     name-sg = capítulo ,
6823     Name-pl = Capítulos ,
6824     name-pl = capítulos ,
6825
6826     type = section ,
6827     gender = f ,
6828     Name-sg = Sección ,
6829     name-sg = sección ,
6830     Name-pl = Secciones ,
6831     name-pl = secciones ,
6832
6833     type = paragraph ,
6834     gender = m ,
6835     Name-sg = Párrafo ,
6836     name-sg = párrafo ,
6837     Name-pl = Párrafos ,
6838     name-pl = párrafos ,
6839
6840     type = appendix ,
6841     gender = m ,
6842     Name-sg = Apéndice ,
6843     name-sg = apéndice ,
6844     Name-pl = Apéndices ,
6845     name-pl = apéndices ,
6846
6847     type = page ,
6848     gender = f ,
6849     Name-sg = Página ,
6850     name-sg = página ,
6851     Name-pl = Páginas ,
6852     name-pl = páginas ,
6853     rangesep = {\textendash} ,
6854     rangetopair = false ,
6855
6856     type = line ,
6857     gender = f ,
6858     Name-sg = Línea ,
6859     name-sg = línea ,
6860     Name-pl = Líneas ,
6861     name-pl = líneas ,
6862

```

```

6863 type = figure ,
6864     gender = f ,
6865     Name-sg = Figura ,
6866     name-sg = figura ,
6867     Name-pl = Figuras ,
6868     name-pl = figuras ,
6869
6870 type = table ,
6871     gender = m ,
6872     Name-sg = Cuadro ,
6873     name-sg = cuadro ,
6874     Name-pl = Cuadros ,
6875     name-pl = cuadros ,
6876
6877 type = item ,
6878     gender = m ,
6879     Name-sg = Punto ,
6880     name-sg = punto ,
6881     Name-pl = Puntos ,
6882     name-pl = puntos ,
6883
6884 type = footnote ,
6885     gender = f ,
6886     Name-sg = Nota ,
6887     name-sg = nota ,
6888     Name-pl = Notas ,
6889     name-pl = notas ,
6890
6891 type = endnote ,
6892     gender = f ,
6893     Name-sg = Nota ,
6894     name-sg = nota ,
6895     Name-pl = Notas ,
6896     name-pl = notas ,
6897
6898 type = note ,
6899     gender = f ,
6900     Name-sg = Nota ,
6901     name-sg = nota ,
6902     Name-pl = Notas ,
6903     name-pl = notas ,
6904
6905 type = equation ,
6906     gender = f ,
6907     Name-sg = Ecuación ,
6908     name-sg = ecuación ,
6909     Name-pl = Ecuaciones ,
6910     name-pl = ecuaciones ,
6911     rebounds-first-sg = {(,),(,)} ,
6912     rebounds = {(,),(,)} ,
6913
6914 type = theorem ,
6915     gender = m ,
6916     Name-sg = Teorema ,

```

```

6917     name-sg = teorema ,
6918     Name-pl = Teoremas ,
6919     name-pl = teoremas ,
6920
6921     type = lemma ,
6922     gender = m ,
6923     Name-sg = Lema ,
6924     name-sg = lema ,
6925     Name-pl = Lemas ,
6926     name-pl = lemas ,
6927
6928     type = corollary ,
6929     gender = m ,
6930     Name-sg = Corolario ,
6931     name-sg = corolario ,
6932     Name-pl = Corolarios ,
6933     name-pl = corolarios ,
6934
6935     type = proposition ,
6936     gender = f ,
6937     Name-sg = Proposición ,
6938     name-sg = proposición ,
6939     Name-pl = Proposiciones ,
6940     name-pl = proposiciones ,
6941
6942     type = definition ,
6943     gender = f ,
6944     Name-sg = Definición ,
6945     name-sg = definición ,
6946     Name-pl = Definiciones ,
6947     name-pl = definiciones ,
6948
6949     type = proof ,
6950     gender = f ,
6951     Name-sg = Demostración ,
6952     name-sg = demostración ,
6953     Name-pl = Demostraciones ,
6954     name-pl = demostraciones ,
6955
6956     type = result ,
6957     gender = m ,
6958     Name-sg = Resultado ,
6959     name-sg = resultado ,
6960     Name-pl = Resultados ,
6961     name-pl = resultados ,
6962
6963     type = remark ,
6964     gender = f ,
6965     Name-sg = Observación ,
6966     name-sg = observación ,
6967     Name-pl = Observaciones ,
6968     name-pl = observaciones ,
6969
6970     type = example ,

```

```

6971   gender = m ,
6972   Name-sg = Ejemplo ,
6973   name-sg = ejemplo ,
6974   Name-pl = Ejemplos ,
6975   name-pl = ejemplos ,
6976
6977   type = algorithm ,
6978   gender = m ,
6979   Name-sg = Algoritmo ,
6980   name-sg = algoritmo ,
6981   Name-pl = Algoritmos ,
6982   name-pl = algoritmos ,
6983
6984   type = listing ,
6985   gender = m ,
6986   Name-sg = Listado ,
6987   name-sg = listado ,
6988   Name-pl = Listados ,
6989   name-pl = listados ,
6990
6991   type = exercise ,
6992   gender = m ,
6993   Name-sg = Ejercicio ,
6994   name-sg = ejercicio ,
6995   Name-pl = Ejercicios ,
6996   name-pl = ejercicios ,
6997
6998   type = solution ,
6999   gender = f ,
7000   Name-sg = Solución ,
7001   name-sg = solución ,
7002   Name-pl = Soluciones ,
7003   name-pl = soluciones ,
7004 </lang-spanish>

```

10.7 Dutch

Dutch language file initially contributed by ‘niluxv’ (PR [#5](#)). All genders were checked against the “Dikke Van Dale”. Many words have multiple genders.

```

7005 <*package>
7006 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7007 </package>
7008 <*lang-dutch>
7009 namesep = {\nobreakspace} ,
7010 pairsep = {\sim\nobreakspace} ,
7011 listsep = {,~} ,
7012 lastsep = {\sim\nobreakspace} ,
7013 tpairsep = {\sim\nobreakspace} ,
7014 tlistsep = {,~} ,
7015 tlastsep = {,~\sim\nobreakspace} ,
7016 notesep = {\sim} ,
7017 rangesep = {\sim/m\nobreakspace} ,

```

```

7018
7019 type = book ,
7020     gender = n ,
7021     Name-sg = Boek ,
7022     name-sg = boek ,
7023     Name-pl = Boeken ,
7024     name-pl = boeken ,
7025
7026 type = part ,
7027     gender = n ,
7028     Name-sg = Deel ,
7029     name-sg = deel ,
7030     Name-pl = Delen ,
7031     name-pl = delen ,
7032
7033 type = chapter ,
7034     gender = n ,
7035     Name-sg = Hoofdstuk ,
7036     name-sg = hoofdstuk ,
7037     Name-pl = Hoofdstukken ,
7038     name-pl = hoofdstukken ,
7039
7040 type = section ,
7041     gender = m ,
7042     Name-sg = Paragraaf ,
7043     name-sg = paragraaf ,
7044     Name-pl = Paragrafen ,
7045     name-pl = paragrafen ,
7046
7047 type = paragraph ,
7048     gender = f ,
7049     Name-sg = Alinea ,
7050     name-sg = alinea ,
7051     Name-pl = Alinea's ,
7052     name-pl = alinea's ,
7053

```

2022-12-27, ‘niluxv’: “bijlage” is chosen over “appendix” (plural “appendices”, gender: m, n) for consistency with babel/polyglossia. “bijlages” is also a valid plural; “bijlagen” is chosen for consistency with babel/polyglossia.

```

7054 type = appendix ,
7055     gender = { f , m } ,
7056     Name-sg = Blage ,
7057     name-sg = blage ,
7058     Name-pl = Blagen ,
7059     name-pl = blagen ,
7060
7061 type = page ,
7062     gender = { f , m } ,
7063     Name-sg = Pagina ,
7064     name-sg = pagina ,
7065     Name-pl = Pagina's ,
7066     name-pl = pagina's ,
7067     rangesep = {\textendash} ,

```

```

7068     rangetopair = false ,
7069
7070 type = line ,
7071     gender = m ,
7072     Name-sg = Regel ,
7073     name-sg = regel ,
7074     Name-pl = Regels ,
7075     name-pl = regels ,
7076
7077 type = figure ,
7078     gender = { n , f , m } ,
7079     Name-sg = Figuur ,
7080     name-sg = figuur ,
7081     Name-pl = Figuren ,
7082     name-pl = figuren ,
7083
7084 type = table ,
7085     gender = { f , m } ,
7086     Name-sg = Tabel ,
7087     name-sg = tabel ,
7088     Name-pl = Tabellen ,
7089     name-pl = tabellen ,
7090
7091 type = item ,
7092     gender = n ,
7093     Name-sg = Punt ,
7094     name-sg = punt ,
7095     Name-pl = Punten ,
7096     name-pl = punten ,
7097
7098 type = footnote ,
7099     gender = { f , m } ,
7100     Name-sg = Voetnoot ,
7101     name-sg = voetnoot ,
7102     Name-pl = Voetnoten ,
7103     name-pl = voetnoten ,
7104
7105 type = endnote ,
7106     gender = { f , m } ,
7107     Name-sg = Eindnoot ,
7108     name-sg = eindnoot ,
7109     Name-pl = Eindnoten ,
7110     name-pl = eindnoten ,
7111
7112 type = note ,
7113     gender = f ,
7114     Name-sg = Opmerking ,
7115     name-sg = opmerking ,
7116     Name-pl = Opmerkingen ,
7117     name-pl = opmerkingen ,
7118
7119 type = equation ,
7120     gender = f ,
7121     Name-sg = Vergelking ,

```

```

7122 name-sg = vergelking ,
7123 Name-pl = Vergelkingen ,
7124 name-pl = vergelkingen ,
7125 Name-sg-ab = Vgl. ,
7126 name-sg-ab = vgl. ,
7127 Name-pl-ab = Vgl.'s ,
7128 name-pl-ab = vgl.'s ,
7129 refbounds-first-sg = {,(,),} ,
7130 refbounds = {(,,),} ,
7131
7132 type = theorem ,
7133 gender = f ,
7134 Name-sg = Stelling ,
7135 name-sg = stelling ,
7136 Name-pl = Stellingen ,
7137 name-pl = stellingen ,
7138

```

2022-01-09, 'niluxv': An alternative plural is “lemmata”. That is also a correct English plural for lemma, but the English language file chooses “lemmas”. For consistency we therefore choose “lemma’s”.

