

# The zref-clever package<sup>\*</sup>

## Code documentation

Gustavo Barros<sup>†</sup>

2022-12-27

### EXPERIMENTAL

## Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>3</b>
<b>3</b>	<b>zref setup</b>	<b>3</b>
<b>4</b>	<b>Plumbing</b>	<b>7</b>
4.1	Auxiliary . . . . .	7
4.2	Messages . . . . .	8
4.3	Data extraction . . . . .	10
4.4	Option infra . . . . .	11
4.5	Reference format . . . . .	20
4.6	Languages . . . . .	24
4.7	Language files . . . . .	29
4.8	Options . . . . .	42
<b>5</b>	<b>Configuration</b>	<b>66</b>
5.1	\zcsetup . . . . .	66
5.2	\zcRefTypeSetup . . . . .	67
5.3	\zcLanguageSetup . . . . .	72
<b>6</b>	<b>User interface</b>	<b>82</b>
6.1	\zceref . . . . .	82
6.2	\zcpageref . . . . .	84
<b>7</b>	<b>Sorting</b>	<b>84</b>
<b>8</b>	<b>Typesetting</b>	<b>91</b>

---

<sup>\*</sup>This file describes v0.3.2, released 2022-12-27.

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

<b>9</b>	<b>Compatibility</b>	<b>126</b>
9.1	appendix	126
9.2	appendices	127
9.3	memoir	128
9.4	KOMA	130
9.5	amsmath	131
9.6	mathtools	134
9.7	breqn	135
9.8	listings	136
9.9	enumitem	136
9.10	subcaption	137
9.11	subfig	138
<b>10</b>	<b>Language files</b>	<b>138</b>
10.1	Localization guidelines	138
10.2	English	141
10.3	German	145
10.4	French	153
10.5	Portuguese	158
10.6	Spanish	162
10.7	Dutch	166
10.8	Italian	170
	<b>Index</b>	<b>175</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L<sup>A</sup>T<sub>E</sub>X3 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} {2022-12-27} {0.3.2}
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-abspage }
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-abspage` provides the `abspage` property which gives us a safe and easy way to sort labels for page references.

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

```

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

```

The reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the⟨counter⟩` and store it “clean” in `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     \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
35     \l__zrefclever_current_counter_tl
36     {
37         \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
38         { \l__zrefclever_current_counter_tl }
39     }
40     { \l__zrefclever_current_counter_tl }
41 }
42 \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`’).

```

43 \zref@newprop { zc@cntval } [0]
44 {
45     \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
46     { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
47     {
48         \cs_if_exist:cT { c@ \@currentcounter }
49         { \int_use:c { c@ \@currentcounter } }
50     }
51 }
52 \zref@addprop \ZREF@mainlist { zc@cntval }
53 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
54 \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>}

55 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1

```

```

56 {
57   \cs_if_exist:cT { c@ \_zrefclever_counter_reset_by:n {#1} }
58   {
59     { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
60     \_zrefclever_get_enclosing_counters_value:e
61     { \_zrefclever_counter_reset_by:n {#1} }
62   }
63 }

```

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](https://tex.stackexchange.com/q/611370/#comment1529282_611385)).

```

64 \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 }

65 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
66 {
67   \bool_if:nTF
68   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
69   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
70   {
71     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
72     { \_zrefclever_counter_reset_by_aux:nn {#1} }
73   }
74 }
75 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
76 {
77   \cs_if_exist:cT { c@ #2 }
78   {
79     \tl_if_empty:cF { cl@ #2 }
80     {
81       \tl_map_tokens:cn { cl@ #2 }
82       { \_zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
83     }
84   }
85 }
86 \cs_new:Npn \_zrefclever_counter_reset_by_auxi:nnn #1#2#3
87 {
88   \str_if_eq:nnT {#2} {#3}
89   { \tl_map_break:n { \seq_map_break:n {#1} } }
90 }

```

(End definition for `\_zrefclever_counter_reset_by:n`.)

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

```

91 \zref@newprop { zc@enclval }
92 {
93   \__zrefclever_get_enclosing_counters_value:e
94   \l__zrefclever_current_counter_tl
95 }
96 \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}`.

```

97 \tl_new:N \g__zrefclever_page_format_tl
98 \AddToHook { shipout / before }
99 {
100   \group_begin:
101   \int_set:Nn \c@page { 1 }
102   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
103   \group_end:
104 }
105 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
106 \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.

```

107 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
108 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
109 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
110 { \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

```

111 \msg_new:nnn { zref-clever } { option-not-type-specific }
112 {
113   Option~'#1'~is-not-type-specific~\msg_line_context:~
114   Set-it-in~'\iow_char:N\zcLanguageSetup'~before-first~'type'~
115   switch-or-as~package~option.
116 }
117 \msg_new:nnn { zref-clever } { option-only-type-specific }
118 {
119   No~type~specified~for~option~'#1'~\msg_line_context:~
120   Set-it~after~'type'~switch.
121 }
122 \msg_new:nnn { zref-clever } { key-requires-value }
123 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:~ }
124 \msg_new:nnn { zref-clever } { language-declared }
125 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing~to~do. }
126 \msg_new:nnn { zref-clever } { unknown-language-alias }
127 {
128   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
129   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
130   '\iow_char:N\zcDeclareLanguageAlias'.
131 }
132 \msg_new:nnn { zref-clever } { unknown-language-setup }
133 {
134   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
135   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
136   '\iow_char:N\zcDeclareLanguageAlias'.
137 }
138 \msg_new:nnn { zref-clever } { unknown-language-opt }
139 {
140   Language~'#1'~is~unknown~\msg_line_context:~
141   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
142   '\iow_char:N\zcDeclareLanguageAlias'.
143 }
144 \msg_new:nnn { zref-clever } { unknown-language-decl }
145 {
146   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
147   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
148   '\iow_char:N\zcDeclareLanguageAlias'.
149 }
150 \msg_new:nnn { zref-clever } { language-no-decl-ref }
151 {
152   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
153   Nothing~to~do~with~option~'d=#2'.
154 }
155 \msg_new:nnn { zref-clever } { language-no-gender }
156 {
157   Language~'#1'~has~no~declared~gender~\msg_line_context:~
158   Nothing~to~do~with~option~'#2=#3'.
159 }
160 \msg_new:nnn { zref-clever } { language-no-decl-setup }

```



```

161 {
162   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
163   Nothing~to~do~with~option~'case=#2'.
164 }
165 \msg_new:nnn { zref-clever } { unknown-decl-case }
166 {
167   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:~
168   Using~default~declension~case.
169 }
170 \msg_new:nnn { zref-clever } { nudge-multitype }
171 {
172   Reference~with~multiple~types~\msg_line_context:~
173   You~may~wish~to~separate~them~or~review~language~around~it.
174 }
175 \msg_new:nnn { zref-clever } { nudge-comptosing }
176 {
177   Multiple~labels~have~been~compressed~into~singular~type~name~
178   for~type~'#1'~\msg_line_context:.
179 }
180 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
181 {
182   Option~'sg'~signals~that~a~singular~type~name~was~expected~
183   \msg_line_context:~But~type~'#1'~has~plural~type~name.
184 }
185 \msg_new:nnn { zref-clever } { gender-not-declared }
186 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
187 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
188 {
189   Gender~mismatch~for~type~'#1'~\msg_line_context:~
190   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
191 }
192 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
193 {
194   You've~specified~'g=#1'~\msg_line_context:~
195   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
196 }
197 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
198 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
199 \msg_new:nnn { zref-clever } { option-document-only }
200 { Option~'#1'~is~only~available~after~\iow_char:N\\begin\{document\}. }
201 \msg_new:nnn { zref-clever } { langfile-loaded }
202 { Loaded~'#1'~language~file. }
203 \msg_new:nnn { zref-clever } { zref-property-undefined }
204 {
205   Option~'ref=#1'~requested~\msg_line_context:~
206   But~the~property~'#1'~is~not~declared,~falling~back~to~'default'.
207 }
208 \msg_new:nnn { zref-clever } { endrange-property-undefined }
209 {
210   Option~'endrange=#1'~requested~\msg_line_context:~
211   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
212 }
213 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
214 {

```

```

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

```

### 4.3 Data extraction

`\_zrefclever_extract_default:Nnn` 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}

266 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
267   {
268     \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
269     { \zref@extractdefault {#2} {#3} {#4} }
270   }
271 \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}

272 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
273   {
274     \exp_args:NNo \exp_args:No
275     \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
276   }
277 \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}

278 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
279   { \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](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}

280 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
281 { 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}

282 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
283 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
284 \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}

285 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
286 {
287   \_zrefclever_language_if_declared:nTF {#1}
288   {
289     g\_zrefclever_opt_language_
290     \tl_use:c { \_zrefclever_language_varname:n {#1} }
291     _ #2 _ #3
292   }
293   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
294 }
295 \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>}}
296 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
297 {
298   \_zrefclever_language_if_declared:nTF {#1}
299   {
300     g__zrefclever_opt_lang_
301     \tl_use:c { \_zrefclever_language_varname:n {#1} }
302     _default_ #2 _ #3
303   }
304   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
305 }
306 \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>}}
307 \cs_new:Npn \_zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
308 {
309   \_zrefclever_language_if_declared:nTF {#1}
310   {
311     g__zrefclever_opt_lang_
312     \tl_use:c { \_zrefclever_language_varname:n {#1} }
313     _type_ #2 _ #3 _ #4
314   }
315   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
316 }
317 \cs_generate_variant:Nn
318 \_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>}}
319 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
320 { c__zrefclever_opt_fallback_ #1 _ #2 }
(End definition for \_zrefclever_opt_varname_fallback:nn.)

```

`\_zrefclever_opt_var_set_bool:n` The L<sup>A</sup>T<sub>E</sub>X3 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](https://tex.stackexchange.com/questions/633341/#comment1579825_633347)

\\_\_zrefclever\_opt\_var\_set\_bool:n {\<option var>}

321 \cs\_new:Npn \\_\_zrefclever\_opt\_var\_set\_bool:n #1  
322 { \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

323 \cs\_new\_protected:Npn \\_\_zrefclever\_opt\_tl\_set:Nn #1#2  
324 {  
325   \tl\_if\_exist:NF #1  
326   { \tl\_new:N #1 }  
327   \tl\_set:Nn #1 {#2}  
328   \bool\_if\_exist:cF { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} }  
329   { \bool\_new:c { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} } }  
330   \bool\_set\_true:c { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} }  
331 }  
332 \cs\_generate\_variant:Nn \\_\_zrefclever\_opt\_tl\_set:Nn { cn }  
333 \cs\_new\_protected:Npn \\_\_zrefclever\_opt\_tl\_clear:N #1  
334 {  
335   \tl\_if\_exist:NF #1  
336   { \tl\_new:N #1 }  
337   \tl\_clear:N #1  
338   \bool\_if\_exist:cF { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} }  
339   { \bool\_new:c { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} } }  
340   \bool\_set\_true:c { \\_\_zrefclever\_opt\_var\_set\_bool:n {#1} }  
341 }  
342 \cs\_generate\_variant:Nn \\_\_zrefclever\_opt\_tl\_clear:N { c }  
343 \cs\_new\_protected:Npn \\_\_zrefclever\_opt\_tl\_gset:Nn #1#2  
344 {  
345   \tl\_if\_exist:NF #1  
346   { \tl\_new:N #1 }  
347   \tl\_gset:Nn #1 {#2}  
348 }  
349 \cs\_generate\_variant:Nn \\_\_zrefclever\_opt\_tl\_gset:Nn { cn }  
350 \cs\_new\_protected:Npn \\_\_zrefclever\_opt\_tl\_gclear:N #1  
351 {  
352   \tl\_if\_exist:NF #1  
353   { \tl\_new:N #1 }  
354   \tl\_gclear:N #1  
355 }  
356 \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>}

357 \cs\_new\_protected:Npn \\_\_zrefclever\_opt\_tl\_unset:N #1  
358 {  
359   \tl\_if\_exist:NT #1

```

360     {
361       \tl_clear:N #1 % ?
362       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
363       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
364       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
365     }
366   }
367   \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>}
368 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
369 {
370   \tl_if_exist:NTF #1
371   {
372     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
373     {
374       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
375       { \prg_return_true: }
376       { \prg_return_false: }
377     }
378     { \prg_return_true: }
379   }
380   { \prg_return_false: }
381 }

```

(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>}
382 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
383 {
384   \__zrefclever_opt_tl_if_set:NF #1
385   {
386     \tl_if_exist:NF #1
387     { \tl_new:N #1 }
388     \tl_gset:Nn #1 {#2}
389   }
390 }
391 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
392 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
393 {
394   \__zrefclever_opt_tl_if_set:NF #1
395   {
396     \tl_if_exist:NF #1
397     { \tl_new:N #1 }
398     \tl_gclear:N #1
399   }

```

```

400 }
401 \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>}}
402 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
403 {
404     \__zrefclever_opt_tl_if_set:NTF #1
405     {
406         \tl_set_eq:NN #2 #1
407         \prg_return_true:
408     }
409     { \prg_return_false: }
410 }
411 \prg_generate_conditional_variant:Nnn
412 \__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>}
413 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
414 { \seq_set_split:Nnn #1 { , } {#2} }
415 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
416 { \seq_gset_split:Nnn #1 { , } {#2} }
417 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
418 {
419     \seq_if_exist:NF #1
420     { \seq_new:N #1 }
421     \seq_set_eq:NN #1 #2
422     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
423     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
424     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
425 }
426 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
427 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
428 {
429     \seq_if_exist:NF #1
430     { \seq_new:N #1 }
431     \seq_gset_eq:NN #1 #2
432 }
433 \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>}

```



```

434 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
435 {
436   \seq_if_exist:NT #1
437   {
438     \seq_clear:N #1 % ?
439     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
440     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
441     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
442   }
443 }
444 \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) {\langle option seq \rangle} {\langle true \rangle} {\langle false \rangle}
445 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
446 {
447   \seq_if_exist:NTF #1
448   {
449     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
450     {
451       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
452       { \prg_return_true: }
453       { \prg_return_false: }
454     }
455     { \prg_return_true: }
456   }
457   { \prg_return_false: }
458 }
459 \prg_generate_conditional_variant:Nnn
460 \__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) {\langle option seq to get \rangle} {\langle seq var to set \rangle}
{\langle true \rangle} {\langle false \rangle}
461 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
462 {
463   \__zrefclever_opt_seq_if_set:NTF #1
464   {
465     \seq_set_eq:NN #2 #1
466     \prg_return_true:
467   }
468   { \prg_return_false: }
469 }
470 \prg_generate_conditional_variant:Nnn
471 \__zrefclever_opt_seq_get:NN { cN } { F }

```

(End definition for \\_\_zrefclever\_opt\_seq\_get:NNTF.)

\\_zrefclever\_opt\_bool\_unset:N Unset  $\langle option bool \rangle$ .

```

\_zrefclever_opt_bool_unset:N {\langle option bool \rangle}

```

```

472 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
473 {
474   \bool_if_exist:NT #1
475   {
476     % \bool_set_false:N #1 % ?
477     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
478     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
479     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
480   }
481 }
482 \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>}
483 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
484 {
485   \bool_if_exist:NTF #1
486   {
487     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
488     {
489       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
490       { \prg_return_true: }
491       { \prg_return_false: }
492     }
493     { \prg_return_true: }
494   }
495   { \prg_return_false: }
496 }
497 \prg_generate_conditional_variant:Nnn
498 \__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>}
499 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
500 {
501   \bool_if_exist:NF #1
502   { \bool_new:N #1 }
503   \bool_set_true:N #1
504   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
505   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
506   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
507 }
508 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
509 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
510 {
511   \bool_if_exist:NF #1
512   { \bool_new:N #1 }

```

```

513     \bool_set_false:N #1
514     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
515       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
516     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
517   }
518 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
519 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
520 {
521   \bool_if_exist:NF #1
522     { \bool_new:N #1 }
523   \bool_gset_true:N #1
524 }
525 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
526 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
527 {
528   \bool_if_exist:NF #1
529     { \bool_new:N #1 }
530   \bool_gset_false:N #1
531 }
532 \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}}
533 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
534 {
535   \__zrefclever_opt_bool_if_set:NTF #1
536   {
537     \bool_set_eq:NN #2 #1
538     \prg_return_true:
539   }
540   { \prg_return_false: }
541 }
542 \prg_generate_conditional_variant:Nnn
543   \__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}}
544 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
545 {
546   \__zrefclever_opt_bool_if_set:NTF #1
547   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
548   { \prg_return_false: }
549 }
550 \prg_generate_conditional_variant:Nnn
551   \__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](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>	

```

552 \tl_new:N \l__zrefclever_setup_type_tl
553 \tl_new:N \l__zrefclever_setup_language_tl
554 \tl_new:N \l__zrefclever_lang_decl_case_tl
555 \seq_new:N \l__zrefclever_lang_declension_seq
556 \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<sup>A</sup>T<sub>E</sub>X3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```

557 \seq_new:N \g_zrefclever_rf_opts_tl_not_type_specific_seq
558 \seq_gset_from_clist:Nn
559 \g_zrefclever_rf_opts_tl_not_type_specific_seq
560 {
561     tpairsep ,
562     tlistsep ,
563     tlastsep ,
564     notesep ,
565 }
566 \seq_new:N \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
567 \seq_gset_from_clist:Nn
568 \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
569 {
570     namesep ,
571     pairsep ,
572     listsep ,
573     lastsep ,
574     rangesep ,
575     namefont ,
576     reffont ,
577 }
578 \seq_new:N \g_zrefclever_rf_opts_seq_refbounds_seq
579 \seq_gset_from_clist:Nn
580 \g_zrefclever_rf_opts_seq_refbounds_seq
581 {
582     refbounds-first ,
583     refbounds-first-sg ,
584     refbounds-first-pb ,
585     refbounds-first-rb ,
586     refbounds-mid ,
587     refbounds-mid-rb ,
588     refbounds-mid-re ,
589     refbounds-last ,
590     refbounds-last-pe ,
591     refbounds-last-re ,
592 }
593 \seq_new:N \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
594 \seq_gset_from_clist:Nn
595 \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
596 {
597     cap ,
598     abbrev ,
599     rangetopair ,
600 }

```

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:.

```

601 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
602 \seq_gset_from_clist:Nn
603 \g__zrefclever_rf_opts_tl_type_names_seq
604 {
605   Name-sg ,
606   name-sg ,
607   Name-pl ,
608   name-pl ,
609   Name-sg-ab ,
610   name-sg-ab ,
611   Name-pl-ab ,
612   name-pl-ab ,
613 }

```

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

```

614 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
615 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
616 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
617 \g__zrefclever_rf_opts_tl_type_names_seq
618 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
619 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
620 \g__zrefclever_rf_opts_tl_not_type_specific_seq
621 \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.

```

622 \clist_map_inline:nn
623 {
624   reference ,
625   typesetup ,
626   langsetup ,
627   langfile ,
628 }
629 {
630   \keys_define:nn { zref-clever/ #1 }
631   {
632     +refbounds-first .meta:n =
633     {
634       refbounds-first = {##1} ,
635       refbounds-first-sg = {##1} ,
636       refbounds-first-pb = {##1} ,
637       refbounds-first-rb = {##1} ,
638     } ,
639     +refbounds-mid .meta:n =
640     {
641       refbounds-mid = {##1} ,
642       refbounds-mid-rb = {##1} ,
643       refbounds-mid-re = {##1} ,
644     } ,
645     +refbounds-last .meta:n =
646     {
647       refbounds-last = {##1} ,

```

```

648         refbounds-last-pe = {##1} ,
649         refbounds-last-re = {##1} ,
650     } ,
651     +refbounds-rb .meta:n =
652     {
653         refbounds-first-rb = {##1} ,
654         refbounds-mid-rb = {##1} ,
655     } ,
656     +refbounds-re .meta:n =
657     {
658         refbounds-mid-re = {##1} ,
659         refbounds-last-re = {##1} ,
660     } ,
661     +refbounds .meta:n =
662     {
663         +refbounds-first = {##1} ,
664         +refbounds-mid = {##1} ,
665         +refbounds-last = {##1} ,
666     } ,
667     refbounds .meta:n = { +refbounds = {##1} } ,
668 }
669 }
670 \clist_map_inline:nn
671 {
672     reference ,
673     typesetup ,
674 }
675 {
676     \keys_define:nn { zref-clever/ #1 }
677     {
678         +refbounds-first .default:o = \c_novalue_tl ,
679         +refbounds-mid .default:o = \c_novalue_tl ,
680         +refbounds-last .default:o = \c_novalue_tl ,
681         +refbounds-rb .default:o = \c_novalue_tl ,
682         +refbounds-re .default:o = \c_novalue_tl ,
683         +refbounds .default:o = \c_novalue_tl ,
684         refbounds .default:o = \c_novalue_tl ,
685     }
686 }
687 \clist_map_inline:nn
688 {
689     langsetup ,
690     langfile ,
691 }
692 {
693     \keys_define:nn { zref-clever/ #1 }
694     {
695         +refbounds-first .value_required:n = true ,
696         +refbounds-mid .value_required:n = true ,
697         +refbounds-last .value_required:n = true ,
698         +refbounds-rb .value_required:n = true ,
699         +refbounds-re .value_required:n = true ,
700         +refbounds .value_required:n = true ,
701         refbounds .value_required:n = true ,

```

```

702     }
703 }

```

## 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.

```

704 \tl_new:N \l__zrefclever_ref_language_tl
705 \tl_new:N \l__zrefclever_current_language_tl
706 \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`.

```

707 \tl_new:N \l_zrefclever_ref_language_tl
708 \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>}

709 \cs_new:Npn \_zrefclever_language_varname:n #1
710 { 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`.

```

711 \cs_set_eq:NN \zrefclever_language_varname:n
712 \_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>}

713 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
714 {
715     \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
716     { \prg_return_true: }
717     { \prg_return_false: }
718 }
719 \prg_generate_conditional_variant:Nnn
720 \_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`.

```

721 \prg_set_eq_conditional:Nn \zrefclever_language_if_declared:n
722 \__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 [<options>] {<language>}

723 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
724 {
725   \group_begin:
726   \tl_if_empty:nF {#2}
727   {
728     \__zrefclever_language_if_declared:nTF {#2}
729     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
730     {
731       \tl_new:c { \__zrefclever_language_varname:n {#2} }
732       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
733       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
734       \keys_set:nn { zref-clever/declarelang } {#1}
735     }
736   }
737   \group_end:
738 }
739 \@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 {<language alias>} {<aliased language>}

740 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
741 {
742   \tl_if_empty:nF {#1}
743   {

```

```

744     \_zrefclever_language_if_declared:nTF {#2}
745     {
746         \tl_new:c { \_zrefclever_language_varname:n {#1} }
747         \tl_gset:cx { \_zrefclever_language_varname:n {#1} }
748         { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
749     }
750     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
751 }
752 }
753 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

754 \keys_define:nn { zref-clever/declarelang }
755 {
756     declension .code:n =
757     {
758         \seq_new:c
759         {
760             \_zrefclever_opt_varname_language:enn
761             { \l__zrefclever_setup_language_tl } { declension } { seq }
762         }
763         \seq_gset_from_clist:cn
764         {
765             \_zrefclever_opt_varname_language:enn
766             { \l__zrefclever_setup_language_tl } { declension } { seq }
767         }
768         {#1}
769     } ,
770     declension .value_required:n = true ,
771     gender .code:n =
772     {
773         \seq_new:c
774         {
775             \_zrefclever_opt_varname_language:enn
776             { \l__zrefclever_setup_language_tl } { gender } { seq }
777         }
778         \seq_gset_from_clist:cn
779         {
780             \_zrefclever_opt_varname_language:enn
781             { \l__zrefclever_setup_language_tl } { gender } { seq }
782         }
783         {#1}
784     } ,
785     gender .value_required:n = true ,
786     allcaps .choices:nn =
787     { true , false }
788     {
789         \bool_new:c
790         {
791             \_zrefclever_opt_varname_language:enn
792             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
793         }
794         \use:c { bool_gset_ \l_keys_choice_tl :c }
795         {

```

```

796         \_zrefclever_opt_varname_language:enn
797         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
798     }
799 },
800 allcaps .default:n = true ,
801 }

```

`\_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.

```

802 \cs_new_protected:Npn \_zrefclever_process_language_settings:
803 {
804     \_zrefclever_language_if_declared:xTF
805     { \l__zrefclever_ref_language_tl }
806     {

```

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.

```

807     \_zrefclever_opt_seq_get:cNF
808     {
809         \_zrefclever_opt_varname_language:enn
810         { \l__zrefclever_ref_language_tl } { declension } { seq }
811     }
812     \l__zrefclever_lang_declension_seq
813     { \seq_clear:N \l__zrefclever_lang_declension_seq }
814     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
815     {
816         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
817         {
818             \msg_warning:nxxx { zref-clever }
819             { language-no-decl-ref }
820             { \l__zrefclever_ref_language_tl }
821             { \l__zrefclever_ref_decl_case_tl }
822             \tl_clear:N \l__zrefclever_ref_decl_case_tl
823         }
824     }
825     {
826         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
827         {
828             \seq_get_left:NN \l__zrefclever_lang_declension_seq
829             \l__zrefclever_ref_decl_case_tl
830         }
831         {
832             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

833         \l__zrefclever_ref_decl_case_tl
834     {
835         \msg_warning:nxxx { zref-clever }
836         { unknown-decl-case }
837         { \l__zrefclever_ref_decl_case_tl }
838         { \l__zrefclever_ref_language_tl }
839         \seq_get_left:NN \l__zrefclever_lang_declension_seq
840         \l__zrefclever_ref_decl_case_tl
841     }
842 }
843 }

```

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.

```

844     \__zrefclever_opt_seq_get:cNF
845     {
846         \__zrefclever_opt_varname_language:enn
847         { \l__zrefclever_ref_language_tl } { gender } { seq }
848     }
849     \l__zrefclever_lang_gender_seq
850     { \seq_clear:N \l__zrefclever_lang_gender_seq }
851     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
852     {
853         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
854         {
855             \msg_warning:nxxxx { zref-clever }
856             { language-no-gender }
857             { \l__zrefclever_ref_language_tl }
858             { g }
859             { \l__zrefclever_ref_gender_tl }
860             \tl_clear:N \l__zrefclever_ref_gender_tl
861         }
862     }
863     {
864         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
865         {
866             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
867             \l__zrefclever_ref_gender_tl
868             {
869                 \msg_warning:nxxx { zref-clever }
870                 { gender-not-declared }
871                 { \l__zrefclever_ref_language_tl }
872                 { \l__zrefclever_ref_gender_tl }
873                 \tl_clear:N \l__zrefclever_ref_gender_tl
874             }
875         }
876     }

```

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

```

877     \__zrefclever_opt_bool_if:cT
878     {
879         \__zrefclever_opt_varname_language:enn
880         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

881     }
882     { \keys_set:nn { zref-clever/reference } { cap = true } }
883   }
884   {

```

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

```

885     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
886     {
887       \msg_warning:nxxx { zref-clever } { unknown-language-decl }
888       { \l__zrefclever_ref_decl_case_tl }
889       { \l__zrefclever_ref_language_tl }
890       \tl_clear:N \l__zrefclever_ref_decl_case_tl
891     }
892     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
893     {
894       \msg_warning:nxxxx { zref-clever }
895       { language-no-gender }
896       { \l__zrefclever_ref_language_tl }
897       { g }
898       { \l__zrefclever_ref_gender_tl }
899       \tl_clear:N \l__zrefclever_ref_gender_tl
900     }
901   }
902 }

```

(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 `.ltx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read

and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load 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.

```
903 \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}

904 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
905 {
906   \group_begin:
907   \@bsphack
908   \__zrefclever_language_if_declared:nT {#1}
909   {
910     \seq_if_in:NxF
911     \g__zrefclever_loaded_langfiles_seq
912     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
913     {
914       \exp_args:Nx \file_get:nnNTF
915       {
916         zref-clever-
917         \tl_use:c { \__zrefclever_language_varname:n {#1} }
918         .lang
919       }
920       { \ExplSyntaxOn }
921       \l_tmpa_tl
922       {
923         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
924         \tl_clear:N \l__zrefclever_setup_type_tl
925         \__zrefclever_opt_seq_get:cNF
926         {
927           \__zrefclever_opt_varname_language:nnn
928           {#1} { declension } { seq }
929         }
930         \l__zrefclever_lang_declension_seq
931         { \seq_clear:N \l__zrefclever_lang_declension_seq }
932         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
```

```

933         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
934       {
935         \seq_get_left:NN \l__zrefclever_lang_declension_seq
936         \l__zrefclever_lang_decl_case_tl
937       }
938     \__zrefclever_opt_seq_get:cNF
939     {
940       \__zrefclever_opt_varname_language:nnn
941       {#1} { gender } { seq }
942     }
943     \l__zrefclever_lang_gender_seq
944     { \seq_clear:N \l__zrefclever_lang_gender_seq }
945     \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
946     \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
947     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
948     \msg_info:nnx { zref-clever } { langfile-loaded }
949     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
950   }
951   {

```

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.

```

952         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
953         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
954       }
955     }
956   }
957   \@esphack
958   \group_end:
959 }
960 \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.