```

7139 type = lemma ,
7140 gender = n ,
7141 Name-sg = Lemma ,
7142 name-sg = lemma ,
7143 Name-pl = Lemma's ,
7144 name-pl = lemma's ,
7145
7146 type = corollary ,
7147 gender = n ,
7148 Name-sg = Gevolg ,
7149 name-sg = gevolg ,
7150 Name-pl = Gevolgen ,
7151 name-pl = gevolgen ,
7152
7153 type = proposition ,
7154 gender = f ,
7155 Name-sg = Propositie ,
7156 name-sg = propositie ,
7157 Name-pl = Propositions ,
7158 name-pl = propositions ,
7159
7160 type = definition ,
7161 gender = f ,
7162 Name-sg = Definitie ,
7163 name-sg = definitie ,
7164 Name-pl = Definities ,
7165 name-pl = definities ,
7166
7167 type = proof ,
7168 gender = n ,
7169 Name-sg = Bews ,
7170 name-sg = bews ,
7171 Name-pl = Bewzen ,

```



```

7172     name-pl = bewzen ,
7173
7174     type = result ,
7175     gender = n ,
7176     Name-sg = Resultaat ,
7177     name-sg = resultaat ,
7178     Name-pl = Resultaten ,
7179     name-pl = resultaten ,
7180
7181     type = remark ,
7182     gender = f ,
7183     Name-sg = Opmerking ,
7184     name-sg = opmerking ,
7185     Name-pl = Opmerkingen ,
7186     name-pl = opmerkingen ,
7187
7188     type = example ,
7189     gender = n ,
7190     Name-sg = Voorbeeld ,
7191     name-sg = voorbeeld ,
7192     Name-pl = Voorbeelden ,
7193     name-pl = voorbeelden ,
7194

```

2022-12-27, ‘niluxv’: “algoritmes” is also a valid plural. “algoritmen” is chosen to be consistent with using “bijlagen” (and not “bijlages”) as the plural of “bijlage”.

```

7195     type = algorithm ,
7196     gender = { n , f , m } ,
7197     Name-sg = Algoritme ,
7198     name-sg = algoritme ,
7199     Name-pl = Algoritmen ,
7200     name-pl = algoritmen ,
7201

```

2022-01-09, ‘niluxv’: EN-NL Van Dale translates listing as (3) “uitdraai van computer-programma”, “listing”.

```

7202     type = listing ,
7203     gender = m ,
7204     Name-sg = Listing ,
7205     name-sg = listing ,
7206     Name-pl = Listings ,
7207     name-pl = listings ,
7208
7209     type = exercise ,
7210     gender = { f , m } ,
7211     Name-sg = Opgave ,
7212     name-sg = opgave ,
7213     Name-pl = Opgaven ,
7214     name-pl = opgaven ,
7215
7216     type = solution ,
7217     gender = f ,
7218     Name-sg = Oplossing ,
7219     name-sg = oplossing ,
7220     Name-pl = Oplossingen ,

```

```

7221   name-pl = oplossingen ,
7222 </lang-dutch>

```

10.8 Italian

Italian language file initially contributed by Matteo Ferrigato (issue [#11](#)), with the help of participants of the Gruppo Utilizzatori Italiani di T_EX (GuIT) forum (at <https://www.guitex.org/home/it/forum/5-tex-e-latex/121856-closed-zref-clever-e-localizzazione-in->

```

7223 <*package>
7224 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7225 </package>
7226 <*lang-italian>
7227 namesep   = {\nobreakspace} ,
7228 pairsep   = {\~e\nobreakspace} ,
7229 listsep    = {,~} ,
7230 lastsep    = {\~e\nobreakspace} ,
7231 tpairsep   = {\~e\nobreakspace} ,
7232 tlistsep   = {,~} ,
7233 tlastsep   = {,~e\nobreakspace} ,
7234 notesep    = {~} ,
7235 rangesep   = {\~a\nobreakspace} ,
7236 +refbounds-rb = {da\nobreakspace,,,} ,
7237
7238 type = book ,
7239   gender = m ,
7240   Name-sg = Libro ,
7241   name-sg = libro ,
7242   Name-pl = Libri ,
7243   name-pl = libri ,
7244
7245 type = part ,
7246   gender = f ,
7247   Name-sg = Parte ,
7248   name-sg = parte ,
7249   Name-pl = Parti ,
7250   name-pl = parti ,
7251
7252 type = chapter ,
7253   gender = m ,
7254   Name-sg = Capitolo ,
7255   name-sg = capitolo ,
7256   Name-pl = Capitoli ,
7257   name-pl = capitoli ,
7258
7259 type = section ,
7260   gender = m ,
7261   Name-sg = Paragrafo ,
7262   name-sg = paragrafo ,
7263   Name-pl = Paragrafi ,
7264   name-pl = paragrafi ,
7265
7266 type = paragraph ,

```

```

7267     gender = m ,
7268     Name-sg = Capoverso ,
7269     name-sg = capoverso ,
7270     Name-pl = Capoversi ,
7271     name-pl = capoversi ,
7272
7273     type = appendix ,
7274     gender = f ,
7275     Name-sg = Appendice ,
7276     name-sg = appendice ,
7277     Name-pl = Appendici ,
7278     name-pl = appendici ,
7279
7280     type = page ,
7281     gender = f ,
7282     Name-sg = Pagina ,
7283     name-sg = pagina ,
7284     Name-pl = Pagine ,
7285     name-pl = pagine ,
7286     Name-sg-ab = Pag. ,
7287     name-sg-ab = pag. ,
7288     Name-pl-ab = Pag. ,
7289     name-pl-ab = pag. ,
7290     rangesep = {\textendash} ,
7291     rangetopair = false ,
7292     +refbounds-rb = {,,} ,
7293
7294     type = line ,
7295     gender = f ,
7296     Name-sg = Riga ,
7297     name-sg = riga ,
7298     Name-pl = Righe ,
7299     name-pl = righe ,
7300
7301     type = figure ,
7302     gender = f ,
7303     Name-sg = Figura ,
7304     name-sg = figura ,
7305     Name-pl = Figure ,
7306     name-pl = figure ,
7307     Name-sg-ab = Fig. ,
7308     name-sg-ab = fig. ,
7309     Name-pl-ab = Fig. ,
7310     name-pl-ab = fig. ,
7311
7312     type = table ,
7313     gender = f ,
7314     Name-sg = Tabella ,
7315     name-sg = tabella ,
7316     Name-pl = Tabelle ,
7317     name-pl = tabelle ,
7318     Name-sg-ab = Tab. ,
7319     name-sg-ab = tab. ,
7320     Name-pl-ab = Tab. ,

```

```

7321     name-pl-ab = tab. ,
7322
7323 type = item ,
7324     gender = m ,
7325     Name-sg = Punto ,
7326     name-sg = punto ,
7327     Name-pl = Punti ,
7328     name-pl = punti ,
7329
7330 type = footnote ,
7331     gender = f ,
7332     Name-sg = Nota ,
7333     name-sg = nota ,
7334     Name-pl = Note ,
7335     name-pl = note ,
7336
7337 type = endnote ,
7338     gender = f ,
7339     Name-sg = Nota ,
7340     name-sg = nota ,
7341     Name-pl = Note ,
7342     name-pl = note ,
7343
7344 type = note ,
7345     gender = f ,
7346     Name-sg = Nota ,
7347     name-sg = nota ,
7348     Name-pl = Note ,
7349     name-pl = note ,
7350
7351 type = equation ,
7352     gender = f ,
7353     Name-sg = Equazione ,
7354     name-sg = equazione ,
7355     Name-pl = Equazioni ,
7356     name-pl = equazioni ,
7357     Name-sg-ab = Eq. ,
7358     name-sg-ab = eq. ,
7359     Name-pl-ab = Eq. ,
7360     name-pl-ab = eq. ,
7361     +refbounds-rb = {da\nobreakspace(,,)} ,
7362     refbounds-first-sg = {(,),(,)} ,
7363     refbounds = {(,,)} ,
7364
7365 type = theorem ,
7366     gender = m ,
7367     Name-sg = Teorema ,
7368     name-sg = teorema ,
7369     Name-pl = Teoremi ,
7370     name-pl = teoremi ,
7371
7372 type = lemma ,
7373     gender = m ,
7374     Name-sg = Lemma ,

```

```

7375     name-sg = lemma ,
7376     Name-pl = Lemmi ,
7377     name-pl = lemmi ,
7378
7379 type = corollary ,
7380     gender = m ,
7381     Name-sg = Corollario ,
7382     name-sg = corollario ,
7383     Name-pl = Corollari ,
7384     name-pl = corollari ,
7385
7386 type = proposition ,
7387     gender = f ,
7388     Name-sg = Proposizione ,
7389     name-sg = proposizione ,
7390     Name-pl = Proposizioni ,
7391     name-pl = proposizioni ,
7392
7393 type = definition ,
7394     gender = f ,
7395     Name-sg = Definizione ,
7396     name-sg = definizione ,
7397     Name-pl = Definizioni ,
7398     name-pl = definizioni ,
7399
7400 type = proof ,
7401     gender = f ,
7402     Name-sg = Dimostrazione ,
7403     name-sg = dimostrazione ,
7404     Name-pl = Dimostrazioni ,
7405     name-pl = dimostrazioni ,
7406
7407 type = result ,
7408     gender = m ,
7409     Name-sg = Risultato ,
7410     name-sg = risultato ,
7411     Name-pl = Risultati ,
7412     name-pl = risultati ,
7413
7414 type = remark ,
7415     gender = f ,
7416     Name-sg = Osservazione ,
7417     name-sg = osservazione ,
7418     Name-pl = Osservazioni ,
7419     name-pl = osservazioni ,
7420
7421 type = example ,
7422     gender = m ,
7423     Name-sg = Esempio ,
7424     name-sg = esempio ,
7425     Name-pl = Esempi ,
7426     name-pl = esempi ,
7427
7428 type = algorithm ,

```