```

961 \keys_define:nn { zref-clever/langfile }
962 {
963   type .code:n =
964   {
965     \tl_if_empty:nTF {#1}
966     { \tl_clear:N \l__zrefclever_setup_type_tl }
967     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
968   } ,
969
970   case .code:n =
971   {
972     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
973     {
974       \msg_info:nnxx { zref-clever } { language-no-decl-setup }
975       { \l__zrefclever_setup_language_tl } {#1}

```

```

976     }
977     {
978         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
979         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
980         {
981             \msg_info:nxxx { zref-clever } { unknown-decl-case }
982             {#1} { \l__zrefclever_setup_language_tl }
983             \seq_get_left:NN \l__zrefclever_lang_declension_seq
984             \l__zrefclever_lang_decl_case_tl
985         }
986     }
987 },
988 case .value_required:n = true ,
989
990 gender .value_required:n = true ,
991 gender .code:n =
992 {
993     \seq_if_empty:NnTF \l__zrefclever_lang_gender_seq
994     {
995         \msg_info:nxxxx { zref-clever } { language-no-gender }
996         { \l__zrefclever_setup_language_tl } { gender } {#1}
997     }
998     {
999         \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1000         {
1001             \msg_info:nnn { zref-clever }
1002             { option-only-type-specific } { gender }
1003         }
1004         {
1005             \seq_clear:N \l_tmpa_seq
1006             \clist_map_inline:nn {#1}
1007             {
1008                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1009                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
1010                 {
1011                     \msg_info:nxxx { zref-clever }
1012                     { gender-not-declared }
1013                     { \l__zrefclever_setup_language_tl } {##1}
1014                 }
1015             }
1016             \__zrefclever_opt_seq_if_set:cF
1017             {
1018                 \__zrefclever_opt_varname_lang_type:eenn
1019                 { \l__zrefclever_setup_language_tl }
1020                 { \l__zrefclever_setup_type_tl }
1021                 { gender }
1022                 { seq }
1023             }
1024             {
1025                 \seq_new:c
1026                 {
1027                     \__zrefclever_opt_varname_lang_type:eenn
1028                     { \l__zrefclever_setup_language_tl }
1029                     { \l__zrefclever_setup_type_tl }

```



```

1030         { gender }
1031         { seq }
1032     }
1033     \seq_gset_eq:cN
1034     {
1035         \__zrefclever_opt_varname_lang_type:eenn
1036         { \l__zrefclever_setup_language_tl }
1037         { \l__zrefclever_setup_type_tl }
1038         { gender }
1039         { seq }
1040     }
1041     \l_tmpa_seq
1042 }
1043 }
1044 }
1045 },
1046 }
1047 \seq_map_inline:Nn
1048 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1049 {
1050     \keys_define:nn { zref-clever/langfile }
1051     {
1052         #1 .value_required:n = true ,
1053         #1 .code:n =
1054         {
1055             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1056             {
1057                 \__zrefclever_opt_tl_gset_if_new:cn
1058                 {
1059                     \__zrefclever_opt_varname_lang_default:enn
1060                     { \l__zrefclever_setup_language_tl }
1061                     {#1} { tl }
1062                 }
1063                 {##1}
1064             }
1065             {
1066                 \msg_info:nnn { zref-clever }
1067                 { option-not-type-specific } {#1}
1068             }
1069         },
1070     }
1071 }
1072 \seq_map_inline:Nn
1073 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1074 {
1075     \keys_define:nn { zref-clever/langfile }
1076     {
1077         #1 .value_required:n = true ,
1078         #1 .code:n =
1079         {
1080             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1081             {
1082                 \__zrefclever_opt_tl_gset_if_new:cn
1083                 {

```

```

1084         \_zrefclever_opt_varname_lang_default:enn
1085         { \l_zrefclever_setup_language_tl }
1086         {#1} { t1 }
1087     }
1088     {##1}
1089 }
1090 {
1091     \_zrefclever_opt_tl_gset_if_new:cn
1092     {
1093         \_zrefclever_opt_varname_lang_type:enn
1094         { \l_zrefclever_setup_language_tl }
1095         { \l_zrefclever_setup_type_tl }
1096         {#1} { t1 }
1097     }
1098     {##1}
1099 }
1100 },
1101 }
1102 }
1103 \keys_define:nn { zref-clever/langfile }
1104 {
1105     endrange .value_required:n = true ,
1106     endrange .code:n =
1107     {
1108         \str_case:nnF {#1}
1109         {
1110             { ref }
1111             {
1112                 \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1113                 {
1114                     \_zrefclever_opt_tl_gclear_if_new:c
1115                     {
1116                         \_zrefclever_opt_varname_lang_default:enn
1117                         { \l_zrefclever_setup_language_tl }
1118                         { endrangefunc } { t1 }
1119                     }
1120                     \_zrefclever_opt_tl_gclear_if_new:c
1121                     {
1122                         \_zrefclever_opt_varname_lang_default:enn
1123                         { \l_zrefclever_setup_language_tl }
1124                         { endrangeprop } { t1 }
1125                     }
1126                 }
1127             }
1128             \_zrefclever_opt_tl_gclear_if_new:c
1129             {
1130                 \_zrefclever_opt_varname_lang_type:enn
1131                 { \l_zrefclever_setup_language_tl }
1132                 { \l_zrefclever_setup_type_tl }
1133                 { endrangefunc } { t1 }
1134             }
1135             \_zrefclever_opt_tl_gclear_if_new:c
1136             {
1137                 \_zrefclever_opt_varname_lang_type:enn

```

```

1138         { \l__zrefclever_setup_language_tl }
1139         { \l__zrefclever_setup_type_tl }
1140         { endrangeprop } { tl }
1141     }
1142 }
1143 }
1144
1145 { stripprefix }
1146 {
1147     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1148     {
1149         \__zrefclever_opt_tl_gset_if_new:cn
1150         {
1151             \__zrefclever_opt_varname_lang_default:enn
1152             { \l__zrefclever_setup_language_tl }
1153             { endrangefunc } { tl }
1154         }
1155         { __zrefclever_get_endrange_stripprefix }
1156         \__zrefclever_opt_tl_gclear_if_new:c
1157         {
1158             \__zrefclever_opt_varname_lang_default:enn
1159             { \l__zrefclever_setup_language_tl }
1160             { endrangeprop } { tl }
1161         }
1162     }
1163     {
1164         \__zrefclever_opt_tl_gset_if_new:cn
1165         {
1166             \__zrefclever_opt_varname_lang_type:enn
1167             { \l__zrefclever_setup_language_tl }
1168             { \l__zrefclever_setup_type_tl }
1169             { endrangefunc } { tl }
1170         }
1171         { __zrefclever_get_endrange_stripprefix }
1172         \__zrefclever_opt_tl_gclear_if_new:c
1173         {
1174             \__zrefclever_opt_varname_lang_type:enn
1175             { \l__zrefclever_setup_language_tl }
1176             { \l__zrefclever_setup_type_tl }
1177             { endrangeprop } { tl }
1178         }
1179     }
1180 }
1181
1182 { pagecomp }
1183 {
1184     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1185     {
1186         \__zrefclever_opt_tl_gset_if_new:cn
1187         {
1188             \__zrefclever_opt_varname_lang_default:enn
1189             { \l__zrefclever_setup_language_tl }
1190             { endrangefunc } { tl }
1191         }

```

```

1192         { __zrefclever_get_endrange_pagecomp }
1193     \__zrefclever_opt_tl_gclear_if_new:c
1194     {
1195         \__zrefclever_opt_varname_lang_default:enn
1196         { \l__zrefclever_setup_language_tl }
1197         { endrangeprop } { t1 }
1198     }
1199 }
1200 {
1201     \__zrefclever_opt_tl_gset_if_new:cn
1202     {
1203         \__zrefclever_opt_varname_lang_type:een
1204         { \l__zrefclever_setup_language_tl }
1205         { \l__zrefclever_setup_type_tl }
1206         { endrangefunc } { t1 }
1207     }
1208     { __zrefclever_get_endrange_pagecomp }
1209     \__zrefclever_opt_tl_gclear_if_new:c
1210     {
1211         \__zrefclever_opt_varname_lang_type:een
1212         { \l__zrefclever_setup_language_tl }
1213         { \l__zrefclever_setup_type_tl }
1214         { endrangeprop } { t1 }
1215     }
1216 }
1217 }
1218
1219 { pagecomp2 }
1220 {
1221     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1222     {
1223         \__zrefclever_opt_tl_gset_if_new:cn
1224         {
1225             \__zrefclever_opt_varname_lang_default:enn
1226             { \l__zrefclever_setup_language_tl }
1227             { endrangefunc } { t1 }
1228         }
1229         { __zrefclever_get_endrange_pagecomptwo }
1230         \__zrefclever_opt_tl_gclear_if_new:c
1231         {
1232             \__zrefclever_opt_varname_lang_default:enn
1233             { \l__zrefclever_setup_language_tl }
1234             { endrangeprop } { t1 }
1235         }
1236     }
1237     {
1238         \__zrefclever_opt_tl_gset_if_new:cn
1239         {
1240             \__zrefclever_opt_varname_lang_type:een
1241             { \l__zrefclever_setup_language_tl }
1242             { \l__zrefclever_setup_type_tl }
1243             { endrangefunc } { t1 }
1244         }
1245         { __zrefclever_get_endrange_pagecomptwo }

```

```

1246         \__zrefclever_opt_tl_gclear_if_new:c
1247         {
1248             \__zrefclever_opt_varname_lang_type:enn
1249             { \l__zrefclever_setup_language_tl }
1250             { \l__zrefclever_setup_type_tl }
1251             { endrangeprop } { tl }
1252         }
1253     }
1254 }
1255 }
1256 {
1257     \tl_if_empty:nTF {#1}
1258     {
1259         \msg_info:nnn { zref-clever }
1260         { endrange-property-undefined } {#1}
1261     }
1262     {
1263         \zref@ifpropundefined {#1}
1264         {
1265             \msg_info:nnn { zref-clever }
1266             { endrange-property-undefined } {#1}
1267         }
1268         {
1269             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1270             {
1271                 \__zrefclever_opt_tl_gset_if_new:cn
1272                 {
1273                     \__zrefclever_opt_varname_lang_default:enn
1274                     { \l__zrefclever_setup_language_tl }
1275                     { endrangefunc } { tl }
1276                 }
1277                 { __zrefclever_get_endrange_property }
1278                 \__zrefclever_opt_tl_gset_if_new:cn
1279                 {
1280                     \__zrefclever_opt_varname_lang_default:enn
1281                     { \l__zrefclever_setup_language_tl }
1282                     { endrangeprop } { tl }
1283                 }
1284                 {#1}
1285             }
1286             {
1287                 \__zrefclever_opt_tl_gset_if_new:cn
1288                 {
1289                     \__zrefclever_opt_varname_lang_type:enn
1290                     { \l__zrefclever_setup_language_tl }
1291                     { \l__zrefclever_setup_type_tl }
1292                     { endrangefunc } { tl }
1293                 }
1294                 { __zrefclever_get_endrange_property }
1295                 \__zrefclever_opt_tl_gset_if_new:cn
1296                 {
1297                     \__zrefclever_opt_varname_lang_type:enn
1298                     { \l__zrefclever_setup_language_tl }
1299                     { \l__zrefclever_setup_type_tl }

```

```

1300             { endrangeprop } { t1 }
1301         }
1302         {#1}
1303     }
1304 }
1305 }
1306 }
1307 },
1308 }
1309 \seq_map_inline:Nn
1310 \g__zrefclever_rf_opts_tl_type_names_seq
1311 {
1312     \keys_define:nn { zref-clever/langfile }
1313     {
1314         #1 .value_required:n = true ,
1315         #1 .code:n =
1316         {
1317             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1318             {
1319                 \msg_info:nnn { zref-clever }
1320                 { option-only-type-specific } {#1}
1321             }
1322             {
1323                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1324                 {
1325                     \__zrefclever_opt_tl_gset_if_new:cn
1326                     {
1327                         \__zrefclever_opt_varname_lang_type:een
1328                         { \l__zrefclever_setup_language_tl }
1329                         { \l__zrefclever_setup_type_tl }
1330                         {#1} { t1 }
1331                     }
1332                     {##1}
1333                 }
1334                 {
1335                     \__zrefclever_opt_tl_gset_if_new:cn
1336                     {
1337                         \__zrefclever_opt_varname_lang_type:een
1338                         { \l__zrefclever_setup_language_tl }
1339                         { \l__zrefclever_setup_type_tl }
1340                         { \l__zrefclever_lang_decl_case_tl - #1 } { t1 }
1341                     }
1342                     {##1}
1343                 }
1344             }
1345         },
1346     }
1347 }
1348 \seq_map_inline:Nn
1349 \g__zrefclever_rf_opts_seq_refbounds_seq
1350 {
1351     \keys_define:nn { zref-clever/langfile }
1352     {
1353         #1 .value_required:n = true ,

```

```

1354 #1 .code:n =
1355 {
1356   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1357   {
1358     \__zrefclever_opt_seq_if_set:cF
1359     {
1360       \__zrefclever_opt_varname_lang_default:enn
1361       { \l__zrefclever_setup_language_tl } {#1} { seq }
1362     }
1363     {
1364       \seq_gclear:N \g_tmpa_seq
1365       \__zrefclever_opt_seq_gset_clist_split:Nn
1366       \g_tmpa_seq {##1}
1367       \bool_lazy_or:nnTF
1368       { \tl_if_empty_p:n {##1} }
1369       {
1370         \int_compare_p:nNn
1371         { \seq_count:N \g_tmpa_seq } = { 4 }
1372       }
1373       {
1374         \__zrefclever_opt_seq_gset_eq:cN
1375         {
1376           \__zrefclever_opt_varname_lang_default:enn
1377           { \l__zrefclever_setup_language_tl }
1378           {#1} { seq }
1379         }
1380         \g_tmpa_seq
1381       }
1382       {
1383         \msg_info:nxxx { zref-clever }
1384         { refbounds-must-be-four }
1385         {#1} { \seq_count:N \g_tmpa_seq }
1386       }
1387     }
1388   }
1389   {
1390     \__zrefclever_opt_seq_if_set:cF
1391     {
1392       \__zrefclever_opt_varname_lang_type:eenn
1393       { \l__zrefclever_setup_language_tl }
1394       { \l__zrefclever_setup_type_tl } {#1} { seq }
1395     }
1396     {
1397       \seq_gclear:N \g_tmpa_seq
1398       \__zrefclever_opt_seq_gset_clist_split:Nn
1399       \g_tmpa_seq {##1}
1400       \bool_lazy_or:nnTF
1401       { \tl_if_empty_p:n {##1} }
1402       {
1403         \int_compare_p:nNn
1404         { \seq_count:N \g_tmpa_seq } = { 4 }
1405       }
1406       {
1407         \__zrefclever_opt_seq_gset_eq:cN

```

```

1408         {
1409             \__zrefclever_opt_varname_lang_type:eenn
1410             { \l__zrefclever_setup_language_tl }
1411             { \l__zrefclever_setup_type_tl }
1412             {#1} { seq }
1413         }
1414         \g_tmpa_seq
1415     }
1416     {
1417         \msg_info:nxxx { zref-clever }
1418         { refbounds-must-be-four }
1419         {#1} { \seq_count:N \g_tmpa_seq }
1420     }
1421 }
1422 }
1423 } ,
1424 }
1425 }
1426 \seq_map_inline:Nn
1427 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1428 {
1429     \keys_define:nn { zref-clever/langfile }
1430     {
1431         #1 .choice: ,
1432         #1 / true .code:n =
1433         {
1434             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1435             {
1436                 \__zrefclever_opt_bool_if_set:cF
1437                 {
1438                     \__zrefclever_opt_varname_lang_default:enn
1439                     { \l__zrefclever_setup_language_tl }
1440                     {#1} { bool }
1441                 }
1442                 {
1443                     \__zrefclever_opt_bool_gset_true:c
1444                     {
1445                         \__zrefclever_opt_varname_lang_default:enn
1446                         { \l__zrefclever_setup_language_tl }
1447                         {#1} { bool }
1448                     }
1449                 }
1450             }
1451         }
1452         \__zrefclever_opt_bool_if_set:cF
1453         {
1454             \__zrefclever_opt_varname_lang_type:eenn
1455             { \l__zrefclever_setup_language_tl }
1456             { \l__zrefclever_setup_type_tl }
1457             {#1} { bool }
1458         }
1459         {
1460             \__zrefclever_opt_bool_gset_true:c
1461             {

```



```

1462         \_zrefclever_opt_varname_lang_type:eenn
1463         { \l__zrefclever_setup_language_tl }
1464         { \l__zrefclever_setup_type_tl }
1465         {#1} { bool }
1466     }
1467 }
1468 }
1469 },
1470 #1 / false .code:n =
1471 {
1472     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1473     {
1474         \_zrefclever_opt_bool_if_set:cF
1475         {
1476             \_zrefclever_opt_varname_lang_default:enn
1477             { \l__zrefclever_setup_language_tl }
1478             {#1} { bool }
1479         }
1480         {
1481             \_zrefclever_opt_bool_gset_false:c
1482             {
1483                 \_zrefclever_opt_varname_lang_default:enn
1484                 { \l__zrefclever_setup_language_tl }
1485                 {#1} { bool }
1486             }
1487         }
1488     }
1489     {
1490         \_zrefclever_opt_bool_if_set:cF
1491         {
1492             \_zrefclever_opt_varname_lang_type:eenn
1493             { \l__zrefclever_setup_language_tl }
1494             { \l__zrefclever_setup_type_tl }
1495             {#1} { bool }
1496         }
1497         {
1498             \_zrefclever_opt_bool_gset_false:c
1499             {
1500                 \_zrefclever_opt_varname_lang_type:eenn
1501                 { \l__zrefclever_setup_language_tl }
1502                 { \l__zrefclever_setup_type_tl }
1503                 {#1} { bool }
1504             }
1505         }
1506     }
1507 },
1508 #1 .default:n = true ,
1509 no #1 .meta:n = { #1 = false } ,
1510 no #1 .value_forbidden:n = true ,
1511 }
1512 }

```

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.

```

1513 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1514 {
1515   \tl_const:cn
1516   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1517 }
1518 \keyval_parse:nnn
1519 { }
1520 { \__zrefclever_opt_tl_cset_fallback:nn }
1521 {
1522   tpairsep = {,~} ,
1523   tlistsep = {,~} ,
1524   tlastsep = {,~} ,
1525   notesep  = {~} ,
1526   namesep  = {\nobreakspace} ,
1527   pairsep  = {,~} ,
1528   listsep  = {,~} ,
1529   lastsep  = {,~} ,
1530   rangesep = {\textendash} ,
1531 }

```

## 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 }

1532 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1533 {
1534   \tl_if_empty:nTF {#3}
1535   { \prop_remove:Nn #1 {#2} }
1536   { \prop_put:Nnn #1 {#2} {#3} }
1537 }

```

(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`.

```

1538 \tl_new:N \l__zrefclever_ref_property_tl
1539 \keys_define:nn { zref-clever/reference }
1540 {
1541   ref .code:n =
1542   {
1543     \tl_if_empty:nTF {#1}
1544     {
1545       \msg_warning:nnn { zref-clever }
1546       { zref-property-undefined } {#1}
1547       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1548     }
1549     {
1550       \zref@ifpropundefined {#1}
1551       {
1552         \msg_warning:nnn { zref-clever }
1553         { zref-property-undefined } {#1}
1554         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1555       }
1556       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1557     }
1558   } ,
1559   ref .initial:n = default ,
1560   ref .value_required:n = true ,
1561   page .meta:n = { ref = page } ,
1562   page .value_forbidden:n = true ,
1563 }

```

### typeset option

```

1564 \bool_new:N \l__zrefclever_typeset_ref_bool
1565 \bool_new:N \l__zrefclever_typeset_name_bool
1566 \keys_define:nn { zref-clever/reference }
1567 {
1568   typeset .choice: ,
1569   typeset / both .code:n =
1570   {
1571     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1572     \bool_set_true:N \l__zrefclever_typeset_name_bool
1573   } ,
1574   typeset / ref .code:n =
1575   {
1576     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1577     \bool_set_false:N \l__zrefclever_typeset_name_bool
1578   } ,
1579   typeset / name .code:n =
1580   {
1581     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1582     \bool_set_true:N \l__zrefclever_typeset_name_bool
1583   } ,
1584   typeset .initial:n = both ,
1585   typeset .value_required:n = true ,
1586
1587   noname .meta:n = { typeset = ref } ,
1588   noname .value_forbidden:n = true ,

```

```

1589     noref .meta:n = { typeset = name } ,
1590     noref .value_forbidden:n = true ,
1591 }

```

#### sort option

```

1592 \bool_new:N \l__zrefclever_typeset_sort_bool
1593 \keys_define:nn { zref-clever/reference }
1594 {
1595     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1596     sort .initial:n = true ,
1597     sort .default:n = true ,
1598     nosort .meta:n = { sort = false },
1599     nosort .value_forbidden:n = true ,
1600 }

```

#### 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`.

```

1601 \seq_new:N \l__zrefclever_typesort_seq
1602 \keys_define:nn { zref-clever/reference }
1603 {
1604     typesort .code:n =
1605     {
1606         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1607         \seq_reverse:N \l__zrefclever_typesort_seq
1608     } ,
1609     typesort .initial:n =
1610     { part , chapter , section , paragraph },
1611     typesort .value_required:n = true ,
1612     notypesort .code:n =
1613     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1614     notypesort .value_forbidden:n = true ,
1615 }

```

#### comp option

```

1616 \bool_new:N \l__zrefclever_typeset_compress_bool
1617 \keys_define:nn { zref-clever/reference }
1618 {
1619     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1620     comp .initial:n = true ,
1621     comp .default:n = true ,
1622     nocomp .meta:n = { comp = false },
1623     nocomp .value_forbidden:n = true ,
1624 }

```

#### 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`.

```

1625 \NewHook { zref-clever/endrange-setup }
1626 \keys_define:nn { zref-clever/reference }
1627 {
1628   endrange .code:n =
1629   {
1630     \str_case:nnF {#1}
1631     {
1632       { ref }
1633       {
1634         \_zrefclever_opt_tl_clear:c
1635         {
1636           \_zrefclever_opt_varname_general:nn
1637           { endrangefunc } { tl }
1638         }
1639         \_zrefclever_opt_tl_clear:c
1640         {
1641           \_zrefclever_opt_varname_general:nn
1642           { endrangeprop } { tl }
1643         }
1644       }
1645     }
1646     { stripprefix }
1647     {

```

```

1648 \_zrefclever_opt_tl_set:cn
1649 {
1650   \_zrefclever_opt_varname_general:nn
1651   { endrangefunc } { t1 }
1652 }
1653 { \_zrefclever_get_endrange_stripprefix }
1654 \_zrefclever_opt_tl_clear:c
1655 {
1656   \_zrefclever_opt_varname_general:nn
1657   { endrangeprop } { t1 }
1658 }
1659 }
1660
1661 { pagecomp }
1662 {
1663   \_zrefclever_opt_tl_set:cn
1664   {
1665     \_zrefclever_opt_varname_general:nn
1666     { endrangefunc } { t1 }
1667   }
1668   { \_zrefclever_get_endrange_pagecomp }
1669   \_zrefclever_opt_tl_clear:c
1670   {
1671     \_zrefclever_opt_varname_general:nn
1672     { endrangeprop } { t1 }
1673   }
1674 }
1675
1676 { pagecomp2 }
1677 {
1678   \_zrefclever_opt_tl_set:cn
1679   {
1680     \_zrefclever_opt_varname_general:nn
1681     { endrangefunc } { t1 }
1682   }
1683   { \_zrefclever_get_endrange_pagecomptwo }
1684   \_zrefclever_opt_tl_clear:c
1685   {
1686     \_zrefclever_opt_varname_general:nn
1687     { endrangeprop } { t1 }
1688   }
1689 }
1690
1691 { unset }
1692 {
1693   \_zrefclever_opt_tl_unset:c
1694   {
1695     \_zrefclever_opt_varname_general:nn
1696     { endrangefunc } { t1 }
1697   }
1698   \_zrefclever_opt_tl_unset:c
1699   {
1700     \_zrefclever_opt_varname_general:nn
1701     { endrangeprop } { t1 }

```

```

1702     }
1703   }
1704 }
1705 {
1706   \tl_if_empty:nTF {#1}
1707   {
1708     \msg_warning:nnn { zref-clever }
1709     { endrange-property-undefined } {#1}
1710   }
1711   {
1712     \zref@ifpropundefined {#1}
1713     {
1714       \msg_warning:nnn { zref-clever }
1715       { endrange-property-undefined } {#1}
1716     }
1717     {
1718       \__zrefclever_opt_tl_set:cn
1719       {
1720         \__zrefclever_opt_varname_general:nn
1721         { endrangefunc } { tl }
1722       }
1723       { __zrefclever_get_endrange_property }
1724       \__zrefclever_opt_tl_set:cn
1725       {
1726         \__zrefclever_opt_varname_general:nn
1727         { endrangeprop } { tl }
1728       }
1729       {#1}
1730     }
1731   }
1732 }
1733 },
1734 endrange .value_required:n = true ,
1735 }
1736 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1737 {
1738   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1739   {
1740     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1741     {
1742       \__zrefclever_extract_default:Nnvn #3
1743       {#2} { \l__zrefclever_ref_property_tl } { }
1744     }
1745     { \tl_set:Nn #3 { zc@missingproperty } }
1746   }
1747   {
1748     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1749     {

```

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.

```

1750 \bool_if:NTF \l__zrefclever_typeset_range_bool
1751 {
1752   \group_begin:
1753   \bool_set_false:N \l_tmpa_bool
1754   \exp_args:Nxx \tl_if_eq:nnT
1755   {
1756     \__zrefclever_extract_unexp:nnn
1757     {#1} { externaldocument } { }
1758   }
1759   {
1760     \__zrefclever_extract_unexp:nnn
1761     {#2} { externaldocument } { }
1762   }
1763   {
1764     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1765     {
1766       \exp_args:Nxx \tl_if_eq:nnT
1767       {
1768         \__zrefclever_extract_unexp:nnn
1769         {#1} { zc@pgfmt } { }
1770       }
1771       {
1772         \__zrefclever_extract_unexp:nnn
1773         {#2} { zc@pgfmt } { }
1774       }
1775       { \bool_set_true:N \l_tmpa_bool }
1776     }
1777     {
1778       \exp_args:Nxx \tl_if_eq:nnT
1779       {
1780         \__zrefclever_extract_unexp:nnn
1781         {#1} { zc@counter } { }
1782       }
1783       {
1784         \__zrefclever_extract_unexp:nnn
1785         {#2} { zc@counter } { }
1786       }
1787       {
1788         \exp_args:Nxx \tl_if_eq:nnT
1789         {
1790           \__zrefclever_extract_unexp:nnn
1791           {#1} { zc@enclval } { }
1792         }
1793         {
1794           \__zrefclever_extract_unexp:nnn
1795           {#2} { zc@enclval } { }
1796         }
1797         { \bool_set_true:N \l_tmpa_bool }
1798       }
1799     }
1800   }
1801   \bool_if:NTF \l_tmpa_bool
1802   {
1803     \__zrefclever_extract_default:Nnvn \l_tmpb_tl

```



```

1804         {#2} { l__zrefclever_endrangeprop_tl } { }
1805     }
1806     {
1807         \zref@ifrefcontainsprop
1808         {#2} { \l__zrefclever_ref_property_tl }
1809         {
1810             \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1811             {#2} { l__zrefclever_ref_property_tl } { }
1812         }
1813         { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1814     }
1815     \exp_args:NNNV
1816     \group_end:
1817     \tl_set:Nn #3 \l_tmpb_tl
1818 }
1819 {
1820     \__zrefclever_extract_default:Nnvn #3
1821     {#2} { l__zrefclever_endrangeprop_tl } { }
1822 }
1823 }
1824 {
1825     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1826     {
1827         \__zrefclever_extract_default:Nnvn #3
1828         {#2} { l__zrefclever_ref_property_tl } { }
1829     }
1830     { \tl_set:Nn #3 { zc@missingproperty } }
1831 }
1832 }
1833 }
1834 \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>.

```

1835 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1836 {
1837     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1838     {
1839         \group_begin:
1840         \UseHook { zref-clever/endrange-setup }
1841         \tl_set:Nx \l_tmpa_tl
1842         {
1843             \__zrefclever_extract:nnn
1844             {#1} { \l__zrefclever_ref_property_tl } { }
1845         }
1846         \tl_set:Nx \l_tmpb_tl
1847         {
1848             \__zrefclever_extract:nnn
1849             {#2} { \l__zrefclever_ref_property_tl } { }
1850         }
1851         \bool_set_false:N \l_tmpa_bool
1852         \bool_until_do:Nn \l_tmpa_bool
1853         {
1854             \exp_args:Nxx \tl_if_eq:nnTF

```

```

1855         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1856         {
1857             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1858             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1859             \tl_if_empty:NT \l_tmpb_tl
1860                 { \bool_set_true:N \l_tmpa_bool }
1861         }
1862         { \bool_set_true:N \l_tmpa_bool }
1863     }
1864     \exp_args:NNNV
1865     \group_end:
1866     \tl_set:Nn #3 \l_tmpb_tl
1867 }
1868 { \tl_set:Nn #3 { zc@missingproperty } }
1869 }
1870 \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>).

```

1871 \prg_new_protected_conditional:Npnn
1872   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1873 {
1874     \regex_match:nnTF { \A\d+\Z } {#1}
1875     { \prg_return_true: }
1876     { \prg_return_false: }
1877 }
1878 \prg_generate_conditional_variant:Nnn
1879   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End definition for `\__zrefclever_is_integer_rgx:n`.)