```

7429   gender = m ,
7430   Name-sg = Algoritmo ,
7431   name-sg = algoritmo ,
7432   Name-pl = Algoritmi ,
7433   name-pl = algoritmi ,
7434
7435   type = listing ,
7436   gender = m ,
7437   Name-sg = Listato ,
7438   name-sg = listato ,
7439   Name-pl = Listati ,
7440   name-pl = listati ,
7441
7442   type = exercise ,
7443   gender = m ,
7444   Name-sg = Esercizio ,
7445   name-sg = esercizio ,
7446   Name-pl = Esercizi ,
7447   name-pl = esercizi ,
7448
7449   type = solution ,
7450   gender = f ,
7451   Name-sg = Soluzione ,
7452   name-sg = soluzione ,
7453   Name-pl = Soluzioni ,
7454   name-pl = soluzioni ,
7455 </lang-italian>

```

Index

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

A		bool commands:
\A	1878	\bool_case_true: 2
\AddToHook	102, 2024, 2068, 2091, 2122, 2124, 2162, 2235, 2277, 2428, 2441, 2449, 5308, 5344, 5361, 5363, 5368, 5413, 5415, 5417, 5419, 5421, 5423, 5425, 5427, 5431, 5435, 5437, 5442, 5452, 5468, 5479, 5513, 5564, 5589	\bool_gset_false:N 534
\alph	5494, 5580	\bool_gset_true:N 527, 2415
\appendix	2, 126, 129, 138	\bool_if:NTF 378, 455, 493, 551, 1754, 1805, 2028, 2032, 2451, 3425, 3828, 3969, 4096, 4132, 4166, 4233, 4246, 4258, 4309, 4326, 4336, 4341, 4387, 4391, 4447, 4472, 4479, 4485, 4496, 4502, 4530, 4575, 4597, 4626, 4775, 4928, 4930, 5516, 5592
\appendixname	126, 140	\bool_if:nTF 71, 3539, 3549, 3573, 3590, 3605, 3670, 3678, 4319, 4696, 4737, 4818, 4835
\AtEndOfPackage	2439	\bool_if_exist:NTF 332, 342, 366, 376, 426, 443, 453, 478, 481, 489, 491, 505, 508, 515, 518, 525, 532
B		\bool_lazy_all:nTF 4415, 4940
\babelname	2078	\bool_lazy_and:nnTF 2572, 3396,
\babelprovide	29, 54	
\bicaption	128	
\bionenumcaption	128	
\bitwonumcaption	128	

3417, 4200, 4271, 4649, 4913, 4955	cs commands:
\bool_lazy_any:nTF 5092, 5101	\cs_generate_variant:Nn 68,
\bool_lazy_or:nnTF 1371, 1404, 1954,	275, 281, 288, 299, 310, 321, 336,
2521, 2785, 3284, 3309, 3400, 4901	346, 353, 360, 371, 395, 405, 430,
\bool_new:N 333, 343, 368, 427,	437, 448, 486, 512, 522, 529, 536,
445, 483, 506, 509, 516, 519, 526,	964, 1838, 1874, 1925, 1971, 2580,
533, 793, 1568, 1569, 1596, 1620,	4731, 4769, 5135, 5239, 5272, 5305
1972, 1979, 1986, 1999, 2000, 2170,	\cs_if_exist:N 24, 27, 49, 52, 61, 81, 5332, 5338
2171, 2172, 2173, 2174, 2270, 2271,	\cs_if_exist_p:N 4202, 4273, 4651, 5626
2408, 3433, 3448, 3710, 3711, 3722,	\cs_if_exist_use:N 5430, 5664
3723, 3730, 3732, 3733, 3746, 3747,	\cs_new:Npn 59, 69, 79, 90,
3748, 3760, 3761, 5478, 5523, 5563	276, 282, 284, 286, 289, 300, 311,
\bool_set:Nn 3393	323, 325, 713, 4692, 4732, 4770, 5125
\bool_set_eq:NN 541	\cs_new_eq:NN 5470, 5475
\bool_set_false:N 367, 444, 480,	\cs_new_protected:Npn
482, 517, 1581, 1585, 1757, 1855, 270, 327, 337, 347,
1900, 1942, 2007, 2016, 2017, 2034,	354, 361, 386, 396, 417, 419, 421,
2041, 2182, 2186, 2193, 2201, 2202,	431, 438, 476, 503, 513, 523, 530,
2203, 2295, 2307, 3438, 3531, 3777,	806, 908, 1517, 1536, 1740, 1839,
3778, 3795, 3834, 3845, 4245, 4437,	1884, 1926, 2447, 2578, 3388, 3452,
4438, 4445, 4446, 4679, 5099, 5116	3494, 3505, 3518, 3648, 3700, 3762,
\bool_set_true:N	3976, 4441, 4688, 4690, 4884, 5119,
. 334, 344, 428, 507, 510,	5136, 5207, 5240, 5273, 5403, 5531
520, 1575, 1576, 1580, 1586, 1779,	\cs_set:Npn 5406
1801, 1864, 1866, 1904, 1906, 1915,	\cs_set_eq:NN
1917, 1946, 1948, 1961, 1963, 2006, 715, 5405, 5411, 5471, 5476
2011, 2012, 2180, 2187, 2192, 2209,	\cs_set_nopar:Npn 5461
2211, 2213, 2216, 2217, 2218, 2283,	\cs_to_str:N 326
2288, 3545, 3555, 3559, 3581, 3596,	
3611, 3635, 3803, 3829, 3835, 3839,	
3846, 3992, 4003, 4014, 4064, 4100,	
4136, 4170, 4187, 4268, 4302, 4474,	
4534, 4580, 4602, 4631, 4889, 4896,	
5018, 5077, 5115, 5152, 5159, 5160,	
5178, 5195, 5197, 5492, 5530, 5577	
\bool_until_do:Nn	
. 1856, 1907, 1949, 3571, 3796	
\bool_while_do:nn 5624	
\l_tmpa_bool 1757, 1779,	
1801, 1805, 1855, 1856, 1864, 1866,	
1900, 1904, 1906, 1907, 1915, 1917,	
1942, 1946, 1948, 1949, 1961, 1963	
C	
\chaptername 140	
clist commands:	
\clist_map_inline:nn	
. 626, 674, 691, 1010,	
2204, 2356, 2417, 2921, 5495, 5582	
\contsubbottom 129	
\contsubcaption 129	
\counterwithin 5	
\crefstripprefix 45	
	D
	\d 1878
	E
	\endinput 12
	\endsidecaption 128
	exp commands:
	\exp_args:NNe 36, 39
	\exp_args:NNNo 272
	\exp_args:NNNV 1819, 1868, 1919, 1965
	\exp_args:NNo 272, 278
	\exp_args:No 278
	\exp_args:Nx 918, 5536, 5544
	\exp_args:Nxx
	1758, 1770, 1782, 1792, 1858, 1909,
	1951, 5138, 5144, 5166, 5170, 5185
	\exp_not:N 112,
	4343, 4356, 4359, 4362, 4708, 4712,
	4720, 4724, 4749, 4752, 4760, 4763,
	4791, 4794, 4798, 4803, 4809, 4812,
	4824, 4827, 4855, 4860, 4870, 4875
	\exp_not:n 279, 4008, 4031,
	4039, 4057, 4070, 4074, 4109, 4118,
	4140, 4148, 4155, 4179, 4193, 4210,

\msg_info:nnnn	7
..... 978, 985, 1015, 1387, 1421	
\msg_info:nnnnn	999
\msg_line_context:	117, 123,
127, 129, 132, 138, 144, 150, 156,	
161, 166, 171, 176, 182, 187, 190,	
193, 198, 202, 209, 214, 219, 226,	
235, 240, 245, 249, 251, 253, 255, 262	
\msg_new:nnn	115, 121,
126, 128, 130, 136, 142, 148, 154,	
159, 164, 169, 174, 179, 184, 189,	
191, 196, 201, 203, 205, 207, 212,	
217, 223, 225, 227, 232, 238, 243,	
248, 250, 252, 254, 256, 258, 260, 265	
\msg_warning:nn	
..... 2033, 2039, 2311, 2575, 4421	
\msg_warning:nnn	
..... 733, 754, 1549, 1556, 1712,	
1718, 2111, 2148, 2160, 2222, 2233,	
2275, 2300, 2384, 2434, 2444, 2594,	
2708, 2714, 2872, 2916, 2962, 3154,	
3160, 3241, 3860, 4240, 4934, 4950	
\msg_warning:nnnn	
.. 822, 839, 873, 891, 2064, 2251,	
2260, 2329, 2482, 2488, 2494, 2500,	
2530, 2743, 2749, 2755, 2761, 2797,	
2889, 2896, 2926, 3209, 3215, 3221,	
3227, 3300, 3325, 3868, 5019, 5079	
\msg_warning:nnnnn	859, 898, 2910, 4969
\msg_warning:nnnnnn	4976
N	
\newcounter	5, 5328, 5329
\NewDocumentCommand	
727, 744, 2576, 2581, 2843, 3386, 3434	
\newfloat	129
\NewHook	1629
\newsubfloat	129
\newtheorem	139
\nobreakspace	1530, 5700, 5701,
5703, 5704, 5706, 5708, 5905, 5906,	
5908, 5909, 5911, 5913, 6352, 6353,	
6355, 6356, 6358, 6360, 6569, 6570,	
6572, 6573, 6575, 6577, 6795, 6796,	
6798, 6799, 6801, 6803, 7009, 7010,	
7012, 7013, 7015, 7017, 7227, 7228,	
7230, 7231, 7233, 7235, 7236, 7361	
\NumCheckSetup	45
\NumsCheckSetup	45
P	
\PackageError	7
\pagename	140
\pagenote	129, 130
\pagenumbering	7
\pageref	84
\PagesCheckSetup	45
\paragraph	61
\part	138
\partname	140
prg commands:	
\prg_generate_conditional_-	
variant:Nnn	415,
463, 474, 501, 546, 554, 723, 1882	
\prg_new_conditional:Npnn	
.... 111, 113, 372, 449, 487, 548, 717	
\prg_new_protected_conditional:Npnn	
..... 406, 465, 537, 1875	
\prg_return_false:	
. 112, 114, 380, 384, 413, 457, 461,	
472, 495, 499, 544, 551, 552, 721, 1880	
\prg_return_true:	
.... 112, 114, 379, 382, 411, 456,	
459, 470, 494, 497, 542, 551, 720, 1879	
\prg_set_eq_conditional:NNn	725
prop commands:	
\prop_if_in:NnTF	36
\prop_if_in_p:Nn	72
\prop_item:Nn	39, 73
\prop_new:N	2322, 2377
\prop_put:Nnn	1540
\prop_remove:Nn	1539
\providecommand	3
\ProvidesExplPackage	14
\ProvidesFile	29
R	
\refstepcounter	3, 128, 132, 134, 135
regex commands:	
\regex_match:nnTF	1878
\renewlist	136
\RequirePackage	16, 17, 18, 19, 2029
S	
\scantokens	126
seq commands:	
\seq_clear:N	442, 817,
854, 935, 948, 1009, 1617, 2518,	
2782, 2856, 2869, 2920, 3454, 5266	
\seq_const_from_clist:Nn	21
\seq_count:N	
1375, 1389, 1408, 1423, 2523, 2532,	
2787, 2799, 3288, 3302, 3313, 3327	
\seq_gclear:N	1368, 1401, 3281, 3306
\seq_gconcat:NNN	619, 623
\seq_get_left:NN	
. 832, 843, 939, 987, 2860, 2898, 3806	
\seq_gput_right:Nn	950, 956, 2421

<code>\seq_gremove_all:Nn</code>	2453	3602, 3616, 3641, 3676, 3691, 3707
<code>\seq_gset_eq:NN</code>	435, 1037	<code>\sort_return_swapped:</code>
<code>\seq_gset_from_clist:Nn</code> 87, 91, 3480, 3556, 3565, 3586,
.... 562, 571, 583, 598, 606, 767, 782		3601, 3617, 3640, 3684, 3694, 3706
<code>\seq_gset_split:Nnn</code>	420	<code>\stepcounter</code>
<code>\seq_if_empty:NTF</code> ..	818, 855, 936,	134, 5346, 5365
976, 997, 2857, 2887, 2908, 3800, 4967		str commands:
<code>\seq_if_exist:NTF</code> ..	423, 433, 440, 451	<code>\str_case:nn</code>
<code>\seq_if_in:NnTF</code>		45
.. 836, 870, 914, 982, 1012, 2358,		<code>\str_case:nnTF</code>
2419, 2452, 2893, 2923, 3498, 4963		1112,
<code>\seq_item:Nn</code> 4702, 4707, 4713, 4715,		1634, 2093, 2130, 2206, 2630, 3003
4718, 4719, 4725, 4726, 4743, 4748,		<code>\str_compare:nNnTF</code>
4753, 4755, 4758, 4759, 4764, 4765,		3562
4796, 4797, 4804, 4806, 4841, 4853,		<code>\str_if_eq:nnTF</code>
4861, 4864, 4868, 4869, 4876, 4877		92, 4734
<code>\seq_map_break:n</code>	93, 3691, 3694	<code>\str_if_eq_p:nn</code> 5097, 5103, 5105, 5109
<code>\seq_map_function:NN</code>	3457	<code>\str_new:N</code>
<code>\seq_map_indexed_inline:Nn</code> .	44, 3652	2044
<code>\seq_map_inline:Nn</code> ...	1051, 1076,	<code>\str_set:Nn</code> ...
1313, 1352, 1430, 2443, 2456, 2504,		2049, 2051, 2053, 2055
2538, 2587, 2599, 2765, 2805, 2944,		<code>\string</code>
2968, 3231, 3271, 3333, 3688, 5534		5541, 5549
<code>\seq_map_tokens:Nn</code>	75	<code>\subbottom</code>
<code>\seq_new:N</code>	424,	129
434, 559, 560, 561, 570, 582, 597,		<code>\subcaption</code>
605, 618, 622, 762, 777, 907, 1029,		129
1605, 2351, 2409, 3432, 3451, 3709,		<code>\subcaptionref</code>
3726, 3749, 3750, 3751, 3752, 3753,		128
3754, 3755, 3756, 3757, 3758, 3759		<code>\subparagraph</code>
<code>\seq_pop_left:NN</code>	3798	138
<code>\seq_put_right:Nn</code>		<code>\subref</code>
..... 1013, 2360, 2924, 3501		137
<code>\seq_reverse:N</code>	1611	<code>\subsections</code>
<code>\seq_set_eq:NN</code>		138
. 425, 469, 3764, 3990, 4001, 4012,		<code>\subsubsection</code>
4062, 4098, 4134, 4168, 4184, 4266,		61
4300, 4311, 4532, 4577, 4599, 4628		<code>\subsubsections</code>
<code>\seq_set_from_clist:Nn</code> ...	1610, 3392	138
<code>\seq_set_split:Nnn</code>	418	<code>\subsubsubsection</code>
<code>\seq_sort:Nn</code>	86, 3460	61
<code>\seq_use:Nn</code>	4981	<code>\subtop</code>
<code>\g_tmpa_seq</code>	1368, 1370,	129
1375, 1384, 1389, 1401, 1403, 1408,		
1418, 1423, 3281, 3283, 3288, 3297,		
3302, 3306, 3308, 3313, 3322, 3327		
<code>\l_tmpa_seq</code> 1009, 1013, 1045, 2518,		
2520, 2523, 2527, 2532, 2782, 2784,		
2787, 2794, 2799, 2920, 2924, 2939		
<code>\setcounter</code> ..	5330, 5331, 5347, 5362, 5366	
<code>\sidefootnote</code>	129, 130	
sort commands:		
<code>\sort_return_same:</code>	87,	
91, 3467, 3472, 3546, 3566, 3587,		

T

<code>\tablename</code>	140
<code>\tag</code>	123, 132, 134
TeX and L ^A T _E X 2 _ε commands:	
<code>\@@sidecaption</code>	128
<code>\@Alph</code>	126
<code>\@addtoreset</code>	5
<code>\@auxout</code>	5540, 5548
<code>\@bsphack</code>	911, 5533
<code>\@capttype</code>	5430, 5664
<code>\@chapapp</code>	126
<code>\@currentcounter</code>	2, 3, 5, 62,
129, 132, 135, 27, 28, 52, 53, 2405, 2406	
<code>\@currentlabel</code>	3, 123, 129, 135
<code>\@elt</code>	5
<code>\@esphack</code>	961, 5553
<code>\@ifl@t@r</code>	3
<code>\@mem@scap@afterhook</code>	128
<code>\@memsubcaption</code>	129
<code>\@onlypreamble</code>	743, 757, 2875
<code>\bbl@loaded</code>	54
<code>\bbl@main@language</code>	24, 2073
<code>\c@lstnumber</code>	135
<code>\c@page</code>	7, 105
<code>\caption@subtypehook</code>	5665
<code>\hyper@link</code>	112, 121
<code>\hyper@linkfile</code>	5123
<code>\lst@AddToHook</code>	5612, 5614

<code>\lst@Init</code>	135		1863, 1914, 2914, 2952, 2976, 3007,
<code>\lst@label</code>	5613		3042, 3079, 3116, 3164, 3239, 3245,
<code>\lst@MakeCaption</code>	135		3279, 3341, 3362, 3408, 4235, 4828,
<code>\ltx@gobble</code>	131		4910, 4932, 4990, 5001, 5044, 5613
<code>\ltx@label</code> .	131, 5470, 5471, 5475, 5476	<code>\tl_if_empty:nTF</code>	
<code>\m@mscaplabel</code>	128	..	730, 746, 969, 1261, 1538, 1547,
<code>\MT@newlabel</code>	5541, 5549		1710, 2414, 2706, 2880, 3152, 5121
<code>\protected@write</code>	5540, 5548	<code>\tl_if_empty_p:N</code> .	2574, 4201, 4272,
<code>\zref@addprop</code> 21, 31, 46, 56, 58, 100, 110			4650, 4943, 4957, 5096, 5106, 5110
<code>\zref@default</code>	112, 4689, 4691	<code>\tl_if_empty_p:n</code>	1372, 1405,
<code>\zref@extractdefault</code>			2522, 2786, 3285, 3310, 3541, 3542,
.....	11, 121, 273, 279, 283		3551, 3552, 3577, 3578, 3593, 3608
<code>\zref@ifpropundefined</code> ..	42, 1267,	<code>\tl_if_eq:NNTF</code>	3512, 3535, 3842
	1554, 1716, 2712, 3158, 5127, 5579	<code>\tl_if_eq:NnTF</code>	
<code>\zref@ifrefcontainsprop</code> ..	42, 45,	...	1768, 3455, 3487, 3662, 3665,
	1744, 1752, 1811, 1829, 1841, 1886,		3690, 3693, 3810, 3858, 4892, 5142
	1928, 3863, 4694, 4777, 4831, 5130	<code>\tl_if_eq:nNTF</code> ..	1758, 1770, 1782,
<code>\zref@ifrefundefined</code>			1792, 1858, 1909, 1951, 3654, 5138,
	3462, 3464, 3476, 3831, 3833, 3838,		5144, 5166, 5170, 5185, 5536, 5544
	3855, 4237, 4248, 4449, 4772, 4886	<code>\tl_if_exist:NTF</code>	329,
<code>\zref@label</code>	131, 5464		339, 349, 356, 363, 374, 390, 400, 719
<code>\zref@localaddprop</code>		<code>\tl_if_exist_p:N</code>	2573
.....	5432, 5517, 5593, 5666	<code>\tl_if_novalue:nTF</code>	
<code>\ZREF@mainlist</code>	21, 31, 46,	2464, 2512, 2607, 2773
	56, 58, 100, 110, 5432, 5517, 5593, 5666	<code>\tl_map_break:n</code>	93
<code>\zref@newprop</code>	5, 7, 20, 22, 32,	<code>\tl_map_tokens:Nn</code>	85
	47, 57, 95, 109, 5429, 5494, 5580, 5663	<code>\tl_new:N</code>	101, 330, 340,
<code>\zref@refused</code>	3853		350, 357, 391, 401, 556, 557, 558,
<code>\zref@wrapper@babel</code> 83, 131, 3387, 5464			708, 709, 710, 711, 735, 750, 1542,
<code>\textendash</code>	1534,		2156, 2175, 2243, 2264, 2315, 2401,
	5755, 6019, 6631, 6853, 7067, 7290		3442, 3443, 3444, 3445, 3446, 3447,
<code>\thechapter</code>	126		3715, 3716, 3717, 3718, 3719, 3720,
<code>\thelstnumber</code>	135		3721, 3724, 3725, 3729, 3731, 3734,
<code>\thepage</code>	7, 106		3735, 3736, 3737, 3738, 3739, 3740,
<code>\thesection</code>	126		3741, 3742, 3743, 3744, 3745, 5433
tl commands:		<code>\tl_put_left:Nn</code> .	4322, 4329, 4372,
<code>\c_novalue_tl</code> ..	682, 683, 684, 685,		5003, 5004, 5046, 5048, 5050, 5052
	686, 687, 688, 2461, 2509, 2604, 2770	<code>\tl_put_right:Nn</code> .	4006, 4029, 4037,
<code>\tl_clear:N</code>	341, 365, 826,		4055, 4068, 4107, 4116, 4138, 4146,
	864, 877, 894, 903, 928, 937, 970,		4153, 4177, 4191, 4208, 4218, 4516,
	2585, 2848, 2858, 2881, 3766, 3767,		4539, 4551, 4585, 4607, 4616, 4636,
	3768, 3769, 3770, 3771, 3802, 4427,		4657, 4667, 4911, 4912, 4923, 5665
	4428, 4429, 4430, 4431, 4477, 4494,	<code>\tl_reverse:N</code>	3522, 3525
	4888, 4895, 4925, 5017, 5076, 5233	<code>\tl_set:Nn</code>	
<code>\tl_const:Nn</code>	1519	..	272, 331, 712, 737, 927, 971, 983,
<code>\tl_gclear:N</code>	358, 402		1551, 1558, 1560, 1749, 1817, 1821,
<code>\tl_gset:Nn</code> ...	106, 351, 392, 736, 751		1834, 1845, 1850, 1861, 1862, 1870,
<code>\tl_head:N</code>			1872, 1890, 1895, 1912, 1913, 1921,
..	3600, 3613, 3625, 3627, 3637, 3639		1923, 1932, 1937, 1958, 1959, 1967,
<code>\tl_head:n</code>	1859, 1910, 1952, 1956		1969, 2072, 2073, 2078, 2079, 2082,
<code>\tl_if_empty:NTF</code>	34, 83, 820,		2083, 2097, 2103, 2108, 2134, 2140,
	830, 857, 868, 889, 896, 1003, 1059,		2145, 2583, 2849, 2882, 2894, 3629,
	1084, 1116, 1151, 1188, 1225, 1273,		3631, 3812, 3813, 3986, 3988, 4260,
	1321, 1327, 1360, 1438, 1476, 1742,		

4283, 4293, 4339, 4462, 4464, 4475,
4492, 4907, 4908, 4921, 5434, 5436
\l_set_eq:NN 410, 4425
\l_show:N 4388
\l_tail:N 3630, 3632
\l_tail:n
... 1861, 1862, 1912, 1913, 1958, 1959
\l_use:N 294,
305, 316, 752, 916, 921, 951, 953, 957
\l_tmpa_tl 925, 949, 1845,
1859, 1861, 1890, 1901, 1910, 1912,
1932, 1943, 1952, 1958, 3413, 3414
\l_tmpb_tl . 1807, 1814, 1817, 1821,
1850, 1859, 1862, 1863, 1870, 1895,
1903, 1910, 1913, 1914, 1921, 1937,
1945, 1952, 1955, 1956, 1959, 1967

U

use commands:
\use:N 25, 28, 798, 4204, 4279, 4653, 5299
\UseHook 1844, 1889, 1931

V

\value 5347, 5366
\verbfootnote 129, 130

Z

\Z 1878
\zcDeclareLanguage . 12, 25, 727, 5690,
5895, 6348, 6563, 6792, 7006, 7224
\zcDeclareLanguageAlias
... 25, 744, 5691, 5692, 5693, 5694,
5695, 5696, 5697, 5898, 5899, 5900,
5901, 5902, 6349, 6564, 6565, 6566
\zcLanguageSetup
.... 20, 29, 30, 67, 72, 73, 140, 2843
\zcpageref 84, 3434
\zceref 20,
64, 66, 82–86, 92, 94, 133, 3386, 3439
\zcRefTypeSetup 20, 67, 2581
\zcsetup 20, 54, 64, 66, 67, 2576
\zlabel 128, 131, 132, 134, 135, 5409, 5613
zrefcheck commands:
\zrefcheck_zceref_beg_label: .. 3399
\zrefcheck_zceref_end_label_-
maybe: 3421
\zrefcheck_zceref_run_checks_on_-
labels:n 3422
zrefclever commands:
\zrefclever_language_if_declared:n
..... 725
\zrefclever_language_if_declared:nTF
..... 725
\zrefclever_language_varname:n ..
..... 715, 715

\l_zrefclever_ref_language_tl .. 711
zrefclever internal commands:
\l_zrefclever_abbrev_bool
..... 3734, 3921, 4914
\l_zrefclever_amsmath_subequations_-
bool 5478, 5492, 5516
\l_zrefclever_breqn_dgroup_bool
..... 5563, 5577, 5592
\l_zrefclever_cap_bool
..... 3734, 3917, 4902
\l_zrefclever_capfirst_bool ...
..... 1979, 1982, 4904
__zrefclever_compat_module:nn ..
..... 64,
2447, 2447, 5306, 5324, 5385, 5457,
5524, 5559, 5599, 5619, 5646, 5669
__zrefclever_counter_reset_by:n
6, 61, 62, 61, 63, 65, 69, 69, 5486, 5571
__zrefclever_counter_reset_by_-
aux:nn 76, 79
__zrefclever_counter_reset_by_-
auxi:nnn 86, 90
\l_zrefclever_counter_resetby_-
prop 5, 62, 72, 73, 2377, 2389
\l_zrefclever_counter_resettters_-
seq . 5, 61, 62, 75, 2351, 2358, 2361
\l_zrefclever_counter_type_prop
..... 4, 60, 36, 39, 2322, 2334
\l_zrefclever_current_counter_-
tl 3, 5, 62, 20, 24,
25, 37, 40, 42, 49, 50, 98, 2401, 2404
\l_zrefclever_current_language_-
tl 24,
54, 709, 2072, 2078, 2082, 2098, 2135
\l_zrefclever_endrangefunc_tl ..
... 3734, 3909, 4201, 4202, 4204,
4272, 4273, 4279, 4650, 4651, 4653
\l_zrefclever_endrangeprop_tl ..
..... 47, 1742, 1752, 3734, 3913
__zrefclever_extract:nnn
..... 11, 282, 282, 1847, 1852,
1892, 1897, 1934, 1939, 3583, 3585,
3598, 3615, 3703, 3705, 5149, 5151,
5155, 5157, 5175, 5177, 5181, 5183
__zrefclever_extract_default:Nnnn
... 11, 270, 270, 275, 1746, 1807,
1814, 1824, 1831, 3496, 3507, 3509,
3520, 3523, 3526, 3528, 3816, 3819
__zrefclever_extract_unexp:nnn .
..... 11, 121,
276, 276, 281, 1760, 1764, 1772,
1776, 1784, 1788, 1794, 1798, 4351,
4705, 4710, 4722, 4746, 4787, 4800,
4849, 4857, 4872, 5128, 5131, 5132,

5139, 5140, 5145, 5146, 5167, 5168,
 5171, 5172, 5187, 5191, 5537, 5545
 _zrefclever_extract_url_-
 unexp:n 4347, 4704,
 4745, 4783, 4845, 5125, 5125, 5135
 _zrefclever_get_enclosing_-
 counters_value:n
 5, 6, 59, 59, 64, 68, 97
 _zrefclever_get_endrange_-
 pagecomp:nnN 1884, 1925
 _zrefclever_get_endrange_-
 pagecomptwo:nnN 1926, 1971
 _zrefclever_get_endrange_-
 property:nnN 45, 1740, 1838
 _zrefclever_get_endrange_-
 stripprefix:nnN 1839, 1874
 _zrefclever_get_ref:nN
 . 112, 113, 4009, 4032, 4040, 4058,
 4071, 4075, 4110, 4119, 4141, 4149,
 4156, 4180, 4194, 4221, 4263, 4296,
 4331, 4519, 4542, 4554, 4588, 4610,
 4619, 4639, 4670, 4692, 4692, 4731
 _zrefclever_get_ref_endrange:nnN
 45,
 113, 4211, 4286, 4660, 4732, 4732, 4769
 _zrefclever_get_ref_first: ...
 112, 113, 117, 4323, 4373, 4770, 4770
 _zrefclever_get_rf_opt_bool:nN 125
 _zrefclever_get_rf_opt_-
 bool:nnnnN 20,
 3914, 3918, 3922, 5273, 5273, 5305
 _zrefclever_get_rf_opt_-
 seq:nnnN .. 20, 124, 3926, 3930,
 3934, 3938, 3942, 3946, 3950, 3954,
 3958, 3962, 4959, 5240, 5240, 5272
 _zrefclever_get_rf_opt_tl:nnnN
 20,
 22, 45, 124, 3410, 3781, 3785, 3789,
 3878, 3882, 3886, 3890, 3894, 3898,
 3902, 3906, 3910, 5207, 5207, 5239
 _zrefclever_hyperlink:nnn
 121, 4345,
 4703, 4744, 4781, 4843, 5119, 5119
 \l_zrefclever_hyperlink_bool ...
 1999, 2006, 2011, 2016, 2028, 2034,
 2041, 3438, 4698, 4739, 4837, 5094
 \l_zrefclever_hyperref_warn_-
 bool ... 2000, 2007, 2012, 2017, 2032
 _zrefclever_if_class_loaded:n .
 111, 113
 _zrefclever_if_class_loaded:nTF
 5387
 _zrefclever_if_package_-
 loaded:n 111, 111
 _zrefclever_if_package_-
 loaded:nTF 2026,
 2070, 2076, 2279, 5326, 5459, 5466,
 5526, 5561, 5601, 5621, 5648, 5671
 _zrefclever_is_integer_rgx:n ..
 1875, 1876, 1883
 _zrefclever_is_integer_rgx:nTF
 1901, 1903, 1943, 1945
 \l_zrefclever_label_a_tl
 . 91, 3715, 3799, 3818, 3831, 3853,
 3855, 3861, 3864, 3870, 3987, 4009,
 4032, 4040, 4075, 4141, 4156, 4206,
 4212, 4221, 4253, 4263, 4281, 4287,
 4296, 4449, 4453, 4463, 4476, 4493,
 4519, 4555, 4619, 4655, 4661, 4670
 \l_zrefclever_label_b_tl
 91, 3715,
 3802, 3807, 3821, 3833, 3838, 4453
 \l_zrefclever_label_count_int ..
 92, 3712, 3772,
 3876, 3979, 4432, 4458, 4686, 4946
 \l_zrefclever_label_enclval_a_-
 tl 3442, 3520, 3522, 3577,
 3593, 3613, 3625, 3629, 3630, 3637
 \l_zrefclever_label_enclval_b_-
 tl 3442, 3523, 3525, 3578,
 3600, 3608, 3627, 3631, 3632, 3639
 \l_zrefclever_label_extdoc_a_tl
 .. 3442, 3526, 3536, 3541, 3551, 3564
 \l_zrefclever_label_extdoc_b_tl
 .. 3442, 3528, 3537, 3542, 3552, 3563
 \l_zrefclever_label_type_a_tl ..
 3411, 3442, 3497, 3499,
 3502, 3508, 3513, 3662, 3690, 3782,
 3786, 3790, 3812, 3817, 3843, 3858,
 3879, 3883, 3887, 3891, 3895, 3899,
 3903, 3907, 3911, 3915, 3919, 3923,
 3927, 3931, 3935, 3939, 3943, 3947,
 3951, 3955, 3959, 3963, 3989, 4465
 \l_zrefclever_label_type_b_tl ..
 3442, 3510,
 3514, 3665, 3693, 3813, 3820, 3844
 _zrefclever_label_type_put_-
 new_right:n 85, 87, 3458, 3494, 3494
 \l_zrefclever_label_types_seq ..
 86, 3451, 3454, 3498, 3501, 3688
 _zrefclever_labels_in_sequence:nn
 .. 47, 93, 122, 4251, 4452, 5136, 5136
 \l_zrefclever_lang_decl_case_tl
 556, 937, 940, 983, 988, 1327, 1344,
 2858, 2861, 2894, 2899, 3245, 3262
 \l_zrefclever_lang_declension_-
 seq 556,
 816, 817, 818, 832, 836, 843, 934,

935, 936, 939, 976, 982, 987, 2855,
 2856, 2857, 2860, 2887, 2893, 2898
 \l_zrefclever_lang_gender_seq . .
 556, 853, 854, 855, 870, 947,
 948, 997, 1012, 2868, 2869, 2908, 2923
 _zrefclever_language_if_-
 declared:n 25, 717, 724, 726
 _zrefclever_language_if_-
 declared:n(TF) 24
 _zrefclever_language_if_-
 declared:nTF 291, 302, 313, 717,
 732, 748, 808, 912, 2109, 2146, 2846
 _zrefclever_language_varname:n
 24, 294, 305,
 316, 713, 713, 716, 719, 735, 736,
 750, 751, 752, 916, 921, 951, 953, 957
 \l_zrefclever_last_of_type_bool
 92, 3709, 3829,
 3834, 3835, 3839, 3845, 3846, 3969
 \l_zrefclever_lastsep_tl . 3734,
 3893, 4039, 4074, 4118, 4155, 4193
 \l_zrefclever_link_star_bool . . .
 . . 3393, 3432, 4699, 4740, 4838, 5095
 \l_zrefclever_listsep_tl
 . . . 3734, 3889, 4070, 4148, 4518,
 4541, 4553, 4587, 4609, 4618, 4638
 \g_zrefclever_loaded_langfiles_-
 seq 907, 915, 950, 956
 _zrefclever_ltxlabel:n
 131, 5461, 5471, 5476
 \l_zrefclever_main_language_tl .
 24,
 54, 710, 2073, 2079, 2083, 2104, 2141
 _zrefclever_mathtools_showonlyrefs:n
 3427, 5531
 \l_zrefclever_mathtools_-
 showonlyrefs_bool 3425, 5523, 5530
 _zrefclever_memoir_both_-
 labels:
 . . 5403, 5414, 5416, 5418, 5422, 5426
 \l_zrefclever_memoir_footnote_-
 type_tl 5433, 5434, 5436, 5440
 _zrefclever_memoir_label_and_-
 zlabel:n 5406, 5411
 _zrefclever_memoir_orig_-
 label:n 5405, 5408
 _zrefclever_name_default:
 4688, 4690, 4820
 \l_zrefclever_name_format_-
 fallback_tl 3721, 4921,
 4925, 4990, 5039, 5051, 5053, 5071
 \l_zrefclever_name_format_tl . . .
 . . . 3721, 4907, 4908, 4911, 4912,
 4922, 4923, 4996, 5003, 5004, 5012,
 5020, 5030, 5047, 5048, 5061, 5081
 \l_zrefclever_name_in_link_bool
 114,
 117, 3721, 4341, 4775, 5099, 5115, 5116
 \l_zrefclever_namefont_tl 3734,
 3901, 4344, 4360, 4792, 4810, 4825
 \l_zrefclever_nameinlink_str . . .
 2044, 2049, 2051,
 2053, 2055, 5097, 5103, 5105, 5109
 \l_zrefclever_namesep_tl
 . . 3734, 3881, 4795, 4813, 4821, 4829
 \l_zrefclever_next_is_same_bool
 92, 122, 3727,
 4446, 4479, 4496, 4502, 5160, 5198
 \l_zrefclever_next_maybe_range_-
 bool
 . . 92, 122, 3727, 4245, 4258, 4445,
 4472, 4485, 5152, 5159, 5178, 5196
 \l_zrefclever_noabbrev_first_-
 bool 1986, 1989, 4918
 \g_zrefclever_nocompat_bool . . .
 2408, 2415, 2451
 \l_zrefclever_nocompat_bool . . . 63
 \g_zrefclever_nocompat_modules_-
 seq 2409, 2419, 2422, 2443, 2452, 2453
 \l_zrefclever_nocompat_modules_-
 seq 63
 \l_zrefclever_nudge_comptosing_-
 bool . . . 2172, 2202, 2211, 2217, 4942
 \l_zrefclever_nudge_enabled_-
 bool 2170, 2180, 2182,
 2186, 2187, 2192, 2193, 4417, 4928
 \l_zrefclever_nudge_gender_bool
 2174, 2203, 2213, 2218, 4956
 \l_zrefclever_nudge_multitype_-
 bool . . . 2171, 2201, 2209, 2216, 4418
 \l_zrefclever_nudge_singular_-
 bool 2173, 2229, 4930
 _zrefclever_opt_bool_get:NN . . .
 537, 547
 _zrefclever_opt_bool_get:NN(TF)
 19
 _zrefclever_opt_bool_get:NNTF .
 . . . 537, 5276, 5281, 5286, 5291, 5296
 _zrefclever_opt_bool_gset_-
 false:N 18,
 503, 530, 536, 1485, 1502, 3364, 3372
 _zrefclever_opt_bool_gset_-
 true:N 18,
 503, 523, 529, 1447, 1464, 3343, 3351
 _zrefclever_opt_bool_if:N 548, 555
 _zrefclever_opt_bool_if:N(TF) . 19