```

1880 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1881 {
1882     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1883     {
1884         \group_begin:
1885         \UseHook { zref-clever/endrange-setup }
1886         \tl_set:Nx \l_tmpa_tl
1887         {
1888             \__zrefclever_extract:nnn
1889             {#1} { \l__zrefclever_ref_property_tl } { }
1890         }
1891         \tl_set:Nx \l_tmpb_tl
1892         {
1893             \__zrefclever_extract:nnn
1894             {#2} { \l__zrefclever_ref_property_tl } { }
1895         }
1896         \bool_set_false:N \l_tmpa_bool
1897         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1898         {
1899             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1900             { \bool_set_true:N \l_tmpa_bool }
1901         }
1902         { \bool_set_true:N \l_tmpa_bool }
1903         \bool_until_do:Nn \l_tmpa_bool

```

```

1904     {
1905         \exp_args:Nxx \tl_if_eq:nnTF
1906         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1907         {
1908             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1909             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1910             \tl_if_empty:NT \l_tmpb_tl
1911             { \bool_set_true:N \l_tmpa_bool }
1912         }
1913         { \bool_set_true:N \l_tmpa_bool }
1914     }
1915     \exp_args:NNNV
1916     \group_end:
1917     \tl_set:Nn #3 \l_tmpb_tl
1918 }
1919 { \tl_set:Nn #3 { zc@missingproperty } }
1920 }
1921 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1922 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1923 {
1924     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1925     {
1926         \group_begin:
1927         \UseHook { zref-clever/endrange-setup }
1928         \tl_set:Nx \l_tmpa_tl
1929         {
1930             \__zrefclever_extract:nnn
1931             {#1} { \l__zrefclever_ref_property_tl } { }
1932         }
1933         \tl_set:Nx \l_tmpb_tl
1934         {
1935             \__zrefclever_extract:nnn
1936             {#2} { \l__zrefclever_ref_property_tl } { }
1937         }
1938         \bool_set_false:N \l_tmpa_bool
1939         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1940         {
1941             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1942             { \bool_set_true:N \l_tmpa_bool }
1943         }
1944         { \bool_set_true:N \l_tmpa_bool }
1945         \bool_until_do:Nn \l_tmpa_bool
1946         {
1947             \exp_args:Nxx \tl_if_eq:nnTF
1948             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1949             {
1950                 \bool_lazy_or:nnTF
1951                 { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1952                 { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1953                 {
1954                     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1955                     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1956                 }
1957                 { \bool_set_true:N \l_tmpa_bool }

```

```

1958         }
1959         { \bool_set_true:N \l_tmpa_bool }
1960     }
1961     \exp_args:NNNV
1962     \group_end:
1963     \tl_set:Nn #3 \l_tmpb_tl
1964 }
1965 { \tl_set:Nn #3 { zc@missingproperty } }
1966 }
1967 \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`.

```

1968 \bool_new:N \l__zrefclever_typeset_range_bool
1969 \keys_define:nn { zref-clever/reference }
1970 {
1971     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1972     range .initial:n = false ,
1973     range .default:n = true ,
1974 }

```

### 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`.

```

1975 \bool_new:N \l__zrefclever_capfirst_bool
1976 \keys_define:nn { zref-clever/reference }
1977 {
1978     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1979     capfirst .initial:n = false ,
1980     capfirst .default:n = true ,
1981 }

```

### 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`.

```

1982 \bool_new:N \l__zrefclever_noabbrev_first_bool
1983 \keys_define:nn { zref-clever/reference }
1984 {
1985     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1986     noabbrevfirst .initial:n = false ,
1987     noabbrevfirst .default:n = true ,
1988 }

```

### S option

```

1989 \keys_define:nn { zref-clever/reference }
1990 {
1991     S .meta:n =

```

```

1992     { capfirst = {#1} , noabbrevfirst = {#1} },
1993     S .default:n = true ,
1994 }

```

### hyperref option

```

1995 \bool_new:N \l__zrefclever_hyperlink_bool
1996 \bool_new:N \l__zrefclever_hyperref_warn_bool
1997 \keys_define:nn { zref-clever/reference }
1998 {
1999     hyperref .choice: ,
2000     hyperref / auto .code:n =
2001     {
2002         \bool_set_true:N \l__zrefclever_hyperlink_bool
2003         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2004     } ,
2005     hyperref / true .code:n =
2006     {
2007         \bool_set_true:N \l__zrefclever_hyperlink_bool
2008         \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2009     } ,
2010     hyperref / false .code:n =
2011     {
2012         \bool_set_false:N \l__zrefclever_hyperlink_bool
2013         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2014     } ,
2015     hyperref .initial:n = auto ,
2016     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.

```

2017     nohyperref .meta:n = { hyperref = false } ,
2018     nohyperref .value_forbidden:n = true ,
2019 }
2020 \AddToHook { begindocument }
2021 {
2022     \__zrefclever_if_package_loaded:nTF { hyperref }
2023     {
2024         \bool_if:NT \l__zrefclever_hyperlink_bool
2025         { \RequirePackage { zref-hyperref } }
2026     }
2027     {
2028         \bool_if:NT \l__zrefclever_hyperref_warn_bool
2029         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2030         \bool_set_false:N \l__zrefclever_hyperlink_bool
2031     }
2032     \keys_define:nn { zref-clever/reference }
2033     {
2034         hyperref .code:n =
2035         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2036         nohyperref .code:n =
2037         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,

```

```

2038     }
2039 }

```

### nameinlink option

```

2040 \str_new:N \l__zrefclever_nameinlink_str
2041 \keys_define:nn { zref-clever/reference }
2042 {
2043   nameinlink .choice: ,
2044   nameinlink / true .code:n =
2045     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2046   nameinlink / false .code:n =
2047     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2048   nameinlink / single .code:n =
2049     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2050   nameinlink / tsingle .code:n =
2051     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2052   nameinlink .initial:n = tsingle ,
2053   nameinlink .default:n = true ,
2054 }

```

### preposinlink option (deprecated)

```

2055 \keys_define:nn { zref-clever/reference }
2056 {
2057   preposinlink .code:n =
2058     {
2059       % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2060       \msg_warning:nnnn { zref-clever }{ option-deprecated }
2061       { preposinlink } { refbounds }
2062     } ,
2063 }

```

### 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`.

```

2064 \AddToHook { begindocument }
2065 {

```

```

2066 \__zrefclever_if_package_loaded:nTF { babel }
2067 {
2068   \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
2069   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2070 }
2071 {
2072   \__zrefclever_if_package_loaded:nTF { polyglossia }
2073   {
2074     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2075     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2076   }
2077   {
2078     \tl_set:Nn \l__zrefclever_current_language_tl { english }
2079     \tl_set:Nn \l__zrefclever_main_language_tl { english }
2080   }
2081 }
2082 }
2083 \keys_define:nn { zref-clever/reference }
2084 {
2085   lang .code:n =
2086   {
2087     \AddToHook { begindocument }
2088     {
2089       \str_case:nnF {#1}
2090       {
2091         { current }
2092         {
2093           \tl_set:Nn \l__zrefclever_ref_language_tl
2094             { \l__zrefclever_current_language_tl }
2095         }
2096
2097         { main }
2098         {
2099           \tl_set:Nn \l__zrefclever_ref_language_tl
2100             { \l__zrefclever_main_language_tl }
2101         }
2102       }
2103       {
2104         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2105         \__zrefclever_language_if_declared:nF {#1}
2106         {
2107           \msg_warning:nnn { zref-clever }
2108             { unknown-language-opt } {#1}
2109         }
2110       }
2111       \__zrefclever_provide_langfile:x
2112         { \l__zrefclever_ref_language_tl }
2113     }
2114   } ,
2115   lang .initial:n = current ,
2116   lang .value_required:n = true ,
2117 }
2118 \AddToHook { begindocument / before }

```

```

2119 {
2120   \AddToHook { begindocument }
2121   {

```

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.

```

2122     \keys_define:nn { zref-clever/reference }
2123     {
2124       lang .code:n =
2125       {
2126         \str_case:nnF {#1}
2127         {
2128           { current }
2129           {
2130             \tl_set:Nn \l__zrefclever_ref_language_tl
2131             { \l__zrefclever_current_language_tl }
2132           }
2133
2134           { main }
2135           {
2136             \tl_set:Nn \l__zrefclever_ref_language_tl
2137             { \l__zrefclever_main_language_tl }
2138           }
2139         }
2140         {
2141           \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2142           \__zrefclever_language_if_declared:nF {#1}
2143           {
2144             \msg_warning:nnn { zref-clever }
2145             { unknown-language-opt } {#1}
2146           }
2147         }
2148       } ,
2149     }
2150   }
2151 }

```

### 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.

```

2152 \tl_new:N \l__zrefclever_ref_decl_case_tl
2153 \keys_define:nn { zref-clever/reference }
2154 {
2155   d .code:n =
2156   { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2157 }
2158 \AddToHook { begindocument }

```



```

2159 {
2160   \keys_define:nn { zref-clever/reference }
2161   {

```

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

```

2162     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2163     d .value_required:n = true ,
2164   }
2165 }

```

### nudge & co. options

```

2166 \bool_new:N \l__zrefclever_nudge_enabled_bool
2167 \bool_new:N \l__zrefclever_nudge_multitype_bool
2168 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2169 \bool_new:N \l__zrefclever_nudge_singular_bool
2170 \bool_new:N \l__zrefclever_nudge_gender_bool
2171 \tl_new:N \l__zrefclever_ref_gender_tl
2172 \keys_define:nn { zref-clever/reference }
2173 {
2174   nudge .choice: ,
2175   nudge / true .code:n =
2176   { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2177   nudge / false .code:n =
2178   { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2179   nudge / ifdraft .code:n =
2180   {
2181     \ifdraft
2182       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2183       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2184     } ,
2185   nudge / iffina .code:n =
2186   {
2187     \ifoptionfinal
2188       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2189       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2190     } ,
2191   nudge .initial:n = false ,
2192   nudge .default:n = true ,
2193   nonnudge .meta:n = { nudge = false } ,
2194   nonnudge .value_forbidden:n = true ,
2195   nudgeif .code:n =
2196   {
2197     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2198     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2199     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2200     \clist_map_inline:nn {#1}
2201     {
2202       \str_case:nnF {##1}
2203       {
2204         { multitype }
2205         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2206         { comptosing }

```

```

2207         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2208         { gender }
2209         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2210         { all }
2211         {
2212             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2213             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2214             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2215         }
2216     }
2217     {
2218         \msg_warning:nnn { zref-clever }
2219         { nudgeif-unknown-value } {##1}
2220     }
2221 }
2222 },
2223 nudgeif .value_required:n = true ,
2224 nudgeif .initial:n = all ,
2225 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2226 sg .initial:n = false ,
2227 sg .default:n = true ,
2228 g .code:n =
2229 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2230 }
2231 \AddToHook { begindocument }
2232 {
2233     \keys_define:nn { zref-clever/reference }
2234     {

```

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

```

2235         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2236         g .value_required:n = true ,
2237     }
2238 }

```

#### font option

```

2239 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2240 \keys_define:nn { zref-clever/reference }
2241 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

#### titleref option

```

2242 \keys_define:nn { zref-clever/reference }
2243 {
2244     titleref .code:n =
2245     {
2246         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2247         \msg_warning:nxxx { zref-clever } { option-deprecated } { titleref }
2248         { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2249     } ,
2250 }

```

#### vario option

```

2251 \keys_define:nn { zref-clever/reference }

```

```

2252 {
2253     vario .code:n =
2254     {
2255         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2256         \msg_warning:nnxx { zref-clever } { option-deprecated } { vario }
2257         { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2258     } ,
2259 }

```

#### note option

```

2260 \tl_new:N \l__zrefclever_zcref_note_tl
2261 \keys_define:nn { zref-clever/reference }
2262 {
2263     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2264     note .value_required:n = true ,
2265 }

```

#### check option

Integration with zref-check.

```

2266 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2267 \bool_new:N \l__zrefclever_zcref_with_check_bool
2268 \keys_define:nn { zref-clever/reference }
2269 {
2270     check .code:n =
2271     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2272 }
2273 \AddToHook { begindocument }
2274 {
2275     \__zrefclever_if_package_loaded:nTF { zref-check }
2276     {
2277         \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2278         {
2279             \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2280             \keys_define:nn { zref-clever/reference }
2281             {
2282                 check .code:n =
2283                 {
2284                     \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2285                     \keys_set:nn { zref-check / zcheck } { #1 }
2286                 } ,
2287                 check .value_required:n = true ,
2288             }
2289         }
2290         {
2291             \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2292             \keys_define:nn { zref-clever/reference }
2293             {
2294                 check .code:n =
2295                 {
2296                     \msg_warning:nnn { zref-clever }
2297                     { zref-check-too-old } { 2021-09-16-v0.2.1 }
2298                 } ,
2299             }

```

```

2300     }
2301   }
2302   {
2303     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2304     \keys_define:nn { zref-clever/reference }
2305     {
2306       check .code:n =
2307       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2308     }
2309   }
2310 }

```

### 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`.

```

2311 \prop_new:N \l__zrefclever_counter_type_prop
2312 \keys_define:nn { zref-clever/label }
2313 {
2314   countertype .code:n =
2315   {
2316     \keyval_parse:nnn
2317     {
2318       \msg_warning:nnnn { zref-clever }
2319       { key-requires-value } { countertype }
2320     }
2321     {
2322       \__zrefclever_prop_put_non_empty:Nnn
2323       \l__zrefclever_counter_type_prop
2324     }
2325     {#1}
2326   } ,
2327   countertype .value_required:n = true ,
2328   countertype .initial:n =
2329   {
2330     subsection      = section ,
2331     subsubsection   = section ,
2332     subparagraph    = paragraph ,
2333     enumi           = item ,
2334     enumii          = item ,
2335     enumiii         = item ,
2336     enumiv          = item ,
2337     mpfootnote      = footnote ,
2338   } ,
2339 }

```

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<sup>A</sup>T<sub>E</sub>X, 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_resetby: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.

```

2340 \seq_new:N \l__zrefclever_counter_resetters_seq
2341 \keys_define:nn { zref-clever/label }
2342 {
2343   counterresetters .code:n =
2344   {
2345     \clist_map_inline:nn {#1}
2346     {
2347       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2348       {
2349         \seq_put_right:Nn
2350         \l__zrefclever_counter_resetters_seq {##1}
2351       }
2352     }
2353   } ,
2354   counterresetters .initial:n =
2355   {
2356     part ,
2357     chapter ,
2358     section ,
2359     subsection ,
2360     subsubsection ,
2361     paragraph ,
2362     subparagraph ,
2363   } ,
2364   counterresetters .value_required:n = true ,
2365 }
```

### 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_reset_by:n` over the search through `\l__zrefclever_counter_resettters_seq`.

```

2366 \prop_new:N \l__zrefclever_counter_resetby_prop
2367 \keys_define:nn { zref-clever/label }
2368 {
2369   counterresetby .code:n =
2370   {
2371     \keyval_parse:nnn
2372     {
2373       \msg_warning:nnn { zref-clever }
2374       { key-requires-value } { counterresetby }
2375     }
2376     {
2377       \__zrefclever_prop_put_non_empty:Nnn
2378       \l__zrefclever_counter_resetby_prop
2379     }
2380     {#1}
2381   } ,
2382   counterresetby .value_required:n = true ,
2383   counterresetby .initial:n =
2384   {

```

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.

```

2385   enumii = enumi ,
2386   enumiii = enumii ,
2387   enumiv = enumiii ,
2388 } ,
2389 }

```

### 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.

```

2390 \tl_new:N \l__zrefclever_current_counter_tl
2391 \keys_define:nn { zref-clever/label }
2392 {
2393   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2394   currentcounter .value_required:n = true ,
2395   currentcounter .initial:n = \@currentcounter ,
2396 }

```

### nocompat option

```

2397 \bool_new:N \g__zrefclever_nocompat_bool
2398 \seq_new:N \g__zrefclever_nocompat_modules_seq
2399 \keys_define:nn { zref-clever/reference }
2400 {
2401   nocompat .code:n =

```

```

2402 {
2403   \tl_if_empty:nTF {#1}
2404     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2405     {
2406       \clist_map_inline:nn {#1}
2407       {
2408         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2409         {
2410           \seq_gput_right:Nn
2411             \g__zrefclever_nocompat_modules_seq {##1}
2412         }
2413       }
2414     }
2415   } ,
2416 }
2417 \AddToHook { begindocument }
2418 {
2419   \keys_define:nn { zref-clever/reference }
2420   {
2421     nocompat .code:n =
2422     {
2423       \msg_warning:nnn { zref-clever }
2424       { option-preamble-only } { nocompat }
2425     }
2426   }
2427 }
2428 \AtEndOfPackage
2429 {
2430   \AddToHook { begindocument }
2431   {
2432     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2433     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2434   }
2435 }

```

`\_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  $\langle module \rangle$  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.

```

\__zrefclever_compat_module:nn {<module>} {<code>}

2436 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2437 {
2438   \AddToHook { begindocument }
2439   {
2440     \bool_if:NF \g__zrefclever_nocompat_bool
2441     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }

```

```

2442     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2443   }
2444 }

```

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

```

2445 \seq_map_inline:Nn
2446   \g__zrefclever_rf_opts_tl_reference_seq
2447 {
2448   \keys_define:nn { zref-clever/reference }
2449   {
2450     #1 .default:o = \c_novalue_tl ,
2451     #1 .code:n =
2452     {
2453       \tl_if_novalue:nTF {##1}
2454       {
2455         \__zrefclever_opt_tl_unset:c
2456         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2457       }
2458       {
2459         \__zrefclever_opt_tl_set:cn
2460         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2461         {##1}
2462       }
2463     } ,
2464   }
2465 }
2466 \keys_define:nn { zref-clever/reference }
2467 {
2468   refpre .code:n =
2469   {
2470     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2471     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2472     { refpre } { refbounds }
2473   } ,
2474   refpos .code:n =
2475   {
2476     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2477     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2478     { refpos } { refbounds }
2479   } ,
2480   preref .code:n =
2481   {
2482     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2483     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2484     { preref } { refbounds }
2485   } ,
2486   postref .code:n =

```



```

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

```

```

2541         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2542     } ,
2543     #1 / unset .code:n =
2544     {
2545         \_zrefclever_opt_bool_unset:c
2546         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2547     } ,
2548     #1 .default:n = true ,
2549     no #1 .meta:n = { #1 = false } ,
2550     no #1 .value_forbidden:n = true ,
2551 }
2552 }

```

## 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.

```

2553 \keys_define:nn { }
2554 {
2555     zref-clever/zcsetup .inherit:n =
2556     {
2557         zref-clever/label ,
2558         zref-clever/reference ,
2559     }
2560 }

```

`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.

```

2561 \bool_lazy_and:nnT
2562 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2563 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2564 { \msg_warning:nn { zref-clever } { load-time-options } }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```
\zcsetup{<options>}
```

```

2565 \NewDocumentCommand \zcsetup { m }
2566 { \_zrefclever_zcsetup:n {#1} }

```

(End definition for `\zcsetup`.)

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

```

    \__zrefclever_zcsetup:n{<options>}

2567 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2568 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2569 \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>}

2570 \NewDocumentCommand \zcRefTypeSetup { m m }
2571 {
2572   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2573   \keys_set:nn { zref-clever/typesetup } {#2}
2574   \tl_clear:N \l__zrefclever_setup_type_tl
2575 }

(End definition for \zcRefTypeSetup.)

2576 \seq_map_inline:Nn
2577 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2578 {
2579   \keys_define:nn { zref-clever/typesetup }
2580   {
2581     #1 .code:n =
2582     {
2583       \msg_warning:nnn { zref-clever }
2584       { option-not-type-specific } {#1}
2585     } ,
2586   }
2587 }
2588 \seq_map_inline:Nn
2589 \g__zrefclever_rf_opts_tl_typesetup_seq
2590 {
2591   \keys_define:nn { zref-clever/typesetup }
2592   {
2593     #1 .default:o = \c_novalue_tl ,
2594     #1 .code:n =
2595     {
2596       \tl_if_novalue:nTF {##1}
2597       {
2598         \__zrefclever_opt_tl_unset:c
2599         {
2600           \__zrefclever_opt_varname_type:enn
2601           { \l__zrefclever_setup_type_tl } {#1} { tl }
2602         }

```

```

2603     }
2604     {
2605         \__zrefclever_opt_tl_set:cn
2606         {
2607             \__zrefclever_opt_varname_type:enn
2608             { \l__zrefclever_setup_type_tl } {#1} { t1 }
2609         }
2610         {##1}
2611     }
2612 } ,
2613 }
2614 }
2615 \keys_define:nn { zref-clever/typesetup }
2616 {
2617     endrange .code:n =
2618     {
2619         \str_case:nnF {#1}
2620         {
2621             { ref }
2622             {
2623                 \__zrefclever_opt_tl_clear:c
2624                 {
2625                     \__zrefclever_opt_varname_type:enn
2626                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2627                 }
2628                 \__zrefclever_opt_tl_clear:c
2629                 {
2630                     \__zrefclever_opt_varname_type:enn
2631                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2632                 }
2633             }
2634
2635             { stripprefix }
2636             {
2637                 \__zrefclever_opt_tl_set:cn
2638                 {
2639                     \__zrefclever_opt_varname_type:enn
2640                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2641                 }
2642                 { __zrefclever_get_endrange_stripprefix }
2643                 \__zrefclever_opt_tl_clear:c
2644                 {
2645                     \__zrefclever_opt_varname_type:enn
2646                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2647                 }
2648             }
2649
2650             { pagecomp }
2651             {
2652                 \__zrefclever_opt_tl_set:cn
2653                 {
2654                     \__zrefclever_opt_varname_type:enn
2655                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2656                 }

```

```

2657         { __zrefclever_get_endrange_pagecomp }
2658     \__zrefclever_opt_tl_clear:c
2659     {
2660         \__zrefclever_opt_varname_type:enn
2661         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2662     }
2663 }
2664
2665 { pagecomp2 }
2666 {
2667     \__zrefclever_opt_tl_set:cn
2668     {
2669         \__zrefclever_opt_varname_type:enn
2670         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2671     }
2672     { __zrefclever_get_endrange_pagecomptwo }
2673     \__zrefclever_opt_tl_clear:c
2674     {
2675         \__zrefclever_opt_varname_type:enn
2676         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2677     }
2678 }
2679
2680 { unset }
2681 {
2682     \__zrefclever_opt_tl_unset:c
2683     {
2684         \__zrefclever_opt_varname_type:enn
2685         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2686     }
2687     \__zrefclever_opt_tl_unset:c
2688     {
2689         \__zrefclever_opt_varname_type:enn
2690         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2691     }
2692 }
2693 }
2694 {
2695     \tl_if_empty:nTF {#1}
2696     {
2697         \msg_warning:nnn { zref-clever }
2698         { endrange-property-undefined } {#1}
2699     }
2700     {
2701         \zref@ifpropundefined {#1}
2702         {
2703             \msg_warning:nnn { zref-clever }
2704             { endrange-property-undefined } {#1}
2705         }
2706         {
2707             \__zrefclever_opt_tl_set:cn
2708             {
2709                 \__zrefclever_opt_varname_type:enn
2710                 { \l__zrefclever_setup_type_tl }

```

```

2711         { endrangefunc } { t1 }
2712     }
2713     { __zrefclever_get_endrange_property }
2714     \__zrefclever_opt_t1_set:cn
2715     {
2716         \__zrefclever_opt_varname_type:enn
2717         { \l__zrefclever_setup_type_t1 }
2718         { endrangeprop } { t1 }
2719     }
2720     {#1}
2721 }
2722 }
2723 }
2724 } ,
2725     endrange .value_required:n = true ,
2726 }
2727 \keys_define:nn { zref-clever/typesetup }
2728 {
2729     refpre .code:n =
2730     {
2731         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2732         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2733         { refpre } { refbounds }
2734     } ,
2735     refpos .code:n =
2736     {
2737         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2738         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2739         { refpos } { refbounds }
2740     } ,
2741     preref .code:n =
2742     {
2743         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2744         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2745         { preref } { refbounds }
2746     } ,
2747     postref .code:n =
2748     {
2749         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2750         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2751         { postref } { refbounds }
2752     } ,
2753 }
2754 \seq_map_inline:Nn
2755 \g__zrefclever_rf_opts_seq_refbounds_seq
2756 {
2757     \keys_define:nn { zref-clever/typesetup }
2758     {
2759         #1 .default:o = \c_novalue_tl ,
2760         #1 .code:n =
2761         {
2762             \tl_if_novalue:nTF {##1}
2763             {
2764                 \__zrefclever_opt_seq_unset:c

```

```

2765         {
2766             \__zrefclever_opt_varname_type:enn
2767             { \l__zrefclever_setup_type_tl } {#1} { seq }
2768         }
2769     }
2770     {
2771         \seq_clear:N \l_tmpa_seq
2772         \__zrefclever_opt_seq_set_clist_split:Nn
2773         \l_tmpa_seq {##1}
2774         \bool_lazy_or:nnTF
2775         { \tl_if_empty_p:n {##1} }
2776         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2777         {
2778             \__zrefclever_opt_seq_set_eq:cN
2779             {
2780                 \__zrefclever_opt_varname_type:enn
2781                 { \l__zrefclever_setup_type_tl } {#1} { seq }
2782             }
2783             \l_tmpa_seq
2784         }
2785         {
2786             \msg_warning:nxxx { zref-clever }
2787             { refbounds-must-be-four }
2788             {#1} { \seq_count:N \l_tmpa_seq }
2789         }
2790     }
2791 } ,
2792 }
2793 }
2794 \seq_map_inline:Nn
2795 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2796 {
2797     \keys_define:nn { zref-clever/typesetup }
2798     {
2799         #1 .choice: ,
2800         #1 / true .code:n =
2801         {
2802             \__zrefclever_opt_bool_set_true:c
2803             {
2804                 \__zrefclever_opt_varname_type:enn
2805                 { \l__zrefclever_setup_type_tl }
2806                 {#1} { bool }
2807             }
2808         } ,
2809         #1 / false .code:n =
2810         {
2811             \__zrefclever_opt_bool_set_false:c
2812             {
2813                 \__zrefclever_opt_varname_type:enn
2814                 { \l__zrefclever_setup_type_tl }
2815                 {#1} { bool }
2816             }
2817         } ,
2818         #1 / unset .code:n =

```

```

2819     {
2820       \__zrefclever_opt_bool_unset:c
2821       {
2822         \__zrefclever_opt_varname_type:enn
2823         { \l__zrefclever_setup_type_tl }
2824         {#1} { bool }
2825       }
2826     } ,
2827     #1 .default:n = true ,
2828     no #1 .meta:n = { #1 = false } ,
2829     no #1 .value_forbidden:n = true ,
2830   }
2831 }

```

### 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>}
2832 \NewDocumentCommand \zcLanguageSetup { m m }
2833 {
2834   \group_begin:
2835   \__zrefclever_language_if_declared:NTF {#1}
2836   {
2837     \tl_clear:N \l__zrefclever_setup_type_tl
2838     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2839     \__zrefclever_opt_seq_get:cNF
2840     {
2841       \__zrefclever_opt_varname_language:nnn
2842       {#1} { declension } { seq }
2843     }
2844     \l__zrefclever_lang_declension_seq
2845     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2846     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2847     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2848     {
2849       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2850       \l__zrefclever_lang_decl_case_tl
2851     }
2852     \__zrefclever_opt_seq_get:cNF
2853     {
2854       \__zrefclever_opt_varname_language:nnn
2855       {#1} { gender } { seq }
2856     }
2857     \l__zrefclever_lang_gender_seq
2858     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2859     \keys_set:nn { zref-clever/langsetup } {#2}

```



```

2860     }
2861     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2862     \group_end:
2863   }
2864   \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2865 \keys_define:nn { zref-clever/langsetup }
2866 {
2867   type .code:n =
2868   {
2869     \tl_if_empty:NTF {#1}
2870     { \tl_clear:N \l__zrefclever_setup_type_tl }
2871     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2872   } ,
2873
2874   case .code:n =
2875   {
2876     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2877     {
2878       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2879       { \l__zrefclever_setup_language_tl } {#1}
2880     }
2881     {
2882       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2883       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2884       {
2885         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2886         {#1} { \l__zrefclever_setup_language_tl }
2887         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2888         \l__zrefclever_lang_decl_case_tl
2889       }
2890     }
2891   } ,
2892   case .value_required:n = true ,
2893
2894   gender .value_required:n = true ,
2895   gender .code:n =
2896   {
2897     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2898     {
2899       \msg_warning:nxxxx { zref-clever } { language-no-gender }
2900       { \l__zrefclever_setup_language_tl } { gender } {#1}
2901     }
2902     {
2903       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2904       {
2905         \msg_warning:nnn { zref-clever }
2906         { option-only-type-specific } { gender }
2907       }
2908       {
2909         \seq_clear:N \l_tmpa_seq

```

```

2910         \clist_map_inline:nn {#1}
2911         {
2912             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2913             { \seq_put_right:Nn \l_tmpa_seq {##1} }
2914             {
2915                 \msg_warning:nxxx { zref-clever }
2916                 { gender-not-declared }
2917                 { \l__zrefclever_setup_language_tl } {##1}
2918             }
2919         }
2920     \__zrefclever_opt_seq_gset_eq:cN
2921     {
2922         \__zrefclever_opt_varname_lang_type:eenn
2923         { \l__zrefclever_setup_language_tl }
2924         { \l__zrefclever_setup_type_tl }
2925         { gender }
2926         { seq }
2927     }
2928     \l_tmpa_seq
2929 }
2930 }
2931 } ,
2932 }
2933 \seq_map_inline:Nn
2934 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2935 {
2936     \keys_define:nn { zref-clever/langsetup }
2937     {
2938         #1 .value_required:n = true ,
2939         #1 .code:n =
2940         {
2941             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2942             {
2943                 \__zrefclever_opt_tl_gset:cn
2944                 {
2945                     \__zrefclever_opt_varname_lang_default:enn
2946                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2947                 }
2948                 {##1}
2949             }
2950             {
2951                 \msg_warning:nnn { zref-clever }
2952                 { option-not-type-specific } {#1}
2953             }
2954         } ,
2955     }
2956 }
2957 \seq_map_inline:Nn
2958 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2959 {
2960     \keys_define:nn { zref-clever/langsetup }
2961     {
2962         #1 .value_required:n = true ,
2963         #1 .code:n =

```

```

2964 {
2965   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2966   {
2967     \__zrefclever_opt_tl_gset:cn
2968     {
2969       \__zrefclever_opt_varname_lang_default:enn
2970       { \l__zrefclever_setup_language_tl } {#1} { t1 }
2971     }
2972     {##1}
2973   }
2974   {
2975     \__zrefclever_opt_tl_gset:cn
2976     {
2977       \__zrefclever_opt_varname_lang_type:eenn
2978       { \l__zrefclever_setup_language_tl }
2979       { \l__zrefclever_setup_type_tl }
2980       {#1} { t1 }
2981     }
2982     {##1}
2983   }
2984   } ,
2985 }
2986 }
2987 \keys_define:nn { zref-clever/langsetup }
2988 {
2989   endrange .value_required:n = true ,
2990   endrange .code:n =
2991   {
2992     \str_case:nnF {#1}
2993     {
2994       { ref }
2995       {
2996         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2997         {
2998           \__zrefclever_opt_tl_gclear:c
2999           {
3000             \__zrefclever_opt_varname_lang_default:enn
3001             { \l__zrefclever_setup_language_tl }
3002             { endrangefunc } { t1 }
3003           }
3004           \__zrefclever_opt_tl_gclear:c
3005           {
3006             \__zrefclever_opt_varname_lang_default:enn
3007             { \l__zrefclever_setup_language_tl }
3008             { endrangeprop } { t1 }
3009           }
3010         }
3011         {
3012           \__zrefclever_opt_tl_gclear:c
3013           {
3014             \__zrefclever_opt_varname_lang_type:eenn
3015             { \l__zrefclever_setup_language_tl }
3016             { \l__zrefclever_setup_type_tl }
3017             { endrangefunc } { t1 }

```

```

3018     }
3019     \__zrefclever_opt_tl_gclear:c
3020     {
3021         \__zrefclever_opt_varname_lang_type:enn
3022         { \l__zrefclever_setup_language_tl }
3023         { \l__zrefclever_setup_type_tl }
3024         { endrangeprop } { tl }
3025     }
3026 }
3027 }
3028
3029 { stripprefix }
3030 {
3031     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3032     {
3033         \__zrefclever_opt_tl_gset:cn
3034         {
3035             \__zrefclever_opt_varname_lang_default:enn
3036             { \l__zrefclever_setup_language_tl }
3037             { endrangefunc } { tl }
3038         }
3039         { __zrefclever_get_endrange_stripprefix }
3040         \__zrefclever_opt_tl_gclear:c
3041         {
3042             \__zrefclever_opt_varname_lang_default:enn
3043             { \l__zrefclever_setup_language_tl }
3044             { endrangeprop } { tl }
3045         }
3046     }
3047     {
3048         \__zrefclever_opt_tl_gset:cn
3049         {
3050             \__zrefclever_opt_varname_lang_type:enn
3051             { \l__zrefclever_setup_language_tl }
3052             { \l__zrefclever_setup_type_tl }
3053             { endrangefunc } { tl }
3054         }
3055         { __zrefclever_get_endrange_stripprefix }
3056         \__zrefclever_opt_tl_gclear:c
3057         {
3058             \__zrefclever_opt_varname_lang_type:enn
3059             { \l__zrefclever_setup_language_tl }
3060             { \l__zrefclever_setup_type_tl }
3061             { endrangeprop } { tl }
3062         }
3063     }
3064 }
3065
3066 { pagecomp }
3067 {
3068     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3069     {
3070         \__zrefclever_opt_tl_gset:cn
3071         {

```

```

3072         \_zrefclever_opt_varname_lang_default:enn
3073         { \l_zrefclever_setup_language_tl }
3074         { endrangefunc } { tl }
3075     }
3076     { __zrefclever_get_endrange_pagecomp }
3077     \_zrefclever_opt_tl_gclear:c
3078     {
3079         \_zrefclever_opt_varname_lang_default:enn
3080         { \l_zrefclever_setup_language_tl }
3081         { endrangeprop } { tl }
3082     }
3083 }
3084 {
3085     \_zrefclever_opt_tl_gset:cn
3086     {
3087         \_zrefclever_opt_varname_lang_type:eenn
3088         { \l_zrefclever_setup_language_tl }
3089         { \l_zrefclever_setup_type_tl }
3090         { endrangefunc } { tl }
3091     }
3092     { __zrefclever_get_endrange_pagecomp }
3093     \_zrefclever_opt_tl_gclear:c
3094     {
3095         \_zrefclever_opt_varname_lang_type:eenn
3096         { \l_zrefclever_setup_language_tl }
3097         { \l_zrefclever_setup_type_tl }
3098         { endrangeprop } { tl }
3099     }
3100 }
3101 }
3102
3103 { pagecomp2 }
3104 {
3105     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3106     {
3107         \_zrefclever_opt_tl_gset:cn
3108         {
3109             \_zrefclever_opt_varname_lang_default:enn
3110             { \l_zrefclever_setup_language_tl }
3111             { endrangefunc } { tl }
3112         }
3113         { __zrefclever_get_endrange_pagecomptwo }
3114         \_zrefclever_opt_tl_gclear:c
3115         {
3116             \_zrefclever_opt_varname_lang_default:enn
3117             { \l_zrefclever_setup_language_tl }
3118             { endrangeprop } { tl }
3119         }
3120     }
3121     {
3122         \_zrefclever_opt_tl_gset:cn
3123         {
3124             \_zrefclever_opt_varname_lang_type:eenn
3125             { \l_zrefclever_setup_language_tl }

```

```

3126         { \l__zrefclever_setup_type_tl }
3127         { endrangefunc } { tl }
3128     }
3129     { __zrefclever_get_endrange_pagecomptwo }
3130 \__zrefclever_opt_tl_gclear:c
3131 {
3132     \__zrefclever_opt_varname_lang_type:enn
3133     { \l__zrefclever_setup_language_tl }
3134     { \l__zrefclever_setup_type_tl }
3135     { endrangeprop } { tl }
3136 }
3137 }
3138 }
3139 }
3140 {
3141     \tl_if_empty:nTF {#1}
3142     {
3143         \msg_warning:nnn { zref-clever }
3144         { endrange-property-undefined } {#1}
3145     }
3146     {
3147         \zref@ifpropundefined {#1}
3148         {
3149             \msg_warning:nnn { zref-clever }
3150             { endrange-property-undefined } {#1}
3151         }
3152         {
3153             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3154             {
3155                 \__zrefclever_opt_tl_gset:cn
3156                 {
3157                     \__zrefclever_opt_varname_lang_default:enn
3158                     { \l__zrefclever_setup_language_tl }
3159                     { endrangefunc } { tl }
3160                 }
3161                 { __zrefclever_get_endrange_property }
3162                 \__zrefclever_opt_tl_gset:cn
3163                 {
3164                     \__zrefclever_opt_varname_lang_default:enn
3165                     { \l__zrefclever_setup_language_tl }
3166                     { endrangeprop } { tl }
3167                 }
3168                 {#1}
3169             }
3170         }
3171         {
3172             \__zrefclever_opt_tl_gset:cn
3173             {
3174                 \__zrefclever_opt_varname_lang_type:enn
3175                 { \l__zrefclever_setup_language_tl }
3176                 { \l__zrefclever_setup_type_tl }
3177                 { endrangefunc } { tl }
3178             }
3179             { __zrefclever_get_endrange_property }
3180             \__zrefclever_opt_tl_gset:cn

```

```

3180         {
3181             \__zrefclever_opt_varname_lang_type:eenm
3182             { \l__zrefclever_setup_language_tl }
3183             { \l__zrefclever_setup_type_tl }
3184             { endrangeprop } { tl }
3185         }
3186         {#1}
3187     }
3188 }
3189 }
3190 }
3191 } ,
3192 }
3193 \keys_define:nn { zref-clever/langsetup }
3194 {
3195     refpre .code:n =
3196     {
3197         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3198         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3199         { refpre } { refbounds }
3200     } ,
3201     refpos .code:n =
3202     {
3203         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3204         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3205         { refpos } { refbounds }
3206     } ,
3207     preref .code:n =
3208     {
3209         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3210         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3211         { preref } { refbounds }
3212     } ,
3213     postref .code:n =
3214     {
3215         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3216         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3217         { postref } { refbounds }
3218     } ,
3219 }
3220 \seq_map_inline:Nn
3221 \g__zrefclever_rf_opts_tl_type_names_seq
3222 {
3223     \keys_define:nn { zref-clever/langsetup }
3224     {
3225         #1 .value_required:n = true ,
3226         #1 .code:n =
3227         {
3228             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3229             {
3230                 \msg_warning:nnn { zref-clever }
3231                 { option-only-type-specific } {#1}
3232             }
3233             {

```

```

3234 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3235 {
3236   \__zrefclever_opt_tl_gset:cn
3237   {
3238     \__zrefclever_opt_varname_lang_type:een
3239     { \l__zrefclever_setup_language_tl }
3240     { \l__zrefclever_setup_type_tl }
3241     {#1} { tl }
3242   }
3243   {##1}
3244 }
3245 {
3246   \__zrefclever_opt_tl_gset:cn
3247   {
3248     \__zrefclever_opt_varname_lang_type:een
3249     { \l__zrefclever_setup_language_tl }
3250     { \l__zrefclever_setup_type_tl }
3251     { \l__zrefclever_lang_decl_case_tl - #1 }
3252     { tl }
3253   }
3254   {##1}
3255 }
3256 } ,
3257 }
3258 }
3259 }
3260 \seq_map_inline:Nn
3261 \g__zrefclever_rf_opts_seq_refbounds_seq
3262 {
3263   \keys_define:nn { zref-clever/langsetup }
3264   {
3265     #1 .value_required:n = true ,
3266     #1 .code:n =
3267     {
3268       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3269       {
3270         \seq_gclear:N \g_tmpa_seq
3271         \__zrefclever_opt_seq_gset_clist_split:Nn
3272         \g_tmpa_seq {##1}
3273         \bool_lazy_or:nnTF
3274         { \tl_if_empty_p:n {##1} }
3275         {
3276           \int_compare_p:nNn
3277           { \seq_count:N \g_tmpa_seq } = { 4 }
3278         }
3279         {
3280           \__zrefclever_opt_seq_gset_eq:cn
3281           {
3282             \__zrefclever_opt_varname_lang_default:enn
3283             { \l__zrefclever_setup_language_tl }
3284             {#1} { seq }
3285           }
3286           \g_tmpa_seq
3287         }
3288       }
3289     }
3290   }

```



```

3288         {
3289             \msg_warning:nxxx { zref-clever }
3290             { rebounds-must-be-four }
3291             {#1} { \seq_count:N \g_tmpa_seq }
3292         }
3293     }
3294     {
3295         \seq_gclear:N \g_tmpa_seq
3296         \__zrefclever_opt_seq_gset_clist_split:Nn
3297             \g_tmpa_seq {##1}
3298         \bool_lazy_or:nnTF
3299             { \tl_if_empty_p:n {##1} }
3300             {
3301                 \int_compare_p:nNn
3302                     { \seq_count:N \g_tmpa_seq } = { 4 }
3303             }
3304             {
3305                 \__zrefclever_opt_seq_gset_eq:cN
3306                 {
3307                     \__zrefclever_opt_varname_lang_type:eenn
3308                     { \l__zrefclever_setup_language_tl }
3309                     { \l__zrefclever_setup_type_tl } {#1} { seq }
3310                 }
3311                 \g_tmpa_seq
3312             }
3313         {
3314             \msg_warning:nxxx { zref-clever }
3315             { rebounds-must-be-four }
3316             {#1} { \seq_count:N \g_tmpa_seq }
3317         }
3318     }
3319 } ,
3320 }
3321 }
3322 \seq_map_inline:Nn
3323 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3324 {
3325     \keys_define:nn { zref-clever/langsetup }
3326     {
3327         #1 .choice: ,
3328         #1 / true .code:n =
3329         {
3330             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3331             {
3332                 \__zrefclever_opt_bool_gset_true:c
3333                 {
3334                     \__zrefclever_opt_varname_lang_default:enn
3335                     { \l__zrefclever_setup_language_tl }
3336                     {#1} { bool }
3337                 }
3338             }
3339             {
3340                 \__zrefclever_opt_bool_gset_true:c
3341                 {

```

```

3342         \_zrefclever_opt_varname_lang_type:enn
3343         { \l\_zrefclever_setup_language_tl }
3344         { \l\_zrefclever_setup_type_tl }
3345         {#1} { bool }
3346     }
3347 }
3348 },
3349 #1 / false .code:n =
3350 {
3351     \tl_if_empty:NTF \l\_zrefclever_setup_type_tl
3352     {
3353         \_zrefclever_opt_bool_gset_false:c
3354         {
3355             \_zrefclever_opt_varname_lang_default:enn
3356             { \l\_zrefclever_setup_language_tl }
3357             {#1} { bool }
3358         }
3359     }
3360     {
3361         \_zrefclever_opt_bool_gset_false:c
3362         {
3363             \_zrefclever_opt_varname_lang_type:enn
3364             { \l\_zrefclever_setup_language_tl }
3365             { \l\_zrefclever_setup_type_tl }
3366             {#1} { bool }
3367         }
3368     }
3369 },
3370 #1 .default:n = true ,
3371 no #1 .meta:n = { #1 = false } ,
3372 no #1 .value_forbidden:n = true ,
3373 }
3374 }

```

## 6 User interface

### 6.1 \zcref

\zcref The main user command of the package.

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

```

3375 \NewDocumentCommand \zcref { s O { } m }
3376 { \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}
```

```

3377 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3378 {
3379   \group_begin:

```

Set options.

```

3380   \keys_set:nn { zref-clever/reference } {#3}

```

Store arguments values.

```

3381   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3382   \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.

```

3383   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }

```

Process language settings.

```

3384   \__zrefclever_process_language_settings:

```

Integration with zref-check.

```

3385   \bool_lazy_and:nnT
3386     { \l__zrefclever_zrefcheck_available_bool }
3387     { \l__zrefclever_zcref_with_check_bool }
3388     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

3389   \bool_lazy_or:nnT
3390     { \l__zrefclever_typeset_sort_bool }
3391     { \l__zrefclever_typeset_range_bool }
3392     { \__zrefclever_sort_labels: }

```

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

```

3393   \group_begin:
3394   \l__zrefclever_ref_typeset_font_tl
3395   \__zrefclever_typeset_refs:
3396   \group_end:

```

Typeset note.

```

3397   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3398   {
3399     \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3400     { \l__zrefclever_label_type_a_tl }
3401     { \l__zrefclever_ref_language_tl }
3402     \l_tmpa_tl
3403     \l_tmpa_tl
3404     \l__zrefclever_zcref_note_tl
3405   }

```

Integration with zref-check.

```

3406   \bool_lazy_and:nnT
3407     { \l__zrefclever_zrefcheck_available_bool }
3408     { \l__zrefclever_zcref_with_check_bool }
3409     {
3410       \zrefcheck_zcref_end_label_maybe:
3411       \zrefcheck_zcref_run_checks_on_labels:n

```

```

3412         { \l__zrefclever_zcref_labels_seq }
3413     }

```

Integration with mathtools.

```

3414     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3415     {
3416         \__zrefclever_mathtools_showonlyrefs:n
3417         { \l__zrefclever_zcref_labels_seq }
3418     }
3419     \group_end:
3420 }

```

(End definition for \\_\_zrefclever\_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3421 \seq_new:N \l__zrefclever_zcref_labels_seq
3422 \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>}

```

```

3423 \NewDocumentCommand \zcpageref { s O { } m }
3424 {
3425     \group_begin:
3426     \IfBooleanT {#1}
3427     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3428     \zcref [#2, ref = page] {#3}
3429     \group_end:
3430 }

```

(End definition for \zcpageref.)

## 7 Sorting

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

$\backslash l\_zrefclever\_label\_type\_a\_tl$ $\backslash l\_zrefclever\_label\_type\_b\_tl$ $\backslash l\_zrefclever\_label\_enclval\_a\_tl$ $\backslash l\_zrefclever\_label\_enclval\_b\_tl$ $\backslash l\_zrefclever\_label\_extdoc\_a\_tl$ $\backslash l\_zrefclever\_label\_extdoc\_b\_tl$	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p> <pre> 3431 \tl_new:N \l__zrefclever_label_type_a_tl 3432 \tl_new:N \l__zrefclever_label_type_b_tl 3433 \tl_new:N \l__zrefclever_label_enclval_a_tl 3434 \tl_new:N \l__zrefclever_label_enclval_b_tl 3435 \tl_new:N \l__zrefclever_label_extdoc_a_tl 3436 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre> <p><i>(End definition for <math>\backslash l\_zrefclever\_label\_type\_a\_tl</math> and others.)</i></p>
$\backslash l\_zrefclever\_sort\_decided\_bool$	<p>Auxiliary variable for <math>\backslash \_zrefclever\_sort\_default\_same\_type:nn</math>, signals if the sorting between two labels has been decided or not.</p> <pre> 3437 \bool_new:N \l__zrefclever_sort_decided_bool </pre> <p><i>(End definition for <math>\backslash l\_zrefclever\_sort\_decided\_bool</math>.)</i></p>
$\backslash l\_zrefclever\_sort\_prior\_a\_int$ $\backslash l\_zrefclever\_sort\_prior\_b\_int$	<p>Auxiliary variables for <math>\backslash \_zrefclever\_sort\_default\_different\_types:nn</math>. Store the sort priority of the “current” and “next” labels.</p> <pre> 3438 \int_new:N \l__zrefclever_sort_prior_a_int 3439 \int_new:N \l__zrefclever_sort_prior_b_int </pre> <p><i>(End definition for <math>\backslash l\_zrefclever\_sort\_prior\_a\_int</math> and <math>\backslash l\_zrefclever\_sort\_prior\_b\_int</math>.)</i></p>
$\backslash l\_zrefclever\_label\_types\_seq$	<p>Stores the order in which reference types appear in the label list supplied by the user in <math>\backslash zcref</math>. This variable is populated by <math>\backslash \_zrefclever\_label\_type\_put\_new\_right:n</math> at the start of <math>\backslash \_zrefclever\_sort\_labels:</math>. This order is required as a “last resort” sort criterion between the reference types, for use in <math>\backslash \_zrefclever\_sort\_default\_different\_types:nn</math>.</p> <pre> 3440 \seq_new:N \l__zrefclever_label_types_seq </pre> <p><i>(End definition for <math>\backslash l\_zrefclever\_label\_types\_seq</math>.)</i></p>
$\backslash \_zrefclever\_sort\_labels:$	<p>The main sorting function. It does not receive arguments, but it is expected to be run inside <math>\backslash \_zrefclever\_zcref:nnnn</math> where a number of environment variables are to be set appropriately. In particular, <math>\backslash l\_zrefclever\_zcref\_labels\_seq</math> should contain the labels received as argument to <math>\backslash zcref</math>, and the function performs its task by sorting this variable.</p> <pre> 3441 \cs_new_protected:Npn \_zrefclever_sort_labels: 3442 { </pre> <p>Store label types sequence.</p> <pre> 3443   \seq_clear:N \l__zrefclever_label_types_seq 3444   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page } 3445   { 3446     \seq_map_function:NN \l__zrefclever_zcref_labels_seq 3447       \_zrefclever_label_type_put_new_right:n 3448   } </pre>

Sort.

```

3449 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3450 {
3451   \zref@ifrefundefined {##1}
3452   {
3453     \zref@ifrefundefined {##2}
3454     {
3455       % Neither label is defined.
3456       \sort_return_same:
3457     }
3458     {
3459       % The second label is defined, but the first isn't, leave the
3460       % undefined first (to be more visible).
3461       \sort_return_same:
3462     }
3463   }
3464   {
3465     \zref@ifrefundefined {##2}
3466     {
3467       % The first label is defined, but the second isn't, bring the
3468       % second forward.
3469       \sort_return_swapped:
3470     }
3471     {
3472       % The interesting case: both labels are defined. References
3473       % to the "default" property or to the "page" are quite
3474       % different with regard to sorting, so we branch them here to
3475       % specialized functions.
3476       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3477       { \__zrefclever_sort_page:nn {##1} {##2} }
3478       { \__zrefclever_sort_default:nn {##1} {##2} }
3479     }
3480   }
3481 }
3482 }

```

(End definition for \\_\_zrefclever\_sort\_labels:.)

\\_\_zrefclever\_label\_type\_put\_new\_right:n

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

3483 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3484 {
3485   \__zrefclever_extract_default:Nnnn
3486   \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3487   \seq_if_in:NVF \l__zrefclever_label_types_seq

```

```

3488 \l__zrefclever_label_type_a_tl
3489 {
3490   \seq_put_right:NV \l__zrefclever_label_types_seq
3491   \l__zrefclever_label_type_a_tl
3492 }
3493 }

```

(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}}

3494 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3495 {
3496   \__zrefclever_extract_default:Nnnn
3497   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3498   \__zrefclever_extract_default:Nnnn
3499   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3500
3501   \tl_if_eq:NNTF
3502   \l__zrefclever_label_type_a_tl
3503   \l__zrefclever_label_type_b_tl
3504   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3505   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3506 }

```

(End definition for \\_\_zrefclever\_sort\_default:nn.)

```

\__zrefclever_sort_default_same_type:nn \__zrefclever_sort_default_same_type:nn {\label a}} {\label b}}

3507 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3508 {
3509   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3510   {#1} {zc@enclval} { }
3511   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3512   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3513   {#2} {zc@enclval} { }
3514   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3515   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3516   {#1} {externaldocument} { }
3517   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3518   {#2} {externaldocument} { }
3519
3520   \bool_set_false:N \l__zrefclever_sort_decided_bool
3521
3522   % First we check if there's any "external document" difference (coming
3523   % from 'zref-xr') and, if so, sort based on that.
3524   \tl_if_eq:NNTF
3525   \l__zrefclever_label_extdoc_a_tl
3526   \l__zrefclever_label_extdoc_b_tl
3527   {

```

```

3528 \bool_if:nTF
3529 {
3530   \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3531   ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3532 }
3533 {
3534   \bool_set_true:N \l__zrefclever_sort_decided_bool
3535   \sort_return_same:
3536 }
3537 {
3538   \bool_if:nTF
3539   {
3540     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3541     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3542   }
3543   {
3544     \bool_set_true:N \l__zrefclever_sort_decided_bool
3545     \sort_return_swapped:
3546   }
3547   {
3548     \bool_set_true:N \l__zrefclever_sort_decided_bool
3549     % Two different "external documents": last resort, sort by the
3550     % document name itself.
3551     \str_compare:eNeTF
3552     { \l__zrefclever_label_extdoc_b_tl } <
3553     { \l__zrefclever_label_extdoc_a_tl }
3554     { \sort_return_swapped: }
3555     { \sort_return_same: }
3556   }
3557 }
3558 }
3559
3560 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3561 {
3562   \bool_if:nTF
3563   {
3564     % Both are empty: neither label has any (further) "enclosing
3565     % counters" (left).
3566     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3567     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3568   }
3569   {
3570     \bool_set_true:N \l__zrefclever_sort_decided_bool
3571     \int_compare:nNnTF
3572     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3573     >
3574     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3575     { \sort_return_swapped: }
3576     { \sort_return_same: }
3577   }
3578   {
3579     \bool_if:nTF
3580     {
3581       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.

```



```

3582     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3583   }
3584   {
3585     \bool_set_true:N \l__zrefclever_sort_decided_bool
3586     \int_compare:nNnTF
3587       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3588       >
3589       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3590       { \sort_return_swapped: }
3591       { \sort_return_same: }
3592   }
3593   {
3594     \bool_if:nTF
3595     {
3596       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3597       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3598     }
3599     {
3600       \bool_set_true:N \l__zrefclever_sort_decided_bool
3601       \int_compare:nNnTF
3602         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3603         <
3604         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3605         { \sort_return_same: }
3606         { \sort_return_swapped: }
3607     }
3608     {
3609       % Neither is empty: we can compare the values of the
3610       % current enclosing counter in the loop, if they are
3611       % equal, we are still in the loop, if they are not, a
3612       % sorting decision can be made directly.
3613       \int_compare:nNnTF
3614         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3615         =
3616         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3617         {
3618           \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3619             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3620           \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3621             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3622         }
3623         {
3624           \bool_set_true:N \l__zrefclever_sort_decided_bool
3625           \int_compare:nNnTF
3626             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3627             >
3628             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3629             { \sort_return_swapped: }
3630             { \sort_return_same: }
3631         }
3632     }
3633   }
3634 }
3635 }

```

```
3636 }
```

(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}
```

```
3637 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
3638 {
```

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”.

```
3639 \int_zero:N \l__zrefclever_sort_prior_a_int
3640 \int_zero:N \l__zrefclever_sort_prior_b_int
3641 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3642 {
3643   \tl_if_eq:nnTF {##2} {{othertypes}}
3644   {
3645     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3646     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3647     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3648     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3649   }
3650   {
3651     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3652     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3653     {
3654       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3655       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3656     }
3657   }
3658 }
```

Then do the actual sorting.

```
3659 \bool_if:nTF
3660 {
3661   \int_compare_p:nNn
3662   { \l__zrefclever_sort_prior_a_int } <
3663   { \l__zrefclever_sort_prior_b_int }
3664 }
3665 { \sort_return_same: }
3666 {
3667   \bool_if:nTF
3668   {
3669     \int_compare_p:nNn
3670     { \l__zrefclever_sort_prior_a_int } >
3671     { \l__zrefclever_sort_prior_b_int }
3672   }
3673   { \sort_return_swapped: }
3674   {
3675     % Sort priorities are equal: the type that occurs first in
3676     % ‘labels’, as given by the user, is kept (or brought) forward.
3677     \seq_map_inline:Nn \l__zrefclever_label_types_seq
3678     {
3679       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
```

```

3680         { \seq_map_break:n { \sort_return_same: } }
3681         {
3682             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3683             { \seq_map_break:n { \sort_return_swapped: } }
3684         }
3685     }
3686 }
3687 }
3688 }

```

(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}

3689 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3690 {
3691     \int_compare:nNnTF
3692     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3693     >
3694     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3695     { \sort_return_swapped: }
3696     { \sort_return_same: }
3697 }

```

(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

Auxiliary variables for `\__zrefclever_typeset_refs`: main stack control.

```
\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_last_bool
\l_zrefclever_last_of_type_bool
3698 \seq_new:N \l__zrefclever_typeset_labels_seq
3699 \bool_new:N \l__zrefclever_typeset_last_bool
3700 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `\__zrefclever_typeset_refs`: main counters.

```
\l_zrefclever_type_count_int
\l_zrefclever_label_count_int
\l__zrefclever_ref_count_int
3701 \int_new:N \l__zrefclever_type_count_int
3702 \int_new:N \l__zrefclever_label_count_int
3703 \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`.)*

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

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl
3704 \tl_new:N \l__zrefclever_label_a_tl
3705 \tl_new:N \l__zrefclever_label_b_tl
3706 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
3707 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
3708 \tl_new:N \l__zrefclever_type_first_label_tl
3709 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

Auxiliary variables for `\__zrefclever_typeset_refs`: type name handling.

```
\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
3710 \tl_new:N \l__zrefclever_type_name_tl
3711 \bool_new:N \l__zrefclever_name_in_link_bool
3712 \bool_new:N \l__zrefclever_type_name_missing_bool
3713 \tl_new:N \l__zrefclever_name_format_tl
3714 \tl_new:N \l__zrefclever_name_format_fallback_tl
3715 \seq_new:N \l__zrefclever_type_name_gender_seq
```

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

Auxiliary variables for `\__zrefclever_typeset_refs`: range handling.

```
\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
3716 \int_new:N \l__zrefclever_range_count_int
3717 \int_new:N \l__zrefclever_range_same_count_int
3718 \tl_new:N \l__zrefclever_range_beg_label_tl
3719 \bool_new:N \l__zrefclever_range_beg_is_first_bool
3720 \tl_new:N \l__zrefclever_range_end_ref_tl
3721 \bool_new:N \l__zrefclever_next_maybe_range_bool
3722 \bool_new:N \l__zrefclever_next_is_same_bool
```

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

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_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

```

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

```

3723 \tl_new:N \l__zrefclever_tpairsep_tl
3724 \tl_new:N \l__zrefclever_tlistsep_tl
3725 \tl_new:N \l__zrefclever_tlastsep_tl
3726 \tl_new:N \l__zrefclever_namesep_tl
3727 \tl_new:N \l__zrefclever_pairsep_tl
3728 \tl_new:N \l__zrefclever_listsep_tl
3729 \tl_new:N \l__zrefclever_lastsep_tl
3730 \tl_new:N \l__zrefclever_rangeseq_tl
3731 \tl_new:N \l__zrefclever_namefont_tl
3732 \tl_new:N \l__zrefclever_reffont_tl
3733 \tl_new:N \l__zrefclever_endrangefunc_tl
3734 \tl_new:N \l__zrefclever_endrangeprop_tl
3735 \bool_new:N \l__zrefclever_cap_bool
3736 \bool_new:N \l__zrefclever_abbrev_bool
3737 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

```

\l__zrefclever_refbounds_first_seq
\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

```

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

```

3738 \seq_new:N \l__zrefclever_refbounds_first_seq
3739 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3740 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3741 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3742 \seq_new:N \l__zrefclever_refbounds_mid_seq
3743 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3744 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3745 \seq_new:N \l__zrefclever_refbounds_last_seq
3746 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3747 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3748 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3749 \bool_new:N \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`.

```

3750 \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`.

```

3751 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3752 {
3753   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3754   \l__zrefclever_zceref_labels_seq
3755   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3756   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3757   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

3758 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3759 \tl_clear:N \l__zrefclever_range_beg_label_tl
3760 \tl_clear:N \l__zrefclever_range_end_ref_tl
3761 \int_zero:N \l__zrefclever_label_count_int
3762 \int_zero:N \l__zrefclever_type_count_int
3763 \int_zero:N \l__zrefclever_ref_count_int
3764 \int_zero:N \l__zrefclever_range_count_int
3765 \int_zero:N \l__zrefclever_range_same_count_int
3766 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3767 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3768
3769 % Get type block options (not type-specific).
3770 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3771   { \l__zrefclever_label_type_a_tl }
3772   { \l__zrefclever_ref_language_tl }
3773   \l__zrefclever_tpairsep_tl
3774 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3775   { \l__zrefclever_label_type_a_tl }
3776   { \l__zrefclever_ref_language_tl }
3777   \l__zrefclever_tlistsep_tl
3778 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3779   { \l__zrefclever_label_type_a_tl }
3780   { \l__zrefclever_ref_language_tl }
3781   \l__zrefclever_tlastsep_tl
3782
3783 % Process label stack.
3784 \bool_set_false:N \l__zrefclever_typeset_last_bool
3785 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3786   {
3787     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3788     \l__zrefclever_label_a_tl
3789     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3790       {
3791         \tl_clear:N \l__zrefclever_label_b_tl
3792         \bool_set_true:N \l__zrefclever_typeset_last_bool
3793       }
3794       {
3795         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3796         \l__zrefclever_label_b_tl
3797       }
3798
3799     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3800       {
3801         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3802         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3803       }
3804       {
3805         \__zrefclever_extract_default:NVnn
3806         \l__zrefclever_label_type_a_tl
3807         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3808         \__zrefclever_extract_default:NVnn
3809         \l__zrefclever_label_type_b_tl
3810         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3811       }

```

```

3812
3813 % First, we establish whether the "current label" (i.e. 'a') is the
3814 % last one of its type. This can happen because the "next label"
3815 % (i.e. 'b') is of a different type (or different definition status),
3816 % or because we are at the end of the list.
3817 \bool_if:NTF \l__zrefclever_typeset_last_bool
3818 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3819 {
3820   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3821   {
3822     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3823     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3824     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3825   }
3826   {
3827     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3828     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3829     {
3830       % Neither is undefined, we must check the types.
3831       \tl_if_eq:NNTF
3832         \l__zrefclever_label_type_a_tl
3833         \l__zrefclever_label_type_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 }
3839
3840 % Handle warnings in case of reference or type undefined.
3841 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3842 \zref@refused { \l__zrefclever_label_a_tl }
3843 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3844 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3845 {}
3846 {
3847   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3848   {
3849     \msg_warning:nxx { zref-clever } { missing-type }
3850     { \l__zrefclever_label_a_tl }
3851   }
3852   \zref@ifrefcontainsprop
3853     { \l__zrefclever_label_a_tl }
3854     { \l__zrefclever_ref_property_tl }
3855     { }
3856     {
3857       \msg_warning:nxxx { zref-clever } { missing-property }
3858       { \l__zrefclever_ref_property_tl }
3859       { \l__zrefclever_label_a_tl }
3860     }
3861 }
3862
3863 % Get possibly type-specific separators, refbounds, font and other
3864 % options, once per type.
3865 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```



```

3866 {
3867   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3868     { \l__zrefclever_label_type_a_tl }
3869     { \l__zrefclever_ref_language_tl }
3870     \l__zrefclever_namesep_tl
3871   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3872     { \l__zrefclever_label_type_a_tl }
3873     { \l__zrefclever_ref_language_tl }
3874     \l__zrefclever_pairsep_tl
3875   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3876     { \l__zrefclever_label_type_a_tl }
3877     { \l__zrefclever_ref_language_tl }
3878     \l__zrefclever_listsep_tl
3879   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3880     { \l__zrefclever_label_type_a_tl }
3881     { \l__zrefclever_ref_language_tl }
3882     \l__zrefclever_lastsep_tl
3883   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3884     { \l__zrefclever_label_type_a_tl }
3885     { \l__zrefclever_ref_language_tl }
3886     \l__zrefclever_rangesep_tl
3887   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3888     { \l__zrefclever_label_type_a_tl }
3889     { \l__zrefclever_ref_language_tl }
3890     \l__zrefclever_namefont_tl
3891   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3892     { \l__zrefclever_label_type_a_tl }
3893     { \l__zrefclever_ref_language_tl }
3894     \l__zrefclever_reffont_tl
3895   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3896     { \l__zrefclever_label_type_a_tl }
3897     { \l__zrefclever_ref_language_tl }
3898     \l__zrefclever_endrangefunc_tl
3899   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3900     { \l__zrefclever_label_type_a_tl }
3901     { \l__zrefclever_ref_language_tl }
3902     \l__zrefclever_endrangeprop_tl
3903   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3904     { \l__zrefclever_label_type_a_tl }
3905     { \l__zrefclever_ref_language_tl }
3906     \l__zrefclever_cap_bool
3907   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3908     { \l__zrefclever_label_type_a_tl }
3909     { \l__zrefclever_ref_language_tl }
3910     \l__zrefclever_abbrev_bool
3911   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3912     { \l__zrefclever_label_type_a_tl }
3913     { \l__zrefclever_ref_language_tl }
3914     \l__zrefclever_rangetopair_bool
3915   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3916     { \l__zrefclever_label_type_a_tl }
3917     { \l__zrefclever_ref_language_tl }
3918     \l__zrefclever_refbounds_first_seq
3919   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3920         { \l__zrefclever_label_type_a_tl }
3921         { \l__zrefclever_ref_language_tl }
3922         \l__zrefclever_refbounds_first_sg_seq
3923     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3924         { \l__zrefclever_label_type_a_tl }
3925         { \l__zrefclever_ref_language_tl }
3926         \l__zrefclever_refbounds_first_pb_seq
3927     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3928         { \l__zrefclever_label_type_a_tl }
3929         { \l__zrefclever_ref_language_tl }
3930         \l__zrefclever_refbounds_first_rb_seq
3931     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3932         { \l__zrefclever_label_type_a_tl }
3933         { \l__zrefclever_ref_language_tl }
3934         \l__zrefclever_refbounds_mid_seq
3935     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3936         { \l__zrefclever_label_type_a_tl }
3937         { \l__zrefclever_ref_language_tl }
3938         \l__zrefclever_refbounds_mid_rb_seq
3939     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3940         { \l__zrefclever_label_type_a_tl }
3941         { \l__zrefclever_ref_language_tl }
3942         \l__zrefclever_refbounds_mid_re_seq
3943     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3944         { \l__zrefclever_label_type_a_tl }
3945         { \l__zrefclever_ref_language_tl }
3946         \l__zrefclever_refbounds_last_seq
3947     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3948         { \l__zrefclever_label_type_a_tl }
3949         { \l__zrefclever_ref_language_tl }
3950         \l__zrefclever_refbounds_last_pe_seq
3951     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3952         { \l__zrefclever_label_type_a_tl }
3953         { \l__zrefclever_ref_language_tl }
3954         \l__zrefclever_refbounds_last_re_seq
3955 }
3956
3957 % Here we send this to a couple of auxiliary functions.
3958 \bool_if:NTF \l__zrefclever_last_of_type_bool
3959 % There exists no next label of the same type as the current.
3960 { \__zrefclever_typeset_refs_last_of_type: }
3961 % There exists a next label of the same type as the current.
3962 { \__zrefclever_typeset_refs_not_last_of_type: }
3963 }
3964 }

```

(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.

```

3965 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3966 {
3967   % Process the current label to the current queue.
3968   \int_case:nnF { \l__zrefclever_label_count_int }
3969   {
3970     % It is the last label of its type, but also the first one, and that's
3971     % what matters here: just store it.
3972     % Test: 'zc-typeset01.lvt': "Last of type: single"
3973     { 0 }
3974     {
3975       \tl_set:NV \l__zrefclever_type_first_label_tl
3976       \l__zrefclever_label_a_tl
3977       \tl_set:NV \l__zrefclever_type_first_label_type_tl
3978       \l__zrefclever_label_type_a_tl
3979       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3980       \l__zrefclever_refbounds_first_sg_seq
3981       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3982     }
3983
3984     % The last is the second: we have a pair (if not repeated).
3985     % Test: 'zc-typeset01.lvt': "Last of type: pair"
3986     { 1 }
3987     {
3988       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
3989       {
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         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3996         {
3997           \exp_not:V \l__zrefclever_pairsep_tl
3998           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3999           \l__zrefclever_refbounds_last_pe_seq
4000         }
4001         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4002         \l__zrefclever_refbounds_first_pb_seq
4003         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4004       }
4005     }
4006   }
4007   % Last is third or more of its type: without repetition, we'd have the
4008   % last element on a list, but control for possible repetition.
4009   {
4010     \int_case:nnF { \l__zrefclever_range_count_int }
4011     {
4012       % There was no range going on.
4013       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4014       { 0 }

```

```

4015 {
4016   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4017   {
4018     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4019     {
4020       \exp_not:V \l__zrefclever_pairsep_tl
4021       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4022       \l__zrefclever_refbounds_last_pe_seq
4023     }
4024   }
4025   {
4026     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4027     {
4028       \exp_not:V \l__zrefclever_lastsep_tl
4029       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4030       \l__zrefclever_refbounds_last_seq
4031     }
4032   }
4033 }
4034 % Last in the range is also the second in it.
4035 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4036 { 1 }
4037 {
4038   \int_compare:nNnTF
4039   { \l__zrefclever_range_same_count_int } = { 1 }
4040   {
4041     % We know 'range_beg_is_first_bool' is false, since this is
4042     % the second element in the range, but the third or more in
4043     % the type list.
4044     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4045     {
4046       \exp_not:V \l__zrefclever_pairsep_tl
4047       \__zrefclever_get_ref:VN
4048       \l__zrefclever_range_beg_label_tl
4049       \l__zrefclever_refbounds_last_pe_seq
4050     }
4051     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4052     \l__zrefclever_refbounds_first_pb_seq
4053     \bool_set_true:N
4054     \l__zrefclever_type_first_refbounds_set_bool
4055   }
4056   {
4057     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4058     {
4059       \exp_not:V \l__zrefclever_listsep_tl
4060       \__zrefclever_get_ref:VN
4061       \l__zrefclever_range_beg_label_tl
4062       \l__zrefclever_refbounds_mid_seq
4063       \exp_not:V \l__zrefclever_lastsep_tl
4064       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4065       \l__zrefclever_refbounds_last_seq
4066     }
4067   }
4068 }

```

```

4069 }
4070 % Last in the range is third or more in it.
4071 {
4072   \int_case:nnF
4073   {
4074     \l__zrefclever_range_count_int -
4075     \l__zrefclever_range_same_count_int
4076   }
4077   {
4078     % Repetition, not a range.
4079     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4080     { 0 }
4081     {
4082       % If 'range_beg_is_first_bool' is true, it means it was also
4083       % the first of the type, and hence its typesetting was
4084       % already handled, and we just have to set rebounds.
4085       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4086       {
4087         \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4088         \l__zrefclever_rebounds_first_sg_seq
4089         \bool_set_true:N
4090         \l__zrefclever_type_first_rebounds_set_bool
4091       }
4092       {
4093         \int_compare:nNnTF
4094         { \l__zrefclever_ref_count_int } < { 2 }
4095         {
4096           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4097           {
4098             \exp_not:V \l__zrefclever_pairsep_tl
4099             \__zrefclever_get_ref:VN
4100             \l__zrefclever_range_beg_label_tl
4101             \l__zrefclever_rebounds_last_pe_seq
4102           }
4103         }
4104         {
4105           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4106           {
4107             \exp_not:V \l__zrefclever_lastsep_tl
4108             \__zrefclever_get_ref:VN
4109             \l__zrefclever_range_beg_label_tl
4110             \l__zrefclever_rebounds_last_seq
4111           }
4112         }
4113       }
4114     }
4115     % A 'range', but with no skipped value, treat as pair if range
4116     % started with first of type, otherwise as list.
4117     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4118     { 1 }
4119     {
4120       % Ditto.
4121       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4122       {

```

```

4123         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4124         \l__zrefclever_refbounds_first_pb_seq
4125     \bool_set_true:N
4126         \l__zrefclever_type_first_refbounds_set_bool
4127     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4128     {
4129         \exp_not:V \l__zrefclever_pairsep_tl
4130         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4131         \l__zrefclever_refbounds_last_pe_seq
4132     }
4133 }
4134 {
4135     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4136     {
4137         \exp_not:V \l__zrefclever_listsep_tl
4138         \__zrefclever_get_ref:VN
4139         \l__zrefclever_range_beg_label_tl
4140         \l__zrefclever_refbounds_mid_seq
4141     }
4142     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4143     {
4144         \exp_not:V \l__zrefclever_lastsep_tl
4145         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4146         \l__zrefclever_refbounds_last_seq
4147     }
4148 }
4149 }
4150 }
4151 {
4152     % An actual range.
4153     % Test: 'zc-typeset01.lvt': "Last of type: range"
4154     % Ditto.
4155     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4156     {
4157         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4158         \l__zrefclever_refbounds_first_rb_seq
4159         \bool_set_true:N
4160         \l__zrefclever_type_first_refbounds_set_bool
4161     }
4162     {
4163         \int_compare:nNnTF
4164         { \l__zrefclever_ref_count_int } < { 2 }
4165         {
4166             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4167             {
4168                 \exp_not:V \l__zrefclever_pairsep_tl
4169                 \__zrefclever_get_ref:VN
4170                 \l__zrefclever_range_beg_label_tl
4171                 \l__zrefclever_refbounds_mid_rb_seq
4172             }
4173             \seq_set_eq:NN
4174             \l__zrefclever_type_first_refbounds_seq
4175             \l__zrefclever_refbounds_first_pb_seq
4176             \bool_set_true:N

```

```

4177         \l__zrefclever_type_first_refbounds_set_bool
4178     }
4179     {
4180         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4181         {
4182             \exp_not:V \l__zrefclever_lastsep_tl
4183             \__zrefclever_get_ref:VN
4184             \l__zrefclever_range_beg_label_tl
4185             \l__zrefclever_refbounds_mid_rb_seq
4186         }
4187     }
4188 }
4189 \bool_lazy_and:nnTF
4190 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4191 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4192 {
4193     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4194     \l__zrefclever_range_beg_label_tl
4195     \l__zrefclever_label_a_tl
4196     \l__zrefclever_range_end_ref_tl
4197     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4198     {
4199         \exp_not:V \l__zrefclever_rangesep_tl
4200         \__zrefclever_get_ref_endrange:VVN
4201         \l__zrefclever_label_a_tl
4202         \l__zrefclever_range_end_ref_tl
4203         \l__zrefclever_refbounds_last_re_seq
4204     }
4205 }
4206 {
4207     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4208     {
4209         \exp_not:V \l__zrefclever_rangesep_tl
4210         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4211         \l__zrefclever_refbounds_last_re_seq
4212     }
4213 }
4214 }
4215 }
4216 }
4217
4218 % Handle "range" option. The idea is simple: if the queue is not empty,
4219 % we replace it with the end of the range (or pair). We can still
4220 % retrieve the end of the range from 'label_a' since we know to be
4221 % processing the last label of its type at this point.
4222 \bool_if:NT \l__zrefclever_typeset_range_bool
4223 {
4224     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4225     {
4226         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4227         { }
4228         {
4229             \msg_warning:nxx { zref-clever } { single-element-range }
4230             { \l__zrefclever_type_first_label_type_tl }

```

```

4231     }
4232 }
4233 {
4234   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4235   \bool_if:NT \l__zrefclever_rangetopair_bool
4236   {
4237     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4238     { }
4239     {
4240       \__zrefclever_labels_in_sequence:nn
4241       { \l__zrefclever_type_first_label_tl }
4242       { \l__zrefclever_label_a_tl }
4243     }
4244   }
4245   % Test: 'zc-typeset01.lvt': "Last of type: option range"
4246   % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4247   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4248   {
4249     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4250     {
4251       \exp_not:V \l__zrefclever_pairsep_tl
4252       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4253       \l__zrefclever_refbounds_last_pe_seq
4254     }
4255     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4256     \l__zrefclever_refbounds_first_pb_seq
4257     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4258   }
4259   {
4260     \bool_lazy_and:nnTF
4261     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4262     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4263     {
4264       % We must get 'type_first_label_tl' instead of
4265       % 'range_beg_label_tl' here, since it is not necessary
4266       % that the first of type was actually starting a range for
4267       % the 'range' option to be used.
4268       \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4269       \l__zrefclever_type_first_label_tl
4270       \l__zrefclever_label_a_tl
4271       \l__zrefclever_range_end_ref_tl
4272       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4273       {
4274         \exp_not:V \l__zrefclever_rangesep_tl
4275         \__zrefclever_get_ref_endrange:VVN
4276         \l__zrefclever_label_a_tl
4277         \l__zrefclever_range_end_ref_tl
4278         \l__zrefclever_refbounds_last_re_seq
4279       }
4280     }
4281   }
4282   \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4283   {
4284     \exp_not:V \l__zrefclever_rangesep_tl

```



```

4285         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4286         \l__zrefclever_refbounds_last_re_seq
4287     }
4288 }
4289 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4290 \l__zrefclever_refbounds_first_rb_seq
4291 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4292 }
4293 }
4294 }
4295
4296 % If none of the special cases for the first of type refbounds have been
4297 % set, do it.
4298 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4299 {
4300     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4301     \l__zrefclever_refbounds_first_seq
4302 }
4303
4304 % Now that the type block is finished, we can add the name and the first
4305 % ref to the queue. Also, if "typeset" option is not "both", handle it
4306 % here as well.
4307 \_zrefclever_type_name_setup:
4308 \bool_if:nTF
4309 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4310 {
4311     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4312     { \_zrefclever_get_ref_first: }
4313 }
4314 {
4315     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4316     {
4317         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4318         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4319         {
4320             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4321             \l__zrefclever_type_first_refbounds_seq
4322         }
4323     }
4324     {
4325         \bool_if:NTF \l__zrefclever_typeset_name_bool
4326         {
4327             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4328             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4329             {
4330                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4331                 {
4332                     \exp_not:N \group_begin:
4333                     \exp_not:V \l__zrefclever_namefont_tl
4334                     \_zrefclever_hyperlink:nnn
4335                     {
4336                         \_zrefclever_extract_url_unexp:V
4337                         \l__zrefclever_type_first_label_tl
4338                     }

```

```

4339         {
4340             \__zrefclever_extract_unexp:Vnn
4341             \l__zrefclever_type_first_label_tl
4342             { anchor } { }
4343         }
4344         { \exp_not:V \l__zrefclever_type_name_tl }
4345         \exp_not:N \group_end:
4346     }
4347     {
4348         \exp_not:N \group_begin:
4349         \exp_not:V \l__zrefclever_namefont_tl
4350         \exp_not:V \l__zrefclever_type_name_tl
4351         \exp_not:N \group_end:
4352     }
4353 }
4354 }
4355 {
4356     % Logically, this case would correspond to "typeset=none", but
4357     % it should not occur, given that the options are set up to
4358     % typeset either "ref" or "name". Still, leave here a
4359     % sensible fallback, equal to the behavior of "both".
4360     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4361     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4362     { \__zrefclever_get_ref_first: }
4363 }
4364 }
4365 }
4366
4367 % Typeset the previous type block, if there is one.
4368 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4369 {
4370     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4371     { \l__zrefclever_tlistsep_tl }
4372     \l__zrefclever_typeset_queue_prev_tl
4373 }
4374
4375 % Extra log for testing.
4376 \bool_if:NT \l__zrefclever_verbose_testing_bool
4377 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4378
4379 % Wrap up loop, or prepare for next iteration.
4380 \bool_if:NTF \l__zrefclever_typeset_last_bool
4381 {
4382     % We are finishing, typeset the current queue.
4383     \int_case:nnF { \l__zrefclever_type_count_int }
4384     {
4385         % Single type.
4386         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4387         { 0 }
4388         { \l__zrefclever_typeset_queue_curr_tl }
4389         % Pair of types.
4390         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4391         { 1 }
4392     }

```

```

4393         \l__zrefclever_tpairsep_tl
4394         \l__zrefclever_typeset_queue_curr_tl
4395     }
4396 }
4397 {
4398     % Last in list of types.
4399     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4400     \l__zrefclever_tlastsep_tl
4401     \l__zrefclever_typeset_queue_curr_tl
4402 }
4403 % And nudge in case of multitype reference.
4404 \bool_lazy_all:nT
4405 {
4406     { \l__zrefclever_nudge_enabled_bool }
4407     { \l__zrefclever_nudge_multitype_bool }
4408     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4409 }
4410 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4411 }
4412 {
4413     % There are further labels, set variables for next iteration.
4414     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4415         \l__zrefclever_typeset_queue_curr_tl
4416     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4417     \tl_clear:N \l__zrefclever_type_first_label_tl
4418     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4419     \tl_clear:N \l__zrefclever_range_beg_label_tl
4420     \tl_clear:N \l__zrefclever_range_end_ref_tl
4421     \int_zero:N \l__zrefclever_label_count_int
4422     \int_zero:N \l__zrefclever_ref_count_int
4423     \int_incr:N \l__zrefclever_type_count_int
4424     \int_zero:N \l__zrefclever_range_count_int
4425     \int_zero:N \l__zrefclever_range_same_count_int
4426     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4427     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4428 }
4429 }

```

(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.

```

4430 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4431 {
4432     % Signal if next label may form a range with the current one (only
4433     % considered if compression is enabled in the first place).
4434     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4435     \bool_set_false:N \l__zrefclever_next_is_same_bool
4436     \bool_if:NT \l__zrefclever_typeset_compress_bool
4437     {
4438         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4439         { }
4440         {
4441             \__zrefclever_labels_in_sequence:nn
4442             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4443     }
4444 }
4445
4446 % Process the current label to the current queue.
4447 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4448 {
4449     % Current label is the first of its type (also not the last, but it
4450     % doesn't matter here): just store the label.
4451     \tl_set:NV \l__zrefclever_type_first_label_tl
4452         \l__zrefclever_label_a_tl
4453     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4454         \l__zrefclever_label_type_a_tl
4455     \int_incr:N \l__zrefclever_ref_count_int
4456
4457     % If the next label may be part of a range, signal it (we deal with it
4458     % as the "first", and must do it there, to handle hyperlinking), but
4459     % also step the range counters.
4460     % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4461     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4462     {
4463         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4464         \tl_set:NV \l__zrefclever_range_beg_label_tl
4465             \l__zrefclever_label_a_tl
4466         \tl_clear:N \l__zrefclever_range_end_ref_tl
4467         \int_incr:N \l__zrefclever_range_count_int
4468         \bool_if:NT \l__zrefclever_next_is_same_bool
4469             { \int_incr:N \l__zrefclever_range_same_count_int }
4470     }
4471 }
4472 {
4473     % Current label is neither the first (nor the last) of its type.
4474     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4475     {
4476         % Starting, or continuing a range.
4477         \int_compare:nNnTF
4478             { \l__zrefclever_range_count_int } = { 0 }
4479         {
4480             % There was no range going, we are starting one.
4481             \tl_set:NV \l__zrefclever_range_beg_label_tl
4482                 \l__zrefclever_label_a_tl
4483             \tl_clear:N \l__zrefclever_range_end_ref_tl
4484             \int_incr:N \l__zrefclever_range_count_int
4485             \bool_if:NT \l__zrefclever_next_is_same_bool
4486                 { \int_incr:N \l__zrefclever_range_same_count_int }
4487         }
4488         {
4489             % Second or more in the range, but not the last.
4490             \int_incr:N \l__zrefclever_range_count_int
4491             \bool_if:NT \l__zrefclever_next_is_same_bool
4492                 { \int_incr:N \l__zrefclever_range_same_count_int }
4493         }
4494     }
4495 }
4496 % Next element is not in sequence: there was no range, or we are

```

```

4497 % closing one.
4498 \int_case:nnF { \l__zrefclever_range_count_int }
4499 {
4500 % There was no range going on.
4501 % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4502 { 0 }
4503 {
4504 \int_incr:N \l__zrefclever_ref_count_int
4505 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4506 {
4507 \exp_not:V \l__zrefclever_listsep_tl
4508 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4509 \l__zrefclever_refbounds_mid_seq
4510 }
4511 }
4512 % Last is second in the range: if 'range_same_count' is also
4513 % '1', it's a repetition (drop it), otherwise, it's a "pair
4514 % within a list", treat as list.
4515 % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4516 % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4517 { 1 }
4518 {
4519 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4520 {
4521 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4522 \l__zrefclever_refbounds_first_seq
4523 \bool_set_true:N
4524 \l__zrefclever_type_first_refbounds_set_bool
4525 }
4526 {
4527 \int_incr:N \l__zrefclever_ref_count_int
4528 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4529 {
4530 \exp_not:V \l__zrefclever_listsep_tl
4531 \__zrefclever_get_ref:VN
4532 \l__zrefclever_range_beg_label_tl
4533 \l__zrefclever_refbounds_mid_seq
4534 }
4535 }
4536 \int_compare:nNnF
4537 { \l__zrefclever_range_same_count_int } = { 1 }
4538 {
4539 \int_incr:N \l__zrefclever_ref_count_int
4540 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4541 {
4542 \exp_not:V \l__zrefclever_listsep_tl
4543 \__zrefclever_get_ref:VN
4544 \l__zrefclever_label_a_tl
4545 \l__zrefclever_refbounds_mid_seq
4546 }
4547 }
4548 }
4549 }
4550 {

```

```

4551 % Last is third or more in the range: if 'range_count' and
4552 % 'range_same_count' are the same, its a repetition (drop it),
4553 % if they differ by '1', its a list, if they differ by more,
4554 % it is a real range.
4555 \int_case:nnF
4556 {
4557   \l__zrefclever_range_count_int -
4558   \l__zrefclever_range_same_count_int
4559 }
4560 {
4561   % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4562   { 0 }
4563   {
4564     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4565     {
4566       \seq_set_eq:NN
4567       \l__zrefclever_type_first_refbounds_seq
4568       \l__zrefclever_refbounds_first_seq
4569       \bool_set_true:N
4570       \l__zrefclever_type_first_refbounds_set_bool
4571     }
4572     {
4573       \int_incr:N \l__zrefclever_ref_count_int
4574       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4575       {
4576         \exp_not:V \l__zrefclever_listsep_tl
4577         \__zrefclever_get_ref:VN
4578         \l__zrefclever_range_beg_label_tl
4579         \l__zrefclever_refbounds_mid_seq
4580       }
4581     }
4582   }
4583   % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4584   { 1 }
4585   {
4586     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4587     {
4588       \seq_set_eq:NN
4589       \l__zrefclever_type_first_refbounds_seq
4590       \l__zrefclever_refbounds_first_seq
4591       \bool_set_true:N
4592       \l__zrefclever_type_first_refbounds_set_bool
4593     }
4594     {
4595       \int_incr:N \l__zrefclever_ref_count_int
4596       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4597       {
4598         \exp_not:V \l__zrefclever_listsep_tl
4599         \__zrefclever_get_ref:VN
4600         \l__zrefclever_range_beg_label_tl
4601         \l__zrefclever_refbounds_mid_seq
4602       }
4603     }
4604     \int_incr:N \l__zrefclever_ref_count_int

```

```

4605 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4606 {
4607   \exp_not:V \l__zrefclever_listsep_tl
4608   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4609   \l__zrefclever_refbounds_mid_seq
4610 }
4611 }
4612 }
4613 {
4614   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4615   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4616   {
4617     \seq_set_eq:NN
4618     \l__zrefclever_type_first_refbounds_seq
4619     \l__zrefclever_refbounds_first_rb_seq
4620     \bool_set_true:N
4621     \l__zrefclever_type_first_refbounds_set_bool
4622   }
4623   {
4624     \int_incr:N \l__zrefclever_ref_count_int
4625     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4626     {
4627       \exp_not:V \l__zrefclever_listsep_tl
4628       \__zrefclever_get_ref:VN
4629       \l__zrefclever_range_beg_label_tl
4630       \l__zrefclever_refbounds_mid_rb_seq
4631     }
4632   }
4633   % For the purposes of the serial comma, and thus for the
4634   % distinction of 'lastsep' and 'pairsep', a "range" counts
4635   % as one. Since 'range_beg' has already been counted
4636   % (here or with the first of type), we refrain from
4637   % incrementing 'ref_count_int'.
4638   \bool_lazy_and:nnTF
4639   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4640   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4641   {
4642     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4643     \l__zrefclever_range_beg_label_tl
4644     \l__zrefclever_label_a_tl
4645     \l__zrefclever_range_end_ref_tl
4646     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4647     {
4648       \exp_not:V \l__zrefclever_rangesep_tl
4649       \__zrefclever_get_ref_endrange:VVN
4650       \l__zrefclever_label_a_tl
4651       \l__zrefclever_range_end_ref_tl
4652       \l__zrefclever_refbounds_mid_re_seq
4653     }
4654   }
4655   {
4656     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4657     {
4658       \exp_not:V \l__zrefclever_rangesep_tl

```

```

4659             \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4660             \l__zrefclever_refbounds_mid_re_seq
4661         }
4662     }
4663 }
4664 }
4665 % We just closed a range, reset 'range_beg_is_first' in case a
4666 % second range for the same type occurs, in which case its
4667 % 'range_beg' will no longer be 'first'.
4668 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4669 % Reset counters.
4670 \int_zero:N \l__zrefclever_range_count_int
4671 \int_zero:N \l__zrefclever_range_same_count_int
4672 }
4673 }
4674 % Step label counter for next iteration.
4675 \int_incr:N \l__zrefclever_label_count_int
4676 }

```

(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.

```

4677 \cs_new_protected:Npn \_zrefclever_ref_default:
4678 { \zref@default }
4679 \cs_new_protected:Npn \_zrefclever_name_default:
4680 { \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}}

4681 \cs_new:Npn \_zrefclever_get_ref:nN #1#2
4682 {
4683   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4684   {
4685     \bool_if:nTF
4686     {
4687       \l__zrefclever_hyperlink_bool &&
4688       ! \l__zrefclever_link_star_bool
4689     }
4690     {
4691       \seq_item:Nn #2 { 1 }
4692       \_zrefclever_hyperlink:nnn
4693       { \_zrefclever_extract_url_unexp:n {#1} }
4694       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4695       {
4696         \seq_item:Nn #2 { 2 }
4697         \exp_not:N \group_begin:
4698         \exp_not:V \l__zrefclever_reffont_tl
4699         \_zrefclever_extract_unexp:nvn {#1}
4700         { \l__zrefclever_ref_property_tl } { }
4701         \exp_not:N \group_end:
4702         \seq_item:Nn #2 { 3 }
4703       }
4704       \seq_item:Nn #2 { 4 }
4705     }
4706     {
4707       \seq_item:Nn #2 { 1 }
4708       \seq_item:Nn #2 { 2 }
4709       \exp_not:N \group_begin:
4710       \exp_not:V \l__zrefclever_reffont_tl
4711       \_zrefclever_extract_unexp:nvn {#1}
4712       { \l__zrefclever_ref_property_tl } { }
4713       \exp_not:N \group_end:
4714       \seq_item:Nn #2 { 3 }
4715       \seq_item:Nn #2 { 4 }
4716     }
4717   }
4718   { \_zrefclever_ref_default: }
4719 }
4720 \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}}
4721 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4722 {
4723   \str_if_eq:nnTF {#2} { zc@missingproperty }
4724   { \__zrefclever_ref_default: }
4725   {
4726     \bool_if:nTF
4727     {
4728       \l__zrefclever_hyperlink_bool &&
4729       ! \l__zrefclever_link_star_bool
4730     }
4731     {
4732       \seq_item:Nn #3 { 1 }
4733       \__zrefclever_hyperlink:nnn
4734       { \__zrefclever_extract_url_unexp:n {#1} }
4735       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4736       {
4737         \seq_item:Nn #3 { 2 }
4738         \exp_not:N \group_begin:
4739         \exp_not:V \l__zrefclever_reffont_tl
4740         \exp_not:n {#2}
4741         \exp_not:N \group_end:
4742         \seq_item:Nn #3 { 3 }
4743       }
4744       \seq_item:Nn #3 { 4 }
4745     }
4746     {
4747       \seq_item:Nn #3 { 1 }
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       \seq_item:Nn #3 { 4 }
4755     }
4756   }
4757 }
4758 \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.

```

4759 \cs_new:Npn \__zrefclever_get_ref_first:
4760 {
4761   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4762   { \__zrefclever_ref_default: }
4763   {
4764     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4765 {
4766   \zref@ifrefcontainsprop
4767   { \l__zrefclever_type_first_label_tl }
4768   { \l__zrefclever_ref_property_tl }
4769   {
4770     \__zrefclever_hyperlink:nnn
4771     {
4772       \__zrefclever_extract_url_unexp:V
4773       \l__zrefclever_type_first_label_tl
4774     }
4775     {
4776       \__zrefclever_extract_unexp:Vnn
4777       \l__zrefclever_type_first_label_tl { anchor } { }
4778     }
4779     {
4780       \exp_not:N \group_begin:
4781       \exp_not:V \l__zrefclever_namefont_tl
4782       \exp_not:V \l__zrefclever_type_name_tl
4783       \exp_not:N \group_end:
4784       \exp_not:V \l__zrefclever_namesep_tl
4785       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4786       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4787       \exp_not:N \group_begin:
4788       \exp_not:V \l__zrefclever_reffont_tl
4789       \__zrefclever_extract_unexp:Vnn
4790       \l__zrefclever_type_first_label_tl
4791       { \l__zrefclever_ref_property_tl } { }
4792       \exp_not:N \group_end:
4793       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4794     }
4795     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4796   }
4797   {
4798     \exp_not:N \group_begin:
4799     \exp_not:V \l__zrefclever_namefont_tl
4800     \exp_not:V \l__zrefclever_type_name_tl
4801     \exp_not:N \group_end:
4802     \exp_not:V \l__zrefclever_namesep_tl
4803     \__zrefclever_ref_default:
4804   }
4805 }
4806 {
4807   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4808   {
4809     \__zrefclever_name_default:
4810     \exp_not:V \l__zrefclever_namesep_tl
4811   }
4812   {
4813     \exp_not:N \group_begin:
4814     \exp_not:V \l__zrefclever_namefont_tl
4815     \exp_not:V \l__zrefclever_type_name_tl
4816     \exp_not:N \group_end:
4817     \tl_if_empty:NF \l__zrefclever_type_name_tl
4818     { \exp_not:V \l__zrefclever_namesep_tl }

```

```

4819     }
4820 \zref@ifrefcontainsprop
4821 { \l__zrefclever_type_first_label_tl }
4822 { \l__zrefclever_ref_property_tl }
4823 {
4824     \bool_if:nTF
4825     {
4826         \l__zrefclever_hyperlink_bool &&
4827         ! \l__zrefclever_link_star_bool
4828     }
4829     {
4830         \seq_item:Nn
4831         \l__zrefclever_type_first_refbounds_seq { 1 }
4832         \__zrefclever_hyperlink:nnn
4833         {
4834             \__zrefclever_extract_url_unexp:V
4835             \l__zrefclever_type_first_label_tl
4836         }
4837         {
4838             \__zrefclever_extract_unexp:Vnn
4839             \l__zrefclever_type_first_label_tl { anchor } { }
4840         }
4841         {
4842             \seq_item:Nn
4843             \l__zrefclever_type_first_refbounds_seq { 2 }
4844             \exp_not:N \group_begin:
4845             \exp_not:V \l__zrefclever_reffont_tl
4846             \__zrefclever_extract_unexp:Vnn
4847             \l__zrefclever_type_first_label_tl
4848             { \l__zrefclever_ref_property_tl } { }
4849             \exp_not:N \group_end:
4850             \seq_item:Nn
4851             \l__zrefclever_type_first_refbounds_seq { 3 }
4852         }
4853         \seq_item:Nn
4854         \l__zrefclever_type_first_refbounds_seq { 4 }
4855     }
4856     {
4857         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4858         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4859         \exp_not:N \group_begin:
4860         \exp_not:V \l__zrefclever_reffont_tl
4861         \__zrefclever_extract_unexp:Vnn
4862         \l__zrefclever_type_first_label_tl
4863         { \l__zrefclever_ref_property_tl } { }
4864         \exp_not:N \group_end:
4865         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4866         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4867     }
4868 }
4869 { \__zrefclever_ref_default: }
4870 }
4871 }
4872 }

```

(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.

```

4873 \cs_new_protected:Npn \_zrefclever_type_name_setup:
4874 {
4875   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
4876   {
4877     \tl_clear:N \l\_zrefclever_type_name_tl
4878     \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4879   }
4880   {
4881     \tl_if_eq:NnTF
4882       \l\_zrefclever_type_first_label_type_tl { zc@missingtype }
4883       {
4884         \tl_clear:N \l\_zrefclever_type_name_tl
4885         \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4886       }
4887       {
4888         % Determine whether we should use capitalization, abbreviation,
4889         % and plural.
4890         \bool_lazy_or:nnTF
4891           { \l\_zrefclever_cap_bool }
4892           {
4893             \l\_zrefclever_capfirst_bool &&
4894             \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 }
4895           }
4896           { \tl_set:Nn \l\_zrefclever_name_format_tl {Name} }
4897           { \tl_set:Nn \l\_zrefclever_name_format_tl {name} }
4898         % If the queue is empty, we have a singular, otherwise, plural.
4899         \tl_if_empty:NTF \l\_zrefclever_typeset_queue_curr_tl
4900           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -sg } }
4901           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -pl } }
4902         \bool_lazy_and:nnTF
4903           { \l\_zrefclever_abbrev_bool }
4904           {
4905             ! \int_compare_p:nNn
4906               { \l\_zrefclever_type_count_int } = { 0 } ||
4907             ! \l\_zrefclever_noabbrev_first_bool
4908           }
4909           {
4910             \tl_set:NV \l\_zrefclever_name_format_fallback_tl
4911               \l\_zrefclever_name_format_tl
4912             \tl_put_right:Nn \l\_zrefclever_name_format_tl { -ab }

```

```

4913     }
4914     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4915
4916 % Handle number and gender nudges.
4917 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4918 {
4919     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4920     {
4921         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4922         {
4923             \msg_warning:nxx { zref-clever }
4924             { nudge-plural-when-sg }
4925             { \l__zrefclever_type_first_label_type_tl }
4926         }
4927     }
4928     {
4929         \bool_lazy_all:nT
4930         {
4931             { \l__zrefclever_nudge_comptosing_bool }
4932             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4933             {
4934                 \int_compare_p:nNn
4935                 { \l__zrefclever_label_count_int } > { 0 }
4936             }
4937         }
4938         {
4939             \msg_warning:nxx { zref-clever }
4940             { nudge-comptosing }
4941             { \l__zrefclever_type_first_label_type_tl }
4942         }
4943     }
4944     \bool_lazy_and:nnT
4945     { \l__zrefclever_nudge_gender_bool }
4946     { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4947     {
4948         \__zrefclever_get_rf_opt_seq:nxxN { gender }
4949         { \l__zrefclever_type_first_label_type_tl }
4950         { \l__zrefclever_ref_language_tl }
4951         \l__zrefclever_type_name_gender_seq
4952         \seq_if_in:NVF
4953         \l__zrefclever_type_name_gender_seq
4954         \l__zrefclever_ref_gender_tl
4955         {
4956             \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4957             {
4958                 \msg_warning:nxxxx { zref-clever }
4959                 { nudge-gender-not-declared-for-type }
4960                 { \l__zrefclever_ref_gender_tl }
4961                 { \l__zrefclever_type_first_label_type_tl }
4962                 { \l__zrefclever_ref_language_tl }
4963             }
4964             {
4965                 \msg_warning:nxxxxx { zref-clever }
4966                 { nudge-gender-mismatch }

```

```

4967         { \l__zrefclever_type_first_label_type_tl }
4968         { \l__zrefclever_ref_gender_tl }
4969         {
4970             \seq_use:Nn
4971                 \l__zrefclever_type_name_gender_seq { , ~ }
4972         }
4973         { \l__zrefclever_ref_language_tl }
4974     }
4975 }
4976 }
4977 }
4978
4979 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4980 {
4981     \__zrefclever_opt_tl_get:cNF
4982     {
4983         \__zrefclever_opt_varname_type:een
4984         { \l__zrefclever_type_first_label_type_tl }
4985         { \l__zrefclever_name_format_tl }
4986         { tl }
4987     }
4988     \l__zrefclever_type_name_tl
4989     {
4990         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
4991         {
4992             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
4993             \tl_put_left:NV \l__zrefclever_name_format_tl
4994                 \l__zrefclever_ref_decl_case_tl
4995         }
4996         \__zrefclever_opt_tl_get:cNF
4997         {
4998             \__zrefclever_opt_varname_lang_type:een
4999             { \l__zrefclever_ref_language_tl }
5000             { \l__zrefclever_type_first_label_type_tl }
5001             { \l__zrefclever_name_format_tl }
5002             { tl }
5003         }
5004         \l__zrefclever_type_name_tl
5005         {
5006             \tl_clear:N \l__zrefclever_type_name_tl
5007             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5008             \msg_warning:nxxx { zref-clever } { missing-name }
5009             { \l__zrefclever_name_format_tl }
5010             { \l__zrefclever_type_first_label_type_tl }
5011         }
5012     }
5013 }
5014 {
5015     \__zrefclever_opt_tl_get:cNF
5016     {
5017         \__zrefclever_opt_varname_type:een
5018         { \l__zrefclever_type_first_label_type_tl }
5019         { \l__zrefclever_name_format_tl }
5020         { tl }

```

```

5021 }
5022 \l__zrefclever_type_name_tl
5023 {
5024   \__zrefclever_opt_tl_get:cNF
5025   {
5026     \__zrefclever_opt_varname_type:een
5027     { \l__zrefclever_type_first_label_type_tl }
5028     { \l__zrefclever_name_format_fallback_tl }
5029     { tl }
5030   }
5031   \l__zrefclever_type_name_tl
5032   {
5033     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5034     {
5035       \tl_put_left:Nn
5036         \l__zrefclever_name_format_tl { - }
5037       \tl_put_left:NV \l__zrefclever_name_format_tl
5038         \l__zrefclever_ref_decl_case_tl
5039       \tl_put_left:Nn
5040         \l__zrefclever_name_format_fallback_tl { - }
5041       \tl_put_left:NV
5042         \l__zrefclever_name_format_fallback_tl
5043         \l__zrefclever_ref_decl_case_tl
5044     }
5045     \__zrefclever_opt_tl_get:cNF
5046     {
5047       \__zrefclever_opt_varname_lang_type:eeen
5048       { \l__zrefclever_ref_language_tl }
5049       { \l__zrefclever_type_first_label_type_tl }
5050       { \l__zrefclever_name_format_tl }
5051       { tl }
5052     }
5053     \l__zrefclever_type_name_tl
5054     {
5055       \__zrefclever_opt_tl_get:cNF
5056       {
5057         \__zrefclever_opt_varname_lang_type:eeen
5058         { \l__zrefclever_ref_language_tl }
5059         { \l__zrefclever_type_first_label_type_tl }
5060         { \l__zrefclever_name_format_fallback_tl }
5061         { tl }
5062       }
5063       \l__zrefclever_type_name_tl
5064       {
5065         \tl_clear:N \l__zrefclever_type_name_tl
5066         \bool_set_true:N
5067           \l__zrefclever_type_name_missing_bool
5068         \msg_warning:nxxx { zref-clever }
5069           { missing-name }
5070           { \l__zrefclever_name_format_tl }
5071           { \l__zrefclever_type_first_label_type_tl }
5072       }
5073     }
5074   }

```



```

5075         }
5076     }
5077 }
5078 }
5079
5080 % Signal whether the type name is to be included in the hyperlink or not.
5081 \bool_lazy_any:nTF
5082 {
5083     { ! \l__zrefclever_hyperlink_bool }
5084     { \l__zrefclever_link_star_bool }
5085     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5086     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5087 }
5088 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5089 {
5090     \bool_lazy_any:nTF
5091     {
5092         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5093         {
5094             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5095             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5096         }
5097         {
5098             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5099             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5100             \l__zrefclever_typeset_last_bool &&
5101             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5102         }
5103     }
5104     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5105     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5106 }
5107 }

```

(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}
5108 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5109 {
5110     \tl_if_empty:nTF {#1}
5111     { \hyperlink {#2} {#3} }
5112     { \hyper@linkfile {#3} {#1} {#2} }
5113 }

```

(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.

```

5114 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5115 {
5116   \zref@ifpropundefined { urluse }
5117   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5118   {
5119     \zref@ifrefcontainsprop {#1} { urluse }
5120     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5121     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5122   }
5123 }
5124 \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}

5125 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5126 {
5127   \exp_args:Nxx \tl_if_eq:nnT
5128   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5129   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5130   {
5131     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5132     {
5133       \exp_args:Nxx \tl_if_eq:nnT
5134       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5135       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5136       {
5137         \int_compare:nNnTF
5138         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5139         =
5140         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5141         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5142         {
5143           \int_compare:nNnT
5144           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5145           =
5146           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5147           {
5148             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5149             \bool_set_true:N \l__zrefclever_next_is_same_bool
5150           }
5151         }
5152       }
5153     }
5154   }
5155   \exp_args:Nxx \tl_if_eq:nnT

```

```

5156 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5157 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5158 {
5159   \exp_args:Nxx \tl_if_eq:nnT
5160   { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5161   { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5162   {
5163     \int_compare:nNnTF
5164     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5165     =
5166     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5167     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5168     {
5169       \int_compare:nNnTF
5170       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5171       =
5172       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5173       {

```

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.

```

5174   \exp_args:Nxx \tl_if_eq:nnT
5175   {
5176     \_zrefclever_extract_unexp:nvn {#1}
5177     { \l__zrefclever_ref_property_tl } { }
5178   }
5179   {
5180     \_zrefclever_extract_unexp:nvn {#2}
5181     { \l__zrefclever_ref_property_tl } { }
5182   }
5183   {
5184     \bool_set_true:N
5185     \l__zrefclever_next_maybe_range_bool
5186     \bool_set_true:N
5187     \l__zrefclever_next_is_same_bool
5188   }
5189 }
5190 }
5191 }
5192 }
5193 }
5194 }
5195 }

```

(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 {\option}
  {\ref type} {\language} {\tl variable}
5196 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5197 {
5198   % First attempt: general options.
5199   \__zrefclever_opt_tl_get:cNF
5200   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5201   #4
5202   {
5203     % If not found, try type specific options.
5204     \__zrefclever_opt_tl_get:cNF
5205     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5206     #4
5207     {
5208       % If not found, try type- and language-specific.
5209       \__zrefclever_opt_tl_get:cNF
5210       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5211       #4
5212       {
5213         % If not found, try language-specific default.
5214         \__zrefclever_opt_tl_get:cNF
5215         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5216         #4
5217         {
5218           % If not found, try fallback.
5219           \__zrefclever_opt_tl_get:cNF
5220           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5221           #4
5222           { \tl_clear:N #4 }
5223         }
5224       }
5225     }
5226   }
5227 }
5228 \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 {\option}
  {\ref type} {\language} {\seq variable}
5229 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5230 {
5231   % First attempt: general options.
5232   \__zrefclever_opt_seq_get:cNF
5233   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5234   #4
5235   {
5236     % If not found, try type specific options.
5237     \__zrefclever_opt_seq_get:cNF
5238     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5239     #4
5240     {
5241       % If not found, try type- and language-specific.
5242       \__zrefclever_opt_seq_get:cNF

```

```

5243         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5244         #4
5245         {
5246             % If not found, try language-specific default.
5247             \__zrefclever_opt_seq_get:cNF
5248             { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5249             #4
5250             {
5251                 % If not found, try fallback.
5252                 \__zrefclever_opt_seq_get:cNF
5253                 { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5254                 #4
5255                 { \seq_clear:N #4 }
5256             }
5257         }
5258     }
5259 }
5260 }
5261 \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>}
5262 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5263 {
5264     % First attempt: general options.
5265     \__zrefclever_opt_bool_get:cNF
5266     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5267     #5
5268     {
5269         % If not found, try type specific options.
5270         \__zrefclever_opt_bool_get:cNF
5271         { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5272         #5
5273         {
5274             % If not found, try type- and language-specific.
5275             \__zrefclever_opt_bool_get:cNF
5276             { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5277             #5
5278             {
5279                 % If not found, try language-specific default.
5280                 \__zrefclever_opt_bool_get:cNF
5281                 { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5282                 #5
5283                 {
5284                     % If not found, try fallback.
5285                     \__zrefclever_opt_bool_get:cNF
5286                     { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5287                     #5
5288                     { \use:c { bool_set_ #2 :N } #5 }
5289                 }
5290             }
5291         }

```

```

5292     }
5293   }
5294   \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<sup>A</sup>T<sub>E</sub>X 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.

```

5295 \__zrefclever_compat_module:nn { appendix }
5296 {
5297   \AddToHook { cmd / appendix / before }
5298   {
5299     \__zrefclever_zcsetup:n
5300     {
5301       countertype =
5302       {
5303         chapter      = appendix ,
5304         section      = appendix ,
5305         subsection   = appendix ,
5306         subsubsection = appendix ,
5307         paragraph    = appendix ,
5308         subparagraph = appendix ,
5309       }
5310     }
5311   }
5312 }

```

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.

```

5313 \__zrefclever_compat_module:nn { appendices }
5314 {
5315   \__zrefclever_if_package_loaded:nT { appendix }
5316   {
5317     \newcounter { zc@appendix }
5318     \newcounter { zc@save@appendix }
5319     \setcounter { zc@appendix } { 0 }
5320     \setcounter { zc@save@appendix } { 0 }
5321     \cs_if_exist:cTF { chapter }
5322     {
5323       \__zrefclever_zcsetup:n
5324       { counterresetby = { chapter = zc@appendix } }
5325     }
5326     {
5327       \cs_if_exist:cT { section }
5328       {
5329         \__zrefclever_zcsetup:n
5330         { counterresetby = { section = zc@appendix } }
5331       }
5332     }
5333   \AddToHook { env / appendices / begin }
5334   {
5335     \stepcounter { zc@save@appendix }
5336     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5337     \__zrefclever_zcsetup:n
5338     {
5339       countertype =
5340       {
5341         chapter      = appendix ,
5342         section      = appendix ,
5343         subsection   = appendix ,
5344         subsubsection = appendix ,
5345         paragraph    = appendix ,
5346         subparagraph = appendix ,
5347       }
5348     }
5349   }
5350   \AddToHook { env / appendices / end }
5351   { \setcounter { zc@appendix } { 0 } }
5352   \AddToHook { cmd / appendix / before }
5353   {
5354     \stepcounter { zc@save@appendix }
5355     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5356   }

```

```

5357 \AddToHook { env / subappendices / begin }
5358 {
5359     \__zrefclever_zcsetup:n
5360     {
5361         countertype =
5362         {
5363             section      = appendix ,
5364             subsection   = appendix ,
5365             subsubsection = appendix ,
5366             paragraph    = appendix ,
5367             subparagraph = appendix ,
5368         } ,
5369     }
5370 }
5371 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5372 }
5373 }

```

### 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.

```

5374 \__zrefclever_compat_module:n { memoir }
5375 {
5376   \__zrefclever_if_class_loaded:nT { memoir }
5377   {

```

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`.

```

5378   \__zrefclever_zcsetup:n
5379   {
5380     countertype =
5381     {
5382       subfigure = figure ,
5383       subtable  = table ,
5384       poemline  = line ,
5385     } ,
5386     counterresetby =
5387     {
5388       subfigure = figure ,
5389       subtable  = table ,
5390     } ,
5391   }

```

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

```

5392   \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5393   {
5394     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5395     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5396     {
5397       \__zrefclever_memoir_orig_label:n {##1}
5398       \zlabel{##1}
5399     }
5400     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n

```

```

5401     }
5402     \AddToHook { env / sidecaption / begin }
5403       { \_zrefclever_memoir_both_labels: }
5404     \AddToHook { env / sidecontcaption / begin }
5405       { \_zrefclever_memoir_both_labels: }
5406     \AddToHook{ cmd / bitwonumcaption / before }
5407       { \group_begin: \_zrefclever_memoir_both_labels: }
5408     \AddToHook{ cmd / bitwonumcaption / after }
5409       { \group_end: }
5410     \AddToHook{ cmd / bionenumcaption / before }
5411       { \group_begin: \_zrefclever_memoir_both_labels: }
5412     \AddToHook{ cmd / bionenumcaption / after }
5413       { \group_end: }
5414     \AddToHook{ cmd / bicaption / before }
5415       { \group_begin: \_zrefclever_memoir_both_labels: }
5416     \AddToHook{ cmd / bicaption / after }
5417       { \group_end: }

```

Support for subcaption reference.

```

5418     \zref@newprop { subcaption }
5419       { \cs_if_exist_use:c { @@thesub \@capttype } }
5420     \AddToHook{ cmd / @memsubcaption / before }
5421       { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5422     \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5423     \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5424     \AddToHook { env / minipage / begin }
5425       { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5426     \AddToHook { cmd / @makefntext / before }
5427       {
5428         \__zrefclever_zcsetup:x
5429           { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5430       }
5431     \AddToHook { cmd / @makesidefntext / before }
5432       { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5433     \__zrefclever_zcsetup:n
5434       {
5435         countertype =
5436           {
5437             sidefootnote = footnote ,
5438             pagenote = endnote ,
5439           } ,
5440       }
5441     \AddToHook { file / \jobname.ent / before }
5442       { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5443     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5444   }
5445 }

```

## 9.4 KOMA

Support for KOMA-Script document classes.

```

5446   \__zrefclever_compat_module:nn { KOMA }

```

```

5447 {
5448   \cs_if_exist:NT \KOMAClassName
5449   {

```

Add support for `captionbeside` and `captionofbeside` environments. These environments *do* run some variation of `\caption` and hence `\refstepcounter`. However, this happens inside a `parbox` inside the environment, thus grouped, such that we cannot see the variables set by `\refstepcounter` when we are setting the label. `\@currentlabel` is smuggled out of the group by KOMA, but the same care is not granted for `\@currentcounter`. So we have to rely on `\@capttype`, which the underlying caption infrastructure feeds to `\refstepcounter`. Since we must use `env/.../after` hooks, care should be taken not to set the `currentcounter` option unscoped, which would be quite disastrous. For this reason, though more “invasive”, we set `\@currentcounter` instead, which at least will be set straight the next time `\refstepcounter` runs. It sounds reasonable, it is the same treatment `\@currentlabel` is receiving in this case.

```

5450   \AddToHook { env / captionbeside / after }
5451   {
5452     \tl_if_exist:NT \@capttype
5453     { \tl_set_eq:NN \@currentcounter \@capttype }
5454   }
5455   \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
5456   \AddToHook { env / captionofbeside / end }
5457   { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
5458   \AddToHook { env / captionofbeside / after }
5459   {
5460     \tl_if_eq:NnF \@currentenv { document }
5461     {
5462       \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
5463       {
5464         \tl_set_eq:NN
5465         \@currentcounter \g__zrefclever_koma_capttype_tl
5466       }
5467     }
5468     \tl_gclear:N \g__zrefclever_koma_capttype_tl
5469   }
5470   \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
5471 }
5472 }

```

## 9.5 amsmath

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

```

5473 \__zrefclever_compat_module:nn { amsmath }
5474 {
5475   \__zrefclever_if_package_loaded:nT { amsmath }
5476   {

```

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...).

```

5477 \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5478 {
5479   \__zrefclever_orig_ltxlabel:n {#1}
5480   \zref@wrapper@babel \zref@label {#1}
5481 }

```

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>.

```

5482 \__zrefclever_if_package_loaded:nTF { hyperref }
5483 {
5484   \AddToHook { package / nameref / after }
5485   {
5486     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5487     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5488   }
5489 }
5490 {
5491   \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5492   \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5493 }

```

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](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).

```

5494     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5495     \AddToHook { env / subequations / begin }
5496     {
5497         \__zrefclever_zcsetup:x
5498         {
5499             counterresetby =
5500             {
5501                 parentequation =
5502                 \__zrefclever_counter_reset_by:n { equation } ,
5503                 equation = parentequation ,
5504             } ,
5505             currentcounter = parentequation ,
5506             countertype = { parentequation = equation } ,
5507         }
5508         \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5509     }

```

`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.

```

5510     \zref@newprop { subeq } { \alph { equation } }
5511     \clist_map_inline:nn
5512     {
5513         equation ,
5514         equation* ,
5515         align ,
5516         align* ,
5517         alignat ,
5518         alignat* ,
5519         flalign ,
5520         flalign* ,
5521         xalignat ,
5522         xalignat* ,
5523         gather ,
5524         gather* ,
5525         multline ,
5526         multline* ,
5527     }
5528     {
5529         \AddToHook { env / #1 / begin }

```

```

5530         {
5531             \__zrefclever_zcsetup:n { currentcounter = equation }
5532             \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5533                 { \zref@localaddprop \ZREF@mainlist { subeq } }
5534         }
5535     }
5536     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5537 }
5538 }

```

## 9.6 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.

```

5539 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5540 \__zrefclever_compat_module:nn { mathtools }
5541 {
5542     \__zrefclever_if_package_loaded:nT { mathtools }
5543     {
5544         \MH_if_boolean:nT { show_only_refs }
5545         {
5546             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5547             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5548             {
5549                 \@bsphack
5550                 \seq_map_inline:Nn #1
5551                 {
5552                     \exp_args:Nx \tl_if_eq:nnTF
5553                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5554                     { equation }
5555                     {
5556                         \protected@write \@auxout { }
5557                         { \string \MT@newlabel {##1} }
5558                     }
5559                 }
5560                 \exp_args:Nx \tl_if_eq:nnT
5561                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5562                 { parentequation }
5563                 {
5564                     \protected@write \@auxout { }
5565                     { \string \MT@newlabel {##1} }
5566                 }
5567             }
5568         }
5569     }
5570 }

```

```

5571         \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5572     }
5573 }
5574 }

```

## 9.7 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”.

```

5575 \__zrefclever_compat_module:nn { breqn }
5576 {
5577     \__zrefclever_if_package_loaded:nT { breqn }
5578 }

```

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>).

```

5579     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5580     \AddToHook { env / dgroup / begin }
5581     {
5582         \__zrefclever_zcsetup:x
5583         {
5584             counterresetby =
5585             {
5586                 parentequation =
5587                 \__zrefclever_counter_reset_by:n { equation } ,
5588                 equation = parentequation ,
5589             } ,
5590             currentcounter = parentequation ,
5591             countertype = { parentequation = equation } ,
5592         }
5593     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5594 }
5595 \zref@ifpropundefined { subeq }
5596 { \zref@newprop { subeq } { \alph { equation } } }
5597 { }
5598 \clist_map_inline:nn
5599 {
5600     dmath ,
5601     dseries ,
5602     darray ,
5603 }
5604 {
5605     \AddToHook { env / #1 / begin }
5606     {
5607         \__zrefclever_zcsetup:n { currentcounter = equation }

```

```

5608         \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5609         { \zref@localaddprop \ZREF@mainlist { subeq } }
5610     }
5611 }
5612 \msg_info:nnn { zref-clever } { compat-package } { breqn }
5613 }
5614 }

```

## 9.8 listings

```

5615 \__zrefclever_compat_module:nn { listings }
5616 {
5617     \__zrefclever_if_package_loaded:nT { listings }
5618     {
5619         \__zrefclever_zcsetup:n
5620         {
5621             countertype =
5622             {
5623                 lstlisting = listing ,
5624                 lstnumber = line ,
5625             } ,
5626             counterresetby = { lstnumber = lstlisting } ,
5627         }

```

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.

```

5628     \lst@AddToHook { PreInit }
5629     { \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.

```

5630     \lst@AddToHook { Init }
5631     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5632     \msg_info:nnn { zref-clever } { compat-package } { listings }
5633 }
5634 }

```

## 9.9 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  $\{\langle max-depth \rangle\}$ . `\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.

```

5635 \__zrefclever_compat_module:nn { enumitem }
5636 {
5637   \__zrefclever_if_package_loaded:nT { enumitem }
5638   {
5639     \int_set:Nn \l_tmpa_int { 5 }
5640     \bool_while_do:nn
5641     {
5642       \cs_if_exist_p:c
5643       { c@ enum \int_to_roman:n { \l_tmpa_int } }
5644     }
5645     {
5646       \__zrefclever_zcsetup:x
5647       {
5648         counterresetby =
5649         {
5650           enum \int_to_roman:n { \l_tmpa_int } =
5651           enum \int_to_roman:n { \l_tmpa_int - 1 }
5652         } ,
5653         countertype =
5654         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5655       }
5656       \int_incr:N \l_tmpa_int
5657     }
5658     \int_compare:nNnT { \l_tmpa_int } > { 5 }
5659     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5660   }
5661 }

```

## 9.10 subcaption

```

5662 \__zrefclever_compat_module:nn { subcaption }
5663 {
5664   \__zrefclever_if_package_loaded:nT { subcaption }
5665   {
5666     \__zrefclever_zcsetup:n
5667     {
5668       countertype =
5669       {
5670         subfigure = figure ,
5671         subtable = table ,
5672       } ,
5673       counterresetby =
5674       {

```

```

5675         subfigure = figure ,
5676         subtable = table ,
5677     } ,
5678 }

```

Support for subref reference.

```

5679     \zref@newprop { subref }
5680     { \cs_if_exist_use:c { thesub \@capttype } }
5681     \tl_put_right:Nn \caption@subtypetool
5682     { \zref@localaddprop \ZREF@mainlist { subref } }
5683 }
5684 }

```

## 9.11 subfig

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

```

5685 \__zrefclever_compat_module:nn { subfig }
5686 {
5687     \__zrefclever_if_package_loaded:nT { subfig }
5688     {
5689         \__zrefclever_zcsetup:n
5690         {
5691             countertype =
5692             {
5693                 subfigure = figure ,
5694                 subtable = table ,
5695             } ,
5696             counterresetby =
5697             {
5698                 subfigure = figure ,
5699                 subtable = table ,
5700             } ,
5701         }
5702     }
5703 }
5704 </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  $\text{\LaTeX}$  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`name. 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.

## 10.2 English

English language file has been initially provided by the author.

```

5705 <*package>
5706 \zcDeclareLanguage { english }
5707 \zcDeclareLanguageAlias { american } { english }
5708 \zcDeclareLanguageAlias { australian } { english }
5709 \zcDeclareLanguageAlias { british } { english }
5710 \zcDeclareLanguageAlias { canadian } { english }
5711 \zcDeclareLanguageAlias { newzealand } { english }
5712 \zcDeclareLanguageAlias { UKenglish } { english }
5713 \zcDeclareLanguageAlias { USenglish } { english }
5714 </package>

5715 <*lang-english>

5716 namesep = {\nobreakspace} ,
5717 pairsep = {\and\nobreakspace} ,
5718 listsep = {,~} ,
5719 lastsep = {\and\nobreakspace} ,
5720 tpairsep = {\and\nobreakspace} ,
5721 tlistsep = {,~} ,
5722 tlastsep = {\and\nobreakspace} ,
5723 notesep = {~} ,
5724 rangesep = {\to\nobreakspace} ,
5725
5726 type = book ,
5727   Name-sg = Book ,
5728   name-sg = book ,
5729   Name-pl = Books ,
5730   name-pl = books ,
5731
5732 type = part ,
5733   Name-sg = Part ,
5734   name-sg = part ,
5735   Name-pl = Parts ,

```

```

5736     name-pl = parts ,
5737
5738 type = chapter ,
5739     Name-sg = Chapter ,
5740     name-sg = chapter ,
5741     Name-pl = Chapters ,
5742     name-pl = chapters ,
5743
5744 type = section ,
5745     Name-sg = Section ,
5746     name-sg = section ,
5747     Name-pl = Sections ,
5748     name-pl = sections ,
5749
5750 type = paragraph ,
5751     Name-sg = Paragraph ,
5752     name-sg = paragraph ,
5753     Name-pl = Paragraphs ,
5754     name-pl = paragraphs ,
5755     Name-sg-ab = Par. ,
5756     name-sg-ab = par. ,
5757     Name-pl-ab = Par. ,
5758     name-pl-ab = par. ,
5759
5760 type = appendix ,
5761     Name-sg = Appendix ,
5762     name-sg = appendix ,
5763     Name-pl = Appendices ,
5764     name-pl = appendices ,
5765
5766 type = page ,
5767     Name-sg = Page ,
5768     name-sg = page ,
5769     Name-pl = Pages ,
5770     name-pl = pages ,
5771     rangesep = {\textendash} ,
5772     rangetopair = false ,
5773
5774 type = line ,
5775     Name-sg = Line ,
5776     name-sg = line ,
5777     Name-pl = Lines ,
5778     name-pl = lines ,
5779
5780 type = figure ,
5781     Name-sg = Figure ,
5782     name-sg = figure ,
5783     Name-pl = Figures ,
5784     name-pl = figures ,
5785     Name-sg-ab = Fig. ,
5786     name-sg-ab = fig. ,
5787     Name-pl-ab = Figs. ,
5788     name-pl-ab = figs. ,
5789

```

```

5790 type = table ,
5791     Name-sg = Table ,
5792     name-sg = table ,
5793     Name-pl = Tables ,
5794     name-pl = tables ,
5795
5796 type = item ,
5797     Name-sg = Item ,
5798     name-sg = item ,
5799     Name-pl = Items ,
5800     name-pl = items ,
5801
5802 type = footnote ,
5803     Name-sg = Footnote ,
5804     name-sg = footnote ,
5805     Name-pl = Footnotes ,
5806     name-pl = footnotes ,
5807
5808 type = endnote ,
5809     Name-sg = Note ,
5810     name-sg = note ,
5811     Name-pl = Notes ,
5812     name-pl = notes ,
5813
5814 type = note ,
5815     Name-sg = Note ,
5816     name-sg = note ,
5817     Name-pl = Notes ,
5818     name-pl = notes ,
5819
5820 type = equation ,
5821     Name-sg = Equation ,
5822     name-sg = equation ,
5823     Name-pl = Equations ,
5824     name-pl = equations ,
5825     Name-sg-ab = Eq. ,
5826     name-sg-ab = eq. ,
5827     Name-pl-ab = Eqs. ,
5828     name-pl-ab = eqs. ,
5829     refbounds-first-sg = {,(,)}, ,
5830     refbounds = {(,,)} ,
5831
5832 type = theorem ,
5833     Name-sg = Theorem ,
5834     name-sg = theorem ,
5835     Name-pl = Theorems ,
5836     name-pl = theorems ,
5837
5838 type = lemma ,
5839     Name-sg = Lemma ,
5840     name-sg = lemma ,
5841     Name-pl = Lemmas ,
5842     name-pl = lemmas ,
5843

```

```

5844 type = corollary ,
5845     Name-sg = Corollary ,
5846     name-sg = corollary ,
5847     Name-pl = Corollaries ,
5848     name-pl = corollaries ,
5849
5850 type = proposition ,
5851     Name-sg = Proposition ,
5852     name-sg = proposition ,
5853     Name-pl = Propositions ,
5854     name-pl = propositions ,
5855
5856 type = definition ,
5857     Name-sg = Definition ,
5858     name-sg = definition ,
5859     Name-pl = Definitions ,
5860     name-pl = definitions ,
5861
5862 type = proof ,
5863     Name-sg = Proof ,
5864     name-sg = proof ,
5865     Name-pl = Proofs ,
5866     name-pl = proofs ,
5867
5868 type = result ,
5869     Name-sg = Result ,
5870     name-sg = result ,
5871     Name-pl = Results ,
5872     name-pl = results ,
5873
5874 type = remark ,
5875     Name-sg = Remark ,
5876     name-sg = remark ,
5877     Name-pl = Remarks ,
5878     name-pl = remarks ,
5879
5880 type = example ,
5881     Name-sg = Example ,
5882     name-sg = example ,
5883     Name-pl = Examples ,
5884     name-pl = examples ,
5885
5886 type = algorithm ,
5887     Name-sg = Algorithm ,
5888     name-sg = algorithm ,
5889     Name-pl = Algorithms ,
5890     name-pl = algorithms ,
5891
5892 type = listing ,
5893     Name-sg = Listing ,
5894     name-sg = listing ,
5895     Name-pl = Listings ,
5896     name-pl = listings ,
5897

```



```

5898 type = exercise ,
5899   Name-sg = Exercise ,
5900   name-sg = exercise ,
5901   Name-pl = Exercises ,
5902   name-pl = exercises ,
5903
5904 type = solution ,
5905   Name-sg = Solution ,
5906   name-sg = solution ,
5907   Name-pl = Solutions ,
5908   name-pl = solutions ,
5909 </lang-english>

```

### 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`.

```

5910 <*package>
5911 \zcDeclareLanguage
5912 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5913 { german }
5914 \zcDeclareLanguageAlias { ngerman } { german }
5915 \zcDeclareLanguageAlias { austrian } { german }
5916 \zcDeclareLanguageAlias { naustrian } { german }
5917 \zcDeclareLanguageAlias { swissgerman } { german }
5918 \zcDeclareLanguageAlias { nswissgerman } { german }
5919 </package>
5920 <*lang-german>
5921 namesep = {\nobreakspace} ,
5922 pairsep = {\und\nobreakspace} ,
5923 listsep = { , ~ } ,
5924 lastsep = {\und\nobreakspace} ,
5925 tpairsep = {\und\nobreakspace} ,
5926 tlistsep = { , ~ } ,
5927 tlastsep = {\und\nobreakspace} ,
5928 notesep = { ~ } ,
5929 rangesep = { ~bis\nobreakspace} ,
5930
5931 type = book ,
5932 gender = n ,
5933 case = N ,
5934   Name-sg = Buch ,
5935   Name-pl = Bücher ,
5936 case = A ,
5937   Name-sg = Buch ,
5938   Name-pl = Bücher ,
5939 case = D ,
5940   Name-sg = Buch ,
5941   Name-pl = Büchern ,
5942 case = G ,
5943   Name-sg = Buches ,

```

```

5944     Name-pl = Bücher ,
5945
5946 type = part ,
5947     gender = m ,
5948     case = N ,
5949     Name-sg = Teil ,
5950     Name-pl = Teile ,
5951     case = A ,
5952     Name-sg = Teil ,
5953     Name-pl = Teile ,
5954     case = D ,
5955     Name-sg = Teil ,
5956     Name-pl = Teilen ,
5957     case = G ,
5958     Name-sg = Teiles ,
5959     Name-pl = Teile ,
5960
5961 type = chapter ,
5962     gender = n ,
5963     case = N ,
5964     Name-sg = Kapitel ,
5965     Name-pl = Kapitel ,
5966     case = A ,
5967     Name-sg = Kapitel ,
5968     Name-pl = Kapitel ,
5969     case = D ,
5970     Name-sg = Kapitel ,
5971     Name-pl = Kapiteln ,
5972     case = G ,
5973     Name-sg = Kapitels ,
5974     Name-pl = Kapitel ,
5975
5976 type = section ,
5977     gender = m ,
5978     case = N ,
5979     Name-sg = Abschnitt ,
5980     Name-pl = Abschnitte ,
5981     case = A ,
5982     Name-sg = Abschnitt ,
5983     Name-pl = Abschnitte ,
5984     case = D ,
5985     Name-sg = Abschnitt ,
5986     Name-pl = Abschnitten ,
5987     case = G ,
5988     Name-sg = Abschnitts ,
5989     Name-pl = Abschnitte ,
5990
5991 type = paragraph ,
5992     gender = m ,
5993     case = N ,
5994     Name-sg = Absatz ,
5995     Name-pl = Absätze ,
5996     case = A ,
5997     Name-sg = Absatz ,

```

```

5998     Name-pl = Absätze ,
5999     case = D ,
6000     Name-sg = Absatz ,
6001     Name-pl = Absätzen ,
6002     case = G ,
6003     Name-sg = Absatzes ,
6004     Name-pl = Absätze ,
6005
6006 type = appendix ,
6007     gender = m ,
6008     case = N ,
6009     Name-sg = Anhang ,
6010     Name-pl = Anhänge ,
6011     case = A ,
6012     Name-sg = Anhang ,
6013     Name-pl = Anhänge ,
6014     case = D ,
6015     Name-sg = Anhang ,
6016     Name-pl = Anhängen ,
6017     case = G ,
6018     Name-sg = Anhangs ,
6019     Name-pl = Anhänge ,
6020
6021 type = page ,
6022     gender = f ,
6023     case = N ,
6024     Name-sg = Seite ,
6025     Name-pl = Seiten ,
6026     case = A ,
6027     Name-sg = Seite ,
6028     Name-pl = Seiten ,
6029     case = D ,
6030     Name-sg = Seite ,
6031     Name-pl = Seiten ,
6032     case = G ,
6033     Name-sg = Seite ,
6034     Name-pl = Seiten ,
6035     rangesep = {\textendash} ,
6036     rangetopair = false ,
6037
6038 type = line ,
6039     gender = f ,
6040     case = N ,
6041     Name-sg = Zeile ,
6042     Name-pl = Zeilen ,
6043     case = A ,
6044     Name-sg = Zeile ,
6045     Name-pl = Zeilen ,
6046     case = D ,
6047     Name-sg = Zeile ,
6048     Name-pl = Zeilen ,
6049     case = G ,
6050     Name-sg = Zeile ,
6051     Name-pl = Zeilen ,

```

```

6052
6053 type = figure ,
6054     gender = f ,
6055     case = N ,
6056         Name-sg = Abbildung ,
6057         Name-pl = Abbildungen ,
6058         Name-sg-ab = Abb. ,
6059         Name-pl-ab = Abb. ,
6060     case = A ,
6061         Name-sg = Abbildung ,
6062         Name-pl = Abbildungen ,
6063         Name-sg-ab = Abb. ,
6064         Name-pl-ab = Abb. ,
6065     case = D ,
6066         Name-sg = Abbildung ,
6067         Name-pl = Abbildungen ,
6068         Name-sg-ab = Abb. ,
6069         Name-pl-ab = Abb. ,
6070     case = G ,
6071         Name-sg = Abbildung ,
6072         Name-pl = Abbildungen ,
6073         Name-sg-ab = Abb. ,
6074         Name-pl-ab = Abb. ,
6075
6076 type = table ,
6077     gender = f ,
6078     case = N ,
6079         Name-sg = Tabelle ,
6080         Name-pl = Tabellen ,
6081     case = A ,
6082         Name-sg = Tabelle ,
6083         Name-pl = Tabellen ,
6084     case = D ,
6085         Name-sg = Tabelle ,
6086         Name-pl = Tabellen ,
6087     case = G ,
6088         Name-sg = Tabelle ,
6089         Name-pl = Tabellen ,
6090
6091 type = item ,
6092     gender = m ,
6093     case = N ,
6094         Name-sg = Punkt ,
6095         Name-pl = Punkte ,
6096     case = A ,
6097         Name-sg = Punkt ,
6098         Name-pl = Punkte ,
6099     case = D ,
6100         Name-sg = Punkt ,
6101         Name-pl = Punkten ,
6102     case = G ,
6103         Name-sg = Punktes ,
6104         Name-pl = Punkte ,
6105

```

```

6106 type = footnote ,
6107     gender = f ,
6108     case = N ,
6109         Name-sg = Fußnote ,
6110         Name-pl = Fußnoten ,
6111     case = A ,
6112         Name-sg = Fußnote ,
6113         Name-pl = Fußnoten ,
6114     case = D ,
6115         Name-sg = Fußnote ,
6116         Name-pl = Fußnoten ,
6117     case = G ,
6118         Name-sg = Fußnote ,
6119         Name-pl = Fußnoten ,
6120
6121 type = endnote ,
6122     gender = f ,
6123     case = N ,
6124         Name-sg = Endnote ,
6125         Name-pl = Endnoten ,
6126     case = A ,
6127         Name-sg = Endnote ,
6128         Name-pl = Endnoten ,
6129     case = D ,
6130         Name-sg = Endnote ,
6131         Name-pl = Endnoten ,
6132     case = G ,
6133         Name-sg = Endnote ,
6134         Name-pl = Endnoten ,
6135
6136 type = note ,
6137     gender = f ,
6138     case = N ,
6139         Name-sg = Anmerkung ,
6140         Name-pl = Anmerkungen ,
6141     case = A ,
6142         Name-sg = Anmerkung ,
6143         Name-pl = Anmerkungen ,
6144     case = D ,
6145         Name-sg = Anmerkung ,
6146         Name-pl = Anmerkungen ,
6147     case = G ,
6148         Name-sg = Anmerkung ,
6149         Name-pl = Anmerkungen ,
6150
6151 type = equation ,
6152     gender = f ,
6153     case = N ,
6154         Name-sg = Gleichung ,
6155         Name-pl = Gleichungen ,
6156     case = A ,
6157         Name-sg = Gleichung ,
6158         Name-pl = Gleichungen ,
6159     case = D ,

```

```

6160     Name-sg = Gleichung ,
6161     Name-pl = Gleichungen ,
6162     case = G ,
6163     Name-sg = Gleichung ,
6164     Name-pl = Gleichungen ,
6165     refbounds-first-sg = {,(,),} ,
6166     refbounds = {(,,)} ,
6167
6168 type = theorem ,
6169     gender = n ,
6170     case = N ,
6171     Name-sg = Theorem ,
6172     Name-pl = Theoreme ,
6173     case = A ,
6174     Name-sg = Theorem ,
6175     Name-pl = Theoreme ,
6176     case = D ,
6177     Name-sg = Theorem ,
6178     Name-pl = Theoremen ,
6179     case = G ,
6180     Name-sg = Theorems ,
6181     Name-pl = Theoreme ,
6182
6183 type = lemma ,
6184     gender = n ,
6185     case = N ,
6186     Name-sg = Lemma ,
6187     Name-pl = Lemmata ,
6188     case = A ,
6189     Name-sg = Lemma ,
6190     Name-pl = Lemmata ,
6191     case = D ,
6192     Name-sg = Lemma ,
6193     Name-pl = Lemmata ,
6194     case = G ,
6195     Name-sg = Lemmas ,
6196     Name-pl = Lemmata ,
6197
6198 type = corollary ,
6199     gender = n ,
6200     case = N ,
6201     Name-sg = Korollar ,
6202     Name-pl = Korollare ,
6203     case = A ,
6204     Name-sg = Korollar ,
6205     Name-pl = Korollare ,
6206     case = D ,
6207     Name-sg = Korollar ,
6208     Name-pl = Korollaren ,
6209     case = G ,
6210     Name-sg = Korollars ,
6211     Name-pl = Korollare ,
6212
6213 type = proposition ,

```

```

6214 gender = m ,
6215 case = N ,
6216     Name-sg = Satz ,
6217     Name-pl = Sätze ,
6218 case = A ,
6219     Name-sg = Satz ,
6220     Name-pl = Sätze ,
6221 case = D ,
6222     Name-sg = Satz ,
6223     Name-pl = Sätzen ,
6224 case = G ,
6225     Name-sg = Satzes ,
6226     Name-pl = Sätze ,
6227
6228 type = definition ,
6229 gender = f ,
6230 case = N ,
6231     Name-sg = Definition ,
6232     Name-pl = Definitionen ,
6233 case = A ,
6234     Name-sg = Definition ,
6235     Name-pl = Definitionen ,
6236 case = D ,
6237     Name-sg = Definition ,
6238     Name-pl = Definitionen ,
6239 case = G ,
6240     Name-sg = Definition ,
6241     Name-pl = Definitionen ,
6242
6243 type = proof ,
6244 gender = m ,
6245 case = N ,
6246     Name-sg = Beweis ,
6247     Name-pl = Beweise ,
6248 case = A ,
6249     Name-sg = Beweis ,
6250     Name-pl = Beweise ,
6251 case = D ,
6252     Name-sg = Beweis ,
6253     Name-pl = Beweisen ,
6254 case = G ,
6255     Name-sg = Beweises ,
6256     Name-pl = Beweise ,
6257
6258 type = result ,
6259 gender = n ,
6260 case = N ,
6261     Name-sg = Ergebnis ,
6262     Name-pl = Ergebnisse ,
6263 case = A ,
6264     Name-sg = Ergebnis ,
6265     Name-pl = Ergebnisse ,
6266 case = D ,
6267     Name-sg = Ergebnis ,

```

```

6268     Name-pl = Ergebnissen ,
6269     case = G ,
6270     Name-sg = Ergebnisses ,
6271     Name-pl = Ergebnisse ,
6272
6273 type = remark ,
6274     gender = f ,
6275     case = N ,
6276     Name-sg = Bemerkung ,
6277     Name-pl = Bemerkungen ,
6278     case = A ,
6279     Name-sg = Bemerkung ,
6280     Name-pl = Bemerkungen ,
6281     case = D ,
6282     Name-sg = Bemerkung ,
6283     Name-pl = Bemerkungen ,
6284     case = G ,
6285     Name-sg = Bemerkung ,
6286     Name-pl = Bemerkungen ,
6287
6288 type = example ,
6289     gender = n ,
6290     case = N ,
6291     Name-sg = Beispiel ,
6292     Name-pl = Beispiele ,
6293     case = A ,
6294     Name-sg = Beispiel ,
6295     Name-pl = Beispiele ,
6296     case = D ,
6297     Name-sg = Beispiel ,
6298     Name-pl = Beispielen ,
6299     case = G ,
6300     Name-sg = Beispiels ,
6301     Name-pl = Beispiele ,
6302
6303 type = algorithm ,
6304     gender = m ,
6305     case = N ,
6306     Name-sg = Algorithmus ,
6307     Name-pl = Algorithmen ,
6308     case = A ,
6309     Name-sg = Algorithmus ,
6310     Name-pl = Algorithmen ,
6311     case = D ,
6312     Name-sg = Algorithmus ,
6313     Name-pl = Algorithmen ,
6314     case = G ,
6315     Name-sg = Algorithmus ,
6316     Name-pl = Algorithmen ,
6317
6318 type = listing ,
6319     gender = n ,
6320     case = N ,
6321     Name-sg = Listing ,

```



```

6322     Name-pl = Listings ,
6323     case = A ,
6324     Name-sg = Listing ,
6325     Name-pl = Listings ,
6326     case = D ,
6327     Name-sg = Listing ,
6328     Name-pl = Listings ,
6329     case = G ,
6330     Name-sg = Listings ,
6331     Name-pl = Listings ,
6332
6333 type = exercise ,
6334 gender = f ,
6335 case = N ,
6336     Name-sg = Übungsaufgabe ,
6337     Name-pl = Übungsaufgaben ,
6338 case = A ,
6339     Name-sg = Übungsaufgabe ,
6340     Name-pl = Übungsaufgaben ,
6341 case = D ,
6342     Name-sg = Übungsaufgabe ,
6343     Name-pl = Übungsaufgaben ,
6344 case = G ,
6345     Name-sg = Übungsaufgabe ,
6346     Name-pl = Übungsaufgaben ,
6347
6348 type = solution ,
6349 gender = f ,
6350 case = N ,
6351     Name-sg = Lösung ,
6352     Name-pl = Lösungen ,
6353 case = A ,
6354     Name-sg = Lösung ,
6355     Name-pl = Lösungen ,
6356 case = D ,
6357     Name-sg = Lösung ,
6358     Name-pl = Lösungen ,
6359 case = G ,
6360     Name-sg = Lösung ,
6361     Name-pl = Lösungen ,
6362 </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<sub>E</sub>X (GUTenberg) (at [https://groups.google.com/g/gut\\_fr/c/rNLm6weGcyg](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 `français`, `frenchb`, and `canadien`, but they are deprecated as options and, if used, they fall back to either `french` or `acadian`.

```

6363 <*package>

```

```

6364 \zcDeclareLanguage [ gender = { f , m } ] { french }
6365 \zcDeclareLanguageAlias { acadian } { french }
6366 \package
6367 (*lang-french)

6368 namesep = {\nobreakspace} ,
6369 pairsep = {\~et\nobreakspace} ,
6370 listsep = { , ~ } ,
6371 lastsep = {\~et\nobreakspace} ,
6372 tpairsep = {\~et\nobreakspace} ,
6373 tlistsep = { , ~ } ,
6374 tlastsep = {\~et\nobreakspace} ,
6375 notesep = { ~ } ,
6376 rangesep = {\~à\nobreakspace} ,
6377
6378 type = book ,
6379   gender = m ,
6380   Name-sg = Livre ,
6381   name-sg = livre ,
6382   Name-pl = Livres ,
6383   name-pl = livres ,
6384
6385 type = part ,
6386   gender = f ,
6387   Name-sg = Partie ,
6388   name-sg = partie ,
6389   Name-pl = Parties ,
6390   name-pl = parties ,
6391
6392 type = chapter ,
6393   gender = m ,
6394   Name-sg = Chapitre ,
6395   name-sg = chapitre ,
6396   Name-pl = Chapitres ,
6397   name-pl = chapitres ,
6398
6399 type = section ,
6400   gender = f ,
6401   Name-sg = Section ,
6402   name-sg = section ,
6403   Name-pl = Sections ,
6404   name-pl = sections ,
6405
6406 type = paragraph ,
6407   gender = m ,
6408   Name-sg = Paragraphe ,
6409   name-sg = paragraphe ,
6410   Name-pl = Paragraphes ,
6411   name-pl = paragraphes ,
6412
6413 type = appendix ,
6414   gender = f ,
6415   Name-sg = Annexe ,
6416   name-sg = annexe ,

```

```

6417     Name-pl = Annexes ,
6418     name-pl = annexes ,
6419
6420     type = page ,
6421     gender = f ,
6422     Name-sg = Page ,
6423     name-sg = page ,
6424     Name-pl = Pages ,
6425     name-pl = pages ,
6426     rangesep = {-} ,
6427     rangetopair = false ,
6428
6429     type = line ,
6430     gender = f ,
6431     Name-sg = Ligne ,
6432     name-sg = ligne ,
6433     Name-pl = Lignes ,
6434     name-pl = lignes ,
6435
6436     type = figure ,
6437     gender = f ,
6438     Name-sg = Figure ,
6439     name-sg = figure ,
6440     Name-pl = Figures ,
6441     name-pl = figures ,
6442
6443     type = table ,
6444     gender = f ,
6445     Name-sg = Table ,
6446     name-sg = table ,
6447     Name-pl = Tables ,
6448     name-pl = tables ,
6449
6450     type = item ,
6451     gender = m ,
6452     Name-sg = Point ,
6453     name-sg = point ,
6454     Name-pl = Points ,
6455     name-pl = points ,
6456
6457     type = footnote ,
6458     gender = f ,
6459     Name-sg = Note ,
6460     name-sg = note ,
6461     Name-pl = Notes ,
6462     name-pl = notes ,
6463
6464     type = endnote ,
6465     gender = f ,
6466     Name-sg = Note ,
6467     name-sg = note ,
6468     Name-pl = Notes ,
6469     name-pl = notes ,
6470

```

```

6471 type = note ,
6472     gender = f ,
6473     Name-sg = Note ,
6474     name-sg = note ,
6475     Name-pl = Notes ,
6476     name-pl = notes ,
6477
6478 type = equation ,
6479     gender = f ,
6480     Name-sg = Équation ,
6481     name-sg = équation ,
6482     Name-pl = Équations ,
6483     name-pl = équations ,
6484     refbounds-first-sg = {(,),(,)},
6485     refbounds = {(,),(,),(,)} ,
6486
6487 type = theorem ,
6488     gender = m ,
6489     Name-sg = Théorème ,
6490     name-sg = théorème ,
6491     Name-pl = Théorèmes ,
6492     name-pl = théorèmes ,
6493
6494 type = lemma ,
6495     gender = m ,
6496     Name-sg = Lemme ,
6497     name-sg = lemme ,
6498     Name-pl = Lemmes ,
6499     name-pl = lemmes ,
6500
6501 type = corollary ,
6502     gender = m ,
6503     Name-sg = Corollaire ,
6504     name-sg = corollaire ,
6505     Name-pl = Corollaires ,
6506     name-pl = corollaires ,
6507
6508 type = proposition ,
6509     gender = f ,
6510     Name-sg = Proposition ,
6511     name-sg = proposition ,
6512     Name-pl = Propositions ,
6513     name-pl = propositions ,
6514
6515 type = definition ,
6516     gender = f ,
6517     Name-sg = Définition ,
6518     name-sg = définition ,
6519     Name-pl = Définitions ,
6520     name-pl = définitions ,
6521
6522 type = proof ,
6523     gender = f ,
6524     Name-sg = Démonstration ,

```

```

6525     name-sg = démonstration ,
6526     Name-pl = Démonstrations ,
6527     name-pl = démonstrations ,
6528
6529     type = result ,
6530     gender = m ,
6531     Name-sg = Résultat ,
6532     name-sg = résultat ,
6533     Name-pl = Résultats ,
6534     name-pl = résultats ,
6535
6536     type = remark ,
6537     gender = f ,
6538     Name-sg = Remarque ,
6539     name-sg = remarque ,
6540     Name-pl = Remarques ,
6541     name-pl = remarques ,
6542
6543     type = example ,
6544     gender = m ,
6545     Name-sg = Exemple ,
6546     name-sg = exemple ,
6547     Name-pl = Exemples ,
6548     name-pl = exemples ,
6549
6550     type = algorithm ,
6551     gender = m ,
6552     Name-sg = Algorithme ,
6553     name-sg = algorithme ,
6554     Name-pl = Algorithmes ,
6555     name-pl = algorithmes ,
6556
6557     type = listing ,
6558     gender = m ,
6559     Name-sg = Listing ,
6560     name-sg = listing ,
6561     Name-pl = Listings ,
6562     name-pl = listings ,
6563
6564     type = exercise ,
6565     gender = m ,
6566     Name-sg = Exercice ,
6567     name-sg = exercice ,
6568     Name-pl = Exercices ,
6569     name-pl = exercices ,
6570
6571     type = solution ,
6572     gender = f ,
6573     Name-sg = Solution ,
6574     name-sg = solution ,
6575     Name-pl = Solutions ,
6576     name-pl = solutions ,
6577 </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.

```
6578 <*package>
6579 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6580 \zcDeclareLanguageAlias { brazilian } { portuguese }
6581 \zcDeclareLanguageAlias { brazil   } { portuguese }
6582 \zcDeclareLanguageAlias { portuges } { portuguese }
6583 </package>

6584 <*lang-portuguese>

6585 namesep = {\nobreakspace} ,
6586 pairsep  = {\nobreakspace} ,
6587 listsep  = {,~} ,
6588 lastsep  = {\nobreakspace} ,
6589 tpairsep = {\nobreakspace} ,
6590 tlistsep = {,~} ,
6591 tlastsep = {\nobreakspace} ,
6592 notesep  = {~} ,
6593 rangesep  = {\nobreakspace} ,
6594
6595 type = book ,
6596   gender = m ,
6597   Name-sg = Livro ,
6598   name-sg  = livro ,
6599   Name-pl  = Livros ,
6600   name-pl  = livros ,
6601
6602 type = part ,
6603   gender = f ,
6604   Name-sg = Parte ,
6605   name-sg = parte ,
6606   Name-pl = Partes ,
6607   name-pl = partes ,
6608
6609 type = chapter ,
6610   gender = m ,
6611   Name-sg = Capítulo ,
6612   name-sg = capítulo ,
6613   Name-pl = Capítulos ,
6614   name-pl = capítulos ,
6615
6616 type = section ,
6617   gender = f ,
6618   Name-sg = Seção ,
6619   name-sg = seção ,
6620   Name-pl = Seções ,
6621   name-pl = seções ,
6622
6623 type = paragraph ,
6624   gender = m ,
6625   Name-sg = Parágrafo ,
```

```

6626 name-sg = parágrafo ,
6627 Name-pl = Parágrafos ,
6628 name-pl = parágrafos ,
6629 Name-sg-ab = Par. ,
6630 name-sg-ab = par. ,
6631 Name-pl-ab = Par. ,
6632 name-pl-ab = par. ,
6633
6634 type = appendix ,
6635 gender = m ,
6636 Name-sg = Apêndice ,
6637 name-sg = apêndice ,
6638 Name-pl = Apêndices ,
6639 name-pl = apêndices ,
6640
6641 type = page ,
6642 gender = f ,
6643 Name-sg = Página ,
6644 name-sg = página ,
6645 Name-pl = Páginas ,
6646 name-pl = páginas ,
6647 rangesep = {\textendash} ,
6648 rangetopair = false ,
6649
6650 type = line ,
6651 gender = f ,
6652 Name-sg = Linha ,
6653 name-sg = linha ,
6654 Name-pl = Linhas ,
6655 name-pl = linhas ,
6656
6657 type = figure ,
6658 gender = f ,
6659 Name-sg = Figura ,
6660 name-sg = figura ,
6661 Name-pl = Figuras ,
6662 name-pl = figuras ,
6663 Name-sg-ab = Fig. ,
6664 name-sg-ab = fig. ,
6665 Name-pl-ab = Figs. ,
6666 name-pl-ab = figs. ,
6667
6668 type = table ,
6669 gender = f ,
6670 Name-sg = Tabela ,
6671 name-sg = tabela ,
6672 Name-pl = Tabelas ,
6673 name-pl = tabelas ,
6674
6675 type = item ,
6676 gender = m ,
6677 Name-sg = Item ,
6678 name-sg = item ,
6679 Name-pl = Itens ,

```

```

6680     name-pl = itens ,
6681
6682 type = footnote ,
6683     gender = f ,
6684     Name-sg = Nota ,
6685     name-sg = nota ,
6686     Name-pl = Notas ,
6687     name-pl = notas ,
6688
6689 type = endnote ,
6690     gender = f ,
6691     Name-sg = Nota ,
6692     name-sg = nota ,
6693     Name-pl = Notas ,
6694     name-pl = notas ,
6695
6696 type = note ,
6697     gender = f ,
6698     Name-sg = Nota ,
6699     name-sg = nota ,
6700     Name-pl = Notas ,
6701     name-pl = notas ,
6702
6703 type = equation ,
6704     gender = f ,
6705     Name-sg = Equação ,
6706     name-sg = equação ,
6707     Name-pl = Equações ,
6708     name-pl = equações ,
6709     Name-sg-ab = Eq. ,
6710     name-sg-ab = eq. ,
6711     Name-pl-ab = Eqs. ,
6712     name-pl-ab = eqs. ,
6713     refbounds-first-sg = {,(,),} ,
6714     refbounds = {(,,)} ,
6715
6716 type = theorem ,
6717     gender = m ,
6718     Name-sg = Teorema ,
6719     name-sg = teorema ,
6720     Name-pl = Teoremas ,
6721     name-pl = teoremas ,
6722
6723 type = lemma ,
6724     gender = m ,
6725     Name-sg = Lema ,
6726     name-sg = lema ,
6727     Name-pl = Lemas ,
6728     name-pl = lemas ,
6729
6730 type = corollary ,
6731     gender = m ,
6732     Name-sg = Corolário ,
6733     name-sg = corolário ,

```



```

6734     Name-pl = Corolários ,
6735     name-pl = corolários ,
6736
6737     type = proposition ,
6738     gender = f ,
6739     Name-sg = Proposição ,
6740     name-sg = proposição ,
6741     Name-pl = Proposições ,
6742     name-pl = proposições ,
6743
6744     type = definition ,
6745     gender = f ,
6746     Name-sg = Definição ,
6747     name-sg = definição ,
6748     Name-pl = Definições ,
6749     name-pl = definições ,
6750
6751     type = proof ,
6752     gender = f ,
6753     Name-sg = Demonstração ,
6754     name-sg = demonstração ,
6755     Name-pl = Demonstrações ,
6756     name-pl = demonstrações ,
6757
6758     type = result ,
6759     gender = m ,
6760     Name-sg = Resultado ,
6761     name-sg = resultado ,
6762     Name-pl = Resultados ,
6763     name-pl = resultados ,
6764
6765     type = remark ,
6766     gender = f ,
6767     Name-sg = Observação ,
6768     name-sg = observação ,
6769     Name-pl = Observações ,
6770     name-pl = observações ,
6771
6772     type = example ,
6773     gender = m ,
6774     Name-sg = Exemplo ,
6775     name-sg = exemplo ,
6776     Name-pl = Exemplos ,
6777     name-pl = exemplos ,
6778
6779     type = algorithm ,
6780     gender = m ,
6781     Name-sg = Algoritmo ,
6782     name-sg = algoritmo ,
6783     Name-pl = Algoritmos ,
6784     name-pl = algoritmos ,
6785
6786     type = listing ,
6787     gender = f ,

```

```

6788   Name-sg = Listagem ,
6789   name-sg = listagem ,
6790   Name-pl = Listagens ,
6791   name-pl = listagens ,
6792
6793   type = exercise ,
6794   gender = m ,
6795   Name-sg = Exercício ,
6796   name-sg = exercício ,
6797   Name-pl = Exercícios ,
6798   name-pl = exercícios ,
6799
6800   type = solution ,
6801   gender = f ,
6802   Name-sg = Solução ,
6803   name-sg = solução ,
6804   Name-pl = Soluções ,
6805   name-pl = soluções ,
6806 </lang-portuguese>

```

## 10.6 Spanish

Spanish language file has been initially provided by the author.

```

6807 <*package>
6808 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6809 </package>
6810 <*lang-spanish>
6811 namesep = {\nobreakspace} ,
6812 pairsep = {\~y\nobreakspace} ,
6813 listsep = {,~} ,
6814 lastsep = {\~y\nobreakspace} ,
6815 tpairsep = {\~y\nobreakspace} ,
6816 tlistsep = {,~} ,
6817 tlastsep = {\~y\nobreakspace} ,
6818 notesep = {\~} ,
6819 rangesep = {\~a\nobreakspace} ,
6820
6821 type = book ,
6822 gender = m ,
6823 Name-sg = Libro ,
6824 name-sg = libro ,
6825 Name-pl = Libros ,
6826 name-pl = libros ,
6827
6828 type = part ,
6829 gender = f ,
6830 Name-sg = Parte ,
6831 name-sg = parte ,
6832 Name-pl = Partes ,
6833 name-pl = partes ,
6834
6835 type = chapter ,

```

```

6836     gender = m ,
6837     Name-sg = Capítulo ,
6838     name-sg = capítulo ,
6839     Name-pl = Capítulos ,
6840     name-pl = capítulos ,
6841
6842     type = section ,
6843     gender = f ,
6844     Name-sg = Sección ,
6845     name-sg = sección ,
6846     Name-pl = Secciones ,
6847     name-pl = secciones ,
6848
6849     type = paragraph ,
6850     gender = m ,
6851     Name-sg = Párrafo ,
6852     name-sg = párrafo ,
6853     Name-pl = Párrafos ,
6854     name-pl = párrafos ,
6855
6856     type = appendix ,
6857     gender = m ,
6858     Name-sg = Apéndice ,
6859     name-sg = apéndice ,
6860     Name-pl = Apéndices ,
6861     name-pl = apéndices ,
6862
6863     type = page ,
6864     gender = f ,
6865     Name-sg = Página ,
6866     name-sg = página ,
6867     Name-pl = Páginas ,
6868     name-pl = páginas ,
6869     rangesep = {\textendash} ,
6870     rangetopair = false ,
6871
6872     type = line ,
6873     gender = f ,
6874     Name-sg = Línea ,
6875     name-sg = línea ,
6876     Name-pl = Líneas ,
6877     name-pl = líneas ,
6878
6879     type = figure ,
6880     gender = f ,
6881     Name-sg = Figura ,
6882     name-sg = figura ,
6883     Name-pl = Figuras ,
6884     name-pl = figuras ,
6885
6886     type = table ,
6887     gender = m ,
6888     Name-sg = Cuadro ,
6889     name-sg = cuadro ,

```

```

6890     Name-pl = Cuadros ,
6891     name-pl = cuadros ,
6892
6893     type = item ,
6894     gender = m ,
6895     Name-sg = Punto ,
6896     name-sg = punto ,
6897     Name-pl = Puntos ,
6898     name-pl = puntos ,
6899
6900     type = footnote ,
6901     gender = f ,
6902     Name-sg = Nota ,
6903     name-sg = nota ,
6904     Name-pl = Notas ,
6905     name-pl = notas ,
6906
6907     type = endnote ,
6908     gender = f ,
6909     Name-sg = Nota ,
6910     name-sg = nota ,
6911     Name-pl = Notas ,
6912     name-pl = notas ,
6913
6914     type = note ,
6915     gender = f ,
6916     Name-sg = Nota ,
6917     name-sg = nota ,
6918     Name-pl = Notas ,
6919     name-pl = notas ,
6920
6921     type = equation ,
6922     gender = f ,
6923     Name-sg = Ecuación ,
6924     name-sg = ecuación ,
6925     Name-pl = Ecuaciones ,
6926     name-pl = ecuaciones ,
6927     refbounds-first-sg = {,(,),} ,
6928     refbounds = {(,,)} ,
6929
6930     type = theorem ,
6931     gender = m ,
6932     Name-sg = Teorema ,
6933     name-sg = teorema ,
6934     Name-pl = Teoremas ,
6935     name-pl = teoremas ,
6936
6937     type = lemma ,
6938     gender = m ,
6939     Name-sg = Lema ,
6940     name-sg = lema ,
6941     Name-pl = Lemas ,
6942     name-pl = lemas ,
6943

```

```

6944 type = corollary ,
6945     gender = m ,
6946     Name-sg = Corolario ,
6947     name-sg = corolario ,
6948     Name-pl = Corolarios ,
6949     name-pl = corolarios ,
6950
6951 type = proposition ,
6952     gender = f ,
6953     Name-sg = Proposición ,
6954     name-sg = proposición ,
6955     Name-pl = Proposiciones ,
6956     name-pl = proposiciones ,
6957
6958 type = definition ,
6959     gender = f ,
6960     Name-sg = Definición ,
6961     name-sg = definición ,
6962     Name-pl = Definiciones ,
6963     name-pl = definiciones ,
6964
6965 type = proof ,
6966     gender = f ,
6967     Name-sg = Demostración ,
6968     name-sg = demostración ,
6969     Name-pl = Demostraciones ,
6970     name-pl = demostraciones ,
6971
6972 type = result ,
6973     gender = m ,
6974     Name-sg = Resultado ,
6975     name-sg = resultado ,
6976     Name-pl = Resultados ,
6977     name-pl = resultados ,
6978
6979 type = remark ,
6980     gender = f ,
6981     Name-sg = Observación ,
6982     name-sg = observación ,
6983     Name-pl = Observaciones ,
6984     name-pl = observaciones ,
6985
6986 type = example ,
6987     gender = m ,
6988     Name-sg = Ejemplo ,
6989     name-sg = ejemplo ,
6990     Name-pl = Ejemplos ,
6991     name-pl = ejemplos ,
6992
6993 type = algorithm ,
6994     gender = m ,
6995     Name-sg = Algoritmo ,
6996     name-sg = algoritmo ,
6997     Name-pl = Algoritmos ,

```

```

6998   name-pl = algoritmos ,
6999
7000 type = listing ,
7001   gender = m ,
7002   Name-sg = Listado ,
7003   name-sg = listado ,
7004   Name-pl = Listados ,
7005   name-pl = listados ,
7006
7007 type = exercise ,
7008   gender = m ,
7009   Name-sg = Ejercicio ,
7010   name-sg = ejercicio ,
7011   Name-pl = Ejercicios ,
7012   name-pl = ejercicios ,
7013
7014 type = solution ,
7015   gender = f ,
7016   Name-sg = Solución ,
7017   name-sg = solución ,
7018   Name-pl = Soluciones ,
7019   name-pl = soluciones ,
7020 </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.

```

7021 <*package>
7022 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7023 </package>
7024 <*lang-dutch>
7025 namesep   = {\nobreakspace} ,
7026 pairsep   = {\~en\nobreakspace} ,
7027 listsep    = {,~} ,
7028 lastsep    = {\~en\nobreakspace} ,
7029 tpairsep   = {\~en\nobreakspace} ,
7030 tlistsep   = {,~} ,
7031 tlastsep   = {\~en\nobreakspace} ,
7032 notesep    = {\~} ,
7033 rangesep   = {\~t/m\nobreakspace} ,
7034
7035 type = book ,
7036   gender = n ,
7037   Name-sg = Boek ,
7038   name-sg = boek ,
7039   Name-pl = Boeken ,
7040   name-pl = boeken ,
7041
7042 type = part ,
7043   gender = n ,
7044   Name-sg = Deel ,

```

```

7045 name-sg = deel ,
7046 Name-pl = Delen ,
7047 name-pl = delen ,
7048
7049 type = chapter ,
7050 gender = n ,
7051 Name-sg = Hoofdstuk ,
7052 name-sg = hoofdstuk ,
7053 Name-pl = Hoofdstukken ,
7054 name-pl = hoofdstukken ,
7055
7056 type = section ,
7057 gender = m ,
7058 Name-sg = Paragraaf ,
7059 name-sg = paragraaf ,
7060 Name-pl = Paragrafen ,
7061 name-pl = paragrafen ,
7062
7063 type = paragraph ,
7064 gender = f ,
7065 Name-sg = Alinea ,
7066 name-sg = alinea ,
7067 Name-pl = Alinea's ,
7068 name-pl = alinea's ,
7069

```

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.

```

7070 type = appendix ,
7071 gender = { f , m } ,
7072 Name-sg = Blage ,
7073 name-sg = blage ,
7074 Name-pl = Blagen ,
7075 name-pl = blagen ,
7076
7077 type = page ,
7078 gender = { f , m } ,
7079 Name-sg = Pagina ,
7080 name-sg = pagina ,
7081 Name-pl = Pagina's ,
7082 name-pl = pagina's ,
7083 rangesep = {\textendash} ,
7084 rangetopair = false ,
7085
7086 type = line ,
7087 gender = m ,
7088 Name-sg = Regel ,
7089 name-sg = regel ,
7090 Name-pl = Regels ,
7091 name-pl = regels ,
7092
7093 type = figure ,
7094 gender = { n , f , m } ,

```

```

7095 Name-sg = Figuur ,
7096 name-sg = figuur ,
7097 Name-pl = Figuren ,
7098 name-pl = figuren ,
7099
7100 type = table ,
7101   gender = { f , m } ,
7102   Name-sg = Tabel ,
7103   name-sg = tabel ,
7104   Name-pl = Tabellen ,
7105   name-pl = tabellen ,
7106
7107 type = item ,
7108   gender = n ,
7109   Name-sg = Punt ,
7110   name-sg = punt ,
7111   Name-pl = Punten ,
7112   name-pl = punten ,
7113
7114 type = footnote ,
7115   gender = { f , m } ,
7116   Name-sg = Voetnoot ,
7117   name-sg = voetnoot ,
7118   Name-pl = Voetnoten ,
7119   name-pl = voetnoten ,
7120
7121 type = endnote ,
7122   gender = { f , m } ,
7123   Name-sg = Eindnoot ,
7124   name-sg = eindnoot ,
7125   Name-pl = Eindnoten ,
7126   name-pl = eindnoten ,
7127
7128 type = note ,
7129   gender = f ,
7130   Name-sg = Opmerking ,
7131   name-sg = opmerking ,
7132   Name-pl = Opmerkingen ,
7133   name-pl = opmerkingen ,
7134
7135 type = equation ,
7136   gender = f ,
7137   Name-sg = Vergelking ,
7138   name-sg = vergelking ,
7139   Name-pl = Vergelkingen ,
7140   name-pl = vergelkingen ,
7141   Name-sg-ab = Vgl. ,
7142   name-sg-ab = vgl. ,
7143   Name-pl-ab = Vgl.'s ,
7144   name-pl-ab = vgl.'s ,
7145   refbounds-first-sg = {,(,),} ,
7146   refbounds = {(,,)} ,
7147
7148 type = theorem ,

```



```

7149 gender = f ,
7150 Name-sg = Stelling ,
7151 name-sg = stelling ,
7152 Name-pl = Stellingen ,
7153 name-pl = stellingen ,
7154

```

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”.

```

7155 type = lemma ,
7156 gender = n ,
7157 Name-sg = Lemma ,
7158 name-sg = lemma ,
7159 Name-pl = Lemma’s ,
7160 name-pl = lemma’s ,
7161
7162 type = corollary ,
7163 gender = n ,
7164 Name-sg = Gevolg ,
7165 name-sg = gevolg ,
7166 Name-pl = Gevolgen ,
7167 name-pl = gevolgen ,
7168
7169 type = proposition ,
7170 gender = f ,
7171 Name-sg = Propositie ,
7172 name-sg = propositie ,
7173 Name-pl = Propositions ,
7174 name-pl = propositions ,
7175
7176 type = definition ,
7177 gender = f ,
7178 Name-sg = Definitie ,
7179 name-sg = definitie ,
7180 Name-pl = Definities ,
7181 name-pl = definities ,
7182
7183 type = proof ,
7184 gender = n ,
7185 Name-sg = Bews ,
7186 name-sg = bews ,
7187 Name-pl = Bewzen ,
7188 name-pl = bewzen ,
7189
7190 type = result ,
7191 gender = n ,
7192 Name-sg = Resultaat ,
7193 name-sg = resultaat ,
7194 Name-pl = Resultaten ,
7195 name-pl = resultaten ,
7196
7197 type = remark ,
7198 gender = f ,

```

```

7199   Name-sg = Opmerking ,
7200   name-sg = opmerking ,
7201   Name-pl = Opmerkingen ,
7202   name-pl = opmerkingen ,
7203
7204   type = example ,
7205   gender = n ,
7206   Name-sg = Voorbeeld ,
7207   name-sg = voorbeeld ,
7208   Name-pl = Voorbeelden ,
7209   name-pl = voorbeelden ,
7210

```

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”.

```

7211   type = algorithm ,
7212   gender = { n , f , m } ,
7213   Name-sg = Algoritme ,
7214   name-sg = algoritme ,
7215   Name-pl = Algoritmen ,
7216   name-pl = algoritmen ,
7217

```

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

```

7218   type = listing ,
7219   gender = m ,
7220   Name-sg = Listing ,
7221   name-sg = listing ,
7222   Name-pl = Listings ,
7223   name-pl = listings ,
7224
7225   type = exercise ,
7226   gender = { f , m } ,
7227   Name-sg = Opgave ,
7228   name-sg = opgave ,
7229   Name-pl = Opgaven ,
7230   name-pl = opgaven ,
7231
7232   type = solution ,
7233   gender = f ,
7234   Name-sg = Oplossing ,
7235   name-sg = oplossing ,
7236   Name-pl = Oplossingen ,
7237   name-pl = oplossingen ,
7238   </lang-dutch>

```

## 10.8 Italian

Italian language file initially contributed by Matteo Ferrigato (issue [#11](#)).

```

7239 <*package>
7240 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7241 </package>

```

```

7242 <*lang-italian>
7243 namesep = {\nobreakspace} ,
7244 pairsep = {\~e\nobreakspace} ,
7245 listsep = {,~} ,
7246 lastsep = {\~e\nobreakspace} ,
7247 tpairsep = {\~e\nobreakspace} ,
7248 tlistsep = {,~} ,
7249 tlastsep = {,~e\nobreakspace} ,
7250 notesep = {\~} ,
7251 rangesep = {\~a\nobreakspace} ,
7252 +refbounds-rb = {da\nobreakspace,,,} ,
7253
7254 type = book ,
7255     gender = m ,
7256     Name-sg = Libro ,
7257     name-sg = libro ,
7258     Name-pl = Libri ,
7259     name-pl = libri ,
7260
7261 type = part ,
7262     gender = f ,
7263     Name-sg = Parte ,
7264     name-sg = parte ,
7265     Name-pl = Parti ,
7266     name-pl = parti ,
7267
7268 type = chapter ,
7269     gender = m ,
7270     Name-sg = Capitolo ,
7271     name-sg = capitolo ,
7272     Name-pl = Capitoli ,
7273     name-pl = capitoli ,
7274
7275 type = section ,
7276     gender = m ,
7277     Name-sg = Paragrafo ,
7278     name-sg = paragrafo ,
7279     Name-pl = Paragrafi ,
7280     name-pl = paragrafi ,
7281
7282 type = paragraph ,
7283     gender = m ,
7284     Name-sg = Capoverso ,
7285     name-sg = capoverso ,
7286     Name-pl = Capoversi ,
7287     name-pl = capoversi ,
7288
7289 type = appendix ,
7290     gender = f ,
7291     Name-sg = Appendice ,
7292     name-sg = appendice ,
7293     Name-pl = Appendici ,
7294     name-pl = appendici ,
7295

```

```

7296 type = page ,
7297     gender = f ,
7298     Name-sg = Pagina ,
7299     name-sg = pagina ,
7300     Name-pl = Pagine ,
7301     name-pl = pagine ,
7302     Name-sg-ab = Pag. ,
7303     name-sg-ab = pag. ,
7304     Name-pl-ab = Pag. ,
7305     name-pl-ab = pag. ,
7306     rangesep = {\textendash} ,
7307     rangetopair = false ,
7308     +refbounds-rb = {,,} ,
7309
7310 type = line ,
7311     gender = f ,
7312     Name-sg = Riga ,
7313     name-sg = riga ,
7314     Name-pl = Righe ,
7315     name-pl = righe ,
7316
7317 type = figure ,
7318     gender = f ,
7319     Name-sg = Figura ,
7320     name-sg = figura ,
7321     Name-pl = Figure ,
7322     name-pl = figure ,
7323     Name-sg-ab = Fig. ,
7324     name-sg-ab = fig. ,
7325     Name-pl-ab = Fig. ,
7326     name-pl-ab = fig. ,
7327
7328 type = table ,
7329     gender = f ,
7330     Name-sg = Tabella ,
7331     name-sg = tabella ,
7332     Name-pl = Tabelle ,
7333     name-pl = tabelle ,
7334     Name-sg-ab = Tab. ,
7335     name-sg-ab = tab. ,
7336     Name-pl-ab = Tab. ,
7337     name-pl-ab = tab. ,
7338
7339 type = item ,
7340     gender = m ,
7341     Name-sg = Punto ,
7342     name-sg = punto ,
7343     Name-pl = Punti ,
7344     name-pl = punti ,
7345
7346 type = footnote ,
7347     gender = f ,
7348     Name-sg = Nota ,
7349     name-sg = nota ,

```

```

7350 Name-pl = Note ,
7351 name-pl = note ,
7352
7353 type = endnote ,
7354 gender = f ,
7355 Name-sg = Nota ,
7356 name-sg = nota ,
7357 Name-pl = Note ,
7358 name-pl = note ,
7359
7360 type = note ,
7361 gender = f ,
7362 Name-sg = Nota ,
7363 name-sg = nota ,
7364 Name-pl = Note ,
7365 name-pl = note ,
7366
7367 type = equation ,
7368 gender = f ,
7369 Name-sg = Equazione ,
7370 name-sg = equazione ,
7371 Name-pl = Equazioni ,
7372 name-pl = equazioni ,
7373 Name-sg-ab = Eq. ,
7374 name-sg-ab = eq. ,
7375 Name-pl-ab = Eq. ,
7376 name-pl-ab = eq. ,
7377 +refbounds-rb = {da\nobreakspace(,,)} ,
7378 refbounds-first-sg = {(,,)} ,
7379 refbounds = {(,,)} ,
7380
7381 type = theorem ,
7382 gender = m ,
7383 Name-sg = Teorema ,
7384 name-sg = teorema ,
7385 Name-pl = Teoremi ,
7386 name-pl = teoremi ,
7387
7388 type = lemma ,
7389 gender = m ,
7390 Name-sg = Lemma ,
7391 name-sg = lemma ,
7392 Name-pl = Lemmi ,
7393 name-pl = lemmi ,
7394
7395 type = corollary ,
7396 gender = m ,
7397 Name-sg = Corollario ,
7398 name-sg = corollario ,
7399 Name-pl = Corollari ,
7400 name-pl = corollari ,
7401
7402 type = proposition ,
7403 gender = f ,

```

```

7404 Name-sg = Proposizione ,
7405 name-sg = proposizione ,
7406 Name-pl = Proposizioni ,
7407 name-pl = proposizioni ,
7408
7409 type = definition ,
7410 gender = f ,
7411 Name-sg = Definizione ,
7412 name-sg = definizione ,
7413 Name-pl = Definizioni ,
7414 name-pl = definizioni ,
7415
7416 type = proof ,
7417 gender = f ,
7418 Name-sg = Dimostrazione ,
7419 name-sg = dimostrazione ,
7420 Name-pl = Dimostrazioni ,
7421 name-pl = dimostrazioni ,
7422
7423 type = result ,
7424 gender = m ,
7425 Name-sg = Risultato ,
7426 name-sg = risultato ,
7427 Name-pl = Risultati ,
7428 name-pl = risultati ,
7429
7430 type = remark ,
7431 gender = f ,
7432 Name-sg = Osservazione ,
7433 name-sg = osservazione ,
7434 Name-pl = Osservazioni ,
7435 name-pl = osservazioni ,
7436
7437 type = example ,
7438 gender = m ,
7439 Name-sg = Esempio ,
7440 name-sg = esempio ,
7441 Name-pl = Esempi ,
7442 name-pl = esempi ,
7443
7444 type = algorithm ,
7445 gender = m ,
7446 Name-sg = Algoritmo ,
7447 name-sg = algoritmo ,
7448 Name-pl = Algoritmi ,
7449 name-pl = algoritmi ,
7450
7451 type = listing ,
7452 gender = m ,
7453 Name-sg = Listato ,
7454 name-sg = listato ,
7455 Name-pl = Listati ,
7456 name-pl = listati ,
7457

```

```

7458 type = exercise ,
7459 gender = m ,
7460 Name-sg = Esercizio ,
7461 name-sg = esercizio ,
7462 Name-pl = Esercizi ,
7463 name-pl = esercizi ,
7464
7465 type = solution ,
7466 gender = f ,
7467 Name-sg = Soluzione ,
7468 name-sg = soluzione ,
7469 Name-pl = Soluzioni ,
7470 name-pl = soluzioni ,
7471 </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	
\A .....	1874
\AddToHook .....	98, 2020,
	2064, 2087, 2118, 2120, 2158, 2231,
	2273, 2417, 2430, 2438, 5297, 5333,
	5350, 5352, 5357, 5402, 5404, 5406,
	5408, 5410, 5412, 5414, 5416, 5420,
	5424, 5426, 5431, 5441, 5450, 5456,
	5458, 5484, 5495, 5529, 5580, 5605
\alph .....	5510, 5596
\appendix .....	2, <i>126</i> , <i>129</i> , <i>139</i>
\appendixname .....	<i>126</i> , <i>140</i>
\AtEndOfPackage .....	2428
B	
\babelname .....	2074
\babelprovide .....	29, <i>54</i>
\bicaption .....	<i>128</i>
\bionenumcaption .....	<i>128</i>
\bitwonumcaption .....	<i>128</i>
bool commands:	
\bool_case_true: .....	<i>2</i>
\bool_gset_false:N .....	530
\bool_gset_true:N .....	523, 2404
\bool_if:NTF .....	374, 451, 489,
	547, 1750, 1801, 2024, 2028, 2440,
	3414, 3817, 3958, 4085, 4121, 4155,
	4222, 4235, 4247, 4298, 4315, 4325,
	4330, 4376, 4380, 4436, 4461, 4468,
	4474, 4485, 4491, 4519, 4564, 4586,
	4615, 4764, 4917, 4919, 5532, 5608
\bool_if:nTF .....	67,
	3528, 3538, 3562, 3579, 3594, 3659,
	3667, 4308, 4685, 4726, 4807, 4824
\bool_if_exist:NTF .....	328, 338,
	362, 372, 422, 439, 449, 474, 477,
	485, 487, 501, 504, 511, 514, 521, 528
\bool_lazy_all:nTF .....	4404, 4929
\bool_lazy_and:nnTF ..	2561, 3385,
	3406, 4189, 4260, 4638, 4902, 4944
\bool_lazy_any:nTF .....	5081, 5090
\bool_lazy_or:nnTF	1367, 1400, 1950,
	2510, 2774, 3273, 3298, 3389, 4890
\bool_new:N .....	329, 339, 364, 423,
	441, 479, 502, 505, 512, 515, 522,
	529, 789, 1564, 1565, 1592, 1616,
	1968, 1975, 1982, 1995, 1996, 2166,
	2167, 2168, 2169, 2170, 2266, 2267,
	2397, 3422, 3437, 3699, 3700, 3711,
	3712, 3719, 3721, 3722, 3735, 3736,
	3737, 3749, 3750, 5494, 5539, 5579
\bool_set:Nn .....	3382
\bool_set_eq:NN .....	537
\bool_set_false:N ..	363, 440, 476,
	478, 513, 1577, 1581, 1753, 1851,
	1896, 1938, 2003, 2012, 2013, 2030,
	2037, 2178, 2182, 2189, 2197, 2198,
	2199, 2291, 2303, 3427, 3520, 3766,
	3767, 3784, 3823, 3834, 4234, 4426,
	4427, 4434, 4435, 4668, 5088, 5105
\bool_set_true:N .....	
	..... 330, 340, 424, 503, 506,
	516, 1571, 1572, 1576, 1582, 1775,

1797, 1860, 1862, 1900, 1902, 1911, 1913, 1942, 1944, 1957, 1959, 2002, 2007, 2008, 2176, 2183, 2188, 2205, 2207, 2209, 2212, 2213, 2214, 2279, 2284, 3534, 3544, 3548, 3570, 3585, 3600, 3624, 3792, 3818, 3824, 3828, 3835, 3981, 3992, 4003, 4053, 4089, 4125, 4159, 4176, 4257, 4291, 4463, 4523, 4569, 4591, 4620, 4878, 4885, 5007, 5066, 5104, 5141, 5148, 5149, 5167, 5184, 5186, 5508, 5546, 5593	\cs_set:Npn ..... 5395 \cs_set_eq:NN