```

\__zrefclever_opt_bool_if:NTF ...
... 548, 881
\__zrefclever_opt_bool_if_set:N .
... 487, 502
\__zrefclever_opt_bool_if_-
  set:N(TF) ... 18
\__zrefclever_opt_bool_if_-
  set:NTF ... 487,
  539, 550, 1440, 1456, 1478, 1494
\__zrefclever_opt_bool_set_-
  false:N 18, 503, 513, 522, 2551, 2822
\__zrefclever_opt_bool_set_-
  true:N 18, 503, 503, 512, 2546, 2813
\__zrefclever_opt_bool_unset:N .
... 17, 476, 476, 486, 2556, 2831
\__zrefclever_opt_seq_get:NN 465, 475
\__zrefclever_opt_seq_get:NN(TF) 17
\__zrefclever_opt_seq_get:NNTF .
... 465, 811, 848, 929, 942, 2850,
  2863, 5243, 5248, 5253, 5258, 5263
\__zrefclever_opt_seq_gset_-
  clist_split:Nn ... 16,
  417, 419, 1369, 1402, 3282, 3307
\__zrefclever_opt_seq_gset_eq:NN
... 16, 417,
  431, 437, 1378, 1411, 2931, 3291, 3316
\__zrefclever_opt_seq_if_set:N .
... 449, 464
\__zrefclever_opt_seq_if_-
  set:N(TF) ... 17
\__zrefclever_opt_seq_if_set:NTF
... 449, 467, 1020, 1362, 1394
\__zrefclever_opt_seq_set_clist_-
  split:Nn . 16, 417, 417, 2519, 2783
\__zrefclever_opt_seq_set_eq:NN .
... 16, 417, 421, 430, 2525, 2789
\__zrefclever_opt_seq_unset:N ...
... 16, 438, 438, 448, 2514, 2775
\__zrefclever_opt_tl_clear:N ...
... 14, 327,
  337, 346, 1638, 1643, 1658, 1673,
  1688, 2634, 2639, 2654, 2669, 2684
\__zrefclever_opt_tl_cset_-
  fallback:nn ... 1517, 1524
\__zrefclever_opt_tl_gclear:N ...
... 14, 327,
  354, 360, 3009, 3015, 3023, 3030,
  3051, 3067, 3088, 3104, 3125, 3141
\__zrefclever_opt_tl_gclear_if_-
  new:N ... 15, 386,
  396, 405, 1118, 1124, 1132, 1139,
  1160, 1176, 1197, 1213, 1234, 1250
\__zrefclever_opt_tl_get:NN 406, 416
\__zrefclever_opt_tl_get:NN(TF) . 16

\__zrefclever_opt_tl_get:NNTF ...
  406, 4992, 5007, 5026, 5035, 5056,
  5066, 5210, 5215, 5220, 5225, 5230
\__zrefclever_opt_tl_gset:N ... 14
\__zrefclever_opt_tl_gset:Nn ...
  .. 327, 347, 353, 2954, 2978, 2986,
  3044, 3059, 3081, 3096, 3118, 3133,
  3166, 3173, 3182, 3190, 3247, 3257
\__zrefclever_opt_tl_gset_if_-
  new:Nn ... 15,
  386, 386, 395, 1061, 1086, 1095,
  1153, 1168, 1190, 1205, 1227, 1242,
  1275, 1282, 1291, 1299, 1329, 1339
\__zrefclever_opt_tl_if_set:N . 372
\__zrefclever_opt_tl_if_set:N(TF)
... 15
\__zrefclever_opt_tl_if_set:NNTF .
... 372, 388, 398, 408
\__zrefclever_opt_tl_set:N ... 14
\__zrefclever_opt_tl_set:Nn ...
  ... 327, 327, 336,
  1652, 1667, 1682, 1722, 1728, 2470,
  2616, 2648, 2663, 2678, 2718, 2725
\__zrefclever_opt_tl_unset:N ...
  ... 14, 361, 361,
  371, 1697, 1702, 2466, 2609, 2693, 2698
\__zrefclever_opt_var_set_bool:n
  ... 13, 14, 325, 325, 332,
  333, 334, 342, 343, 344, 366, 367,
  368, 376, 378, 426, 427, 428, 443,
  444, 445, 453, 455, 481, 482, 483,
  491, 493, 508, 509, 510, 518, 519, 520
\__zrefclever_opt_varname_-
  fallback:nn ... 13,
  323, 323, 1520, 5231, 5264, 5297
\__zrefclever_opt_varname_-
  general:nn ... 12,
  284, 284, 1640, 1645, 1654, 1660,
  1669, 1675, 1684, 1690, 1699, 1704,
  1724, 1730, 2467, 2471, 2515, 2526,
  2547, 2552, 2557, 5211, 5244, 5277
\__zrefclever_opt_varname_lang_-
  default:nnn . 13, 300, 300, 310,
  1063, 1088, 1120, 1126, 1155, 1162,
  1192, 1199, 1229, 1236, 1277, 1284,
  1364, 1380, 1442, 1449, 1480, 1487,
  2956, 2980, 3011, 3017, 3046, 3053,
  3083, 3090, 3120, 3127, 3168, 3175,
  3293, 3345, 3366, 5226, 5259, 5292
\__zrefclever_opt_varname_lang_-
  type:nnnn ... 13,
  311, 311, 322, 1022, 1031, 1039,
  1097, 1134, 1141, 1170, 1178, 1207,
  1215, 1244, 1252, 1293, 1301, 1331,

```


1341, 1396, 1413, 1458, 1466, 1496,
1504, 2933, 2988, 3025, 3032, 3061,
3069, 3098, 3106, 3135, 3143, 3184,
3192, 3249, 3259, 3318, 3353, 3374,
5009, 5058, 5068, 5221, 5254, 5287
__zrefclever_opt_varname_
 language:nnn 12, 289,
 289, 299, 764, 769, 779, 784, 795,
 800, 813, 850, 883, 931, 944, 2852, 2865
__zrefclever_opt_varname_
 type:nnn 12, 286, 286, 288, 2611,
 2618, 2636, 2641, 2650, 2656, 2665,
 2671, 2680, 2686, 2695, 2700, 2720,
 2727, 2777, 2791, 2815, 2824, 2833,
 4994, 5028, 5037, 5216, 5249, 5282
__zrefclever_orig_ltxlabel:n ...
 5463, 5470, 5475
\g__zrefclever_page_format_tl ...
 7, 101, 106, 109
\l__zrefclever_pairsep_tl
 3734, 3885, 4008,
 4031, 4057, 4109, 4140, 4179, 4262
__zrefclever_process_language_
 settings: .. 57, 58, 806, 806, 3395
__zrefclever_prop_put_non_
 empty:Nnn 42, 1536, 1536, 2333, 2388
__zrefclever_provide_langfile:n
 20,
 30, 31, 83, 908, 908, 964, 2115, 3394
\l__zrefclever_range_beg_is_
 first_bool 3727,
 3777, 4096, 4132, 4166, 4437,
 4474, 4530, 4575, 4597, 4626, 4679
\l__zrefclever_range_beg_label_
 tl 92,
 3727, 3770, 4059, 4072, 4111, 4120,
 4150, 4181, 4195, 4205, 4430, 4475,
 4492, 4543, 4589, 4611, 4640, 4654
\l__zrefclever_range_count_int ..
 92,
 3727, 3775, 4021, 4085, 4435, 4478,
 4489, 4495, 4501, 4509, 4568, 4681
\l__zrefclever_range_end_ref_tl .
 ... 3727, 3771, 4207, 4213, 4282,
 4288, 4431, 4477, 4494, 4656, 4662
\l__zrefclever_range_same_count_
 int 92,
 3727, 3776, 3999, 4050, 4086, 4436,
 4480, 4497, 4503, 4548, 4569, 4682
\l__zrefclever_rangeseq_tl
 3734, 3897,
 4210, 4220, 4285, 4295, 4659, 4669
\l__zrefclever_rangetopair_bool .
 3734, 3925, 4246
\l__zrefclever_ref_count_int ...
 3712, 3774,
 4027, 4105, 4175, 4433, 4466, 4515,
 4538, 4550, 4584, 4606, 4615, 4635
\l__zrefclever_ref_decl_case_tl .
 27, 820, 825, 826, 830, 833,
 837, 841, 844, 889, 892, 894, 2156,
 2166, 5001, 5005, 5044, 5049, 5054
__zrefclever_ref_default: 4688,
 4688, 4729, 4735, 4773, 4814, 4880
\l__zrefclever_ref_gender_tl ...
 28, 857, 863,
 864, 868, 871, 876, 877, 896, 902,
 903, 2175, 2239, 4957, 4965, 4971, 4979
\l__zrefclever_ref_language_tl ..
 24, 27, 54, 708, 712, 809,
 814, 824, 842, 851, 861, 875, 884,
 893, 900, 2097, 2103, 2108, 2116,
 2134, 2140, 2145, 3394, 3412, 3783,
 3787, 3791, 3880, 3884, 3888, 3892,
 3896, 3900, 3904, 3908, 3912, 3916,
 3920, 3924, 3928, 3932, 3936, 3940,
 3944, 3948, 3952, 3956, 3960, 3964,
 4961, 4973, 4984, 5010, 5059, 5069
\l__zrefclever_ref_property_tl ..
 42, 47, 1542,
 1551, 1558, 1560, 1744, 1768, 1812,
 1829, 1841, 1848, 1853, 1886, 1893,
 1898, 1928, 1935, 1940, 3487, 3810,
 3865, 3869, 4694, 4779, 4833, 5142
\l__zrefclever_ref_propserity_tl 3455
\l__zrefclever_ref_typeset_font_
 tl 2243, 2245, 3405
\l__zrefclever_refbounds_first_
 pb_seq 3749,
 3937, 4013, 4063, 4135, 4186, 4267
\l__zrefclever_refbounds_first_
 rb_seq . 3749, 3941, 4169, 4301, 4630
\l__zrefclever_refbounds_first_
 seq 3749, 3929, 4312, 4533, 4579, 4601
\l__zrefclever_refbounds_first_
 sg_seq . 3749, 3933, 3991, 4002, 4099
\l__zrefclever_refbounds_last_
 pe_seq 3749, 3961,
 4010, 4033, 4060, 4112, 4142, 4264
\l__zrefclever_refbounds_last_
 re_seq
 .. 3749, 3965, 4214, 4222, 4289, 4297
\l__zrefclever_refbounds_last_
 seq 3749, 3957, 4041, 4076, 4121, 4157
\l__zrefclever_refbounds_mid_rb_
 seq ... 3749, 3949, 4182, 4196, 4641
\l__zrefclever_refbounds_mid_re_
 seq 3749, 3953, 4663, 4671

\l_zrefclever_refbounds_mid_seq	3079, 3100, 3108, 3116, 3137, 3145,
..... 3749, 3945, 4073, 4151,	3164, 3186, 3194, 3239, 3251, 3261,
4520, 4544, 4556, 4590, 4612, 4620	3279, 3320, 3341, 3355, 3362, 3376
\l_zrefclever_reffont_tl	\l_zrefclever_sort_decided_bool
..... 3734, 3905, 4709, 3448, 3531, 3545, 3555,
4721, 4750, 4761, 4799, 4856, 4871	3559, 3571, 3581, 3596, 3611, 3635
\l_zrefclever_reftype_override_-	_zrefclever_sort_default:nn ...
tl 34, 44, 2315, 2318 87, 3489, 3505, 3505
\g_zrefclever_rf_opts_bool_-	_zrefclever_sort_default_-
maybe_type_specific_seq	different_types:nn
..... 52, 561, 1431, 2539, 2806, 3334 44, 85, 90, 3516, 3648, 3648
\g_zrefclever_rf_opts_seq_-	_zrefclever_sort_default_same_-
refbounds_seq	type:nn ... 85, 87, 3515, 3518, 3518
..... 561, 1353, 2505, 2766, 3272	_zrefclever_sort_labels:
\g_zrefclever_rf_opts_tl_maybe_- 85-87, 91, 3403, 3452, 3452
type_specific_seq 561, 1077, 2969	_zrefclever_sort_page:nn
\g_zrefclever_rf_opts_tl_not_- 91, 3488, 3700, 3700
type_specific_seq	\l_zrefclever_sort_prior_a_int .
..... 561, 1052, 2588, 2945 3449,
\g_zrefclever_rf_opts_tl_-	3650, 3656, 3657, 3663, 3673, 3681
reference_seq 561, 2457	\l_zrefclever_sort_prior_b_int .
\g_zrefclever_rf_opts_tl_type_- 3449,
names_seq 561, 1314, 3232	3651, 3658, 3659, 3666, 3674, 3682
\g_zrefclever_rf_opts_tl_-	\l_zrefclever_tlastsep_tl
typesetup_seq 561, 2600 3734, 3792, 4411
\l_zrefclever_setup_language_tl	\l_zrefclever_tlistsep_tl
..... 556, 737, 765, 770, 780, 3734, 3788, 4382
785, 796, 801, 927, 979, 986, 1000,	\l_zrefclever_tpairsep_tl
1017, 1023, 1032, 1040, 1064, 1089, 3734, 3784, 4404
1098, 1121, 1127, 1135, 1142, 1156,	\l_zrefclever_type_count_int ...
1163, 1171, 1179, 1193, 1200, 1208,	. 92, 117, 3712, 3773, 4379, 4381,
1216, 1230, 1237, 1245, 1253, 1278,	4394, 4419, 4434, 4905, 4917, 5112
1285, 1294, 1302, 1332, 1342, 1365,	\l_zrefclever_type_first_label_-
1381, 1397, 1414, 1443, 1450, 1459,	tl 92,
1467, 1481, 1488, 1497, 1505, 2849,	114, 3715, 3768, 3986, 4237, 4248,
2890, 2897, 2911, 2928, 2934, 2957,	4252, 4280, 4331, 4348, 4352, 4428,
2981, 2989, 3012, 3018, 3026, 3033,	4462, 4772, 4778, 4784, 4788, 4801,
3047, 3054, 3062, 3070, 3084, 3091,	4832, 4846, 4850, 4858, 4873, 4886
3099, 3107, 3121, 3128, 3136, 3144,	\l_zrefclever_type_first_label_-
3169, 3176, 3185, 3193, 3250, 3260,	type_tl ... 92, 117, 3715, 3769,
3294, 3319, 3346, 3354, 3367, 3375	3988, 4241, 4429, 4464, 4893, 4936,
\l_zrefclever_setup_type_tl 556,	4952, 4960, 4972, 4978, 4995, 5011,
928, 970, 971, 1003, 1024, 1033,	5021, 5029, 5038, 5060, 5070, 5082
1041, 1059, 1084, 1099, 1116, 1136,	\l_zrefclever_type_first_-
1143, 1151, 1172, 1180, 1188, 1209,	refbounds_seq
1217, 1225, 1246, 1254, 1273, 1295,	... 3749, 3990, 4001, 4012, 4062,
1303, 1321, 1333, 1343, 1360, 1398,	4098, 4134, 4168, 4185, 4266, 4300,
1415, 1438, 1460, 1468, 1476, 1498,	4311, 4332, 4532, 4578, 4600, 4629,
1506, 2583, 2585, 2612, 2619, 2637,	4796, 4797, 4804, 4806, 4842, 4854,
2642, 2651, 2657, 2666, 2672, 2681,	4862, 4865, 4868, 4869, 4876, 4877
2687, 2696, 2701, 2721, 2728, 2778,	\l_zrefclever_type_first_-
2792, 2816, 2825, 2834, 2848, 2881,	refbounds_set_bool 3749,
2882, 2914, 2935, 2952, 2976, 2990,	3778, 3992, 4003, 4014, 4065, 4101,
3007, 3027, 3034, 3042, 3063, 3071,	

4137, 4171, 4188, 4268, 4302,
 4309, 4438, 4535, 4581, 4603, 4632
 \l_zrefclever_type_name_gender_
 seq ... 3721, 4962, 4964, 4967, 4982
 \l_zrefclever_type_name_
 missing_bool
 .. 3721, 4818, 4889, 4896, 5018, 5078
 _zrefclever_type_name_setup: ..
 20, 22, 114, 4318, 4884, 4884
 \l_zrefclever_type_name_tl
 114, 117,
 3721, 4355, 4361, 4793, 4811, 4826,
 4828, 4888, 4895, 4999, 5015, 5017,
 5033, 5042, 5064, 5074, 5076, 5096
 \l_zrefclever_typeset_compress_
 bool 1620, 1623, 4447
 \l_zrefclever_typeset_labels_
 seq 91, 3709, 3764, 3798, 3800, 3806
 \l_zrefclever_typeset_last_bool
 92, 3709,
 3795, 3796, 3803, 3828, 4391, 5111
 \l_zrefclever_typeset_name_bool
 .. 1569, 1576, 1581, 1586, 4320, 4336
 \l_zrefclever_typeset_queue_
 curr_tl 92,
 94, 112, 117, 3715, 3767, 4006,
 4029, 4037, 4055, 4068, 4107,
 4116, 4138, 4146, 4153, 4177, 4191,
 4208, 4218, 4235, 4260, 4283, 4293,
 4322, 4329, 4339, 4372, 4388, 4399,
 4405, 4412, 4426, 4427, 4516, 4539,
 4551, 4585, 4607, 4616, 4636, 4657,
 4667, 4910, 4932, 4943, 5106, 5110
 \l_zrefclever_typeset_queue_
 prev_tl . 92, 3715, 3766, 4383, 4425
 \l_zrefclever_typeset_range_
 bool ... 1754, 1972, 1975, 3402, 4233
 \l_zrefclever_typeset_ref_bool .
 .. 1568, 1575, 1580, 1585, 4320, 4326
 _zrefclever_typeset_refs:
 91, 93, 94, 3406, 3762, 3762
 _zrefclever_typeset_refs_last_
 of_type:
 .. 98, 112, 114, 117, 3971, 3976, 3976
 _zrefclever_typeset_refs_not_
 last_of_type:
 ... 93, 99, 112, 122, 3973, 4441, 4441
 \l_zrefclever_typeset_sort_bool
 1596, 1599, 3401
 \l_zrefclever_typesort_seq
 . 44, 90, 1605, 1610, 1611, 1617, 3652
 \l_zrefclever_verbose_testing_
 bool 3761, 4387
 _zrefclever_zcref:nnn
 27, 56, 3387, 3388
 _zrefclever_zcref:nnnn 83, 85, 3388
 \l_zrefclever_zcref_labels_seq .
 85, 86, 3392,
 3423, 3428, 3432, 3457, 3460, 3765
 \l_zrefclever_zcref_note_tl ...
 2264, 2267, 3408, 3415
 \l_zrefclever_zcref_with_check_
 bool 2271, 2288, 3398, 3419
 _zrefclever_zcsetup:n
 . 67, 2577, 2578, 2578, 2580, 5310,
 5334, 5340, 5348, 5370, 5389, 5439,
 5443, 5444, 5453, 5481, 5515, 5566,
 5591, 5603, 5615, 5630, 5650, 5673
 \l_zrefclever_zrefcheck_
 available_bool
 .. 2270, 2283, 2295, 2307, 3397, 3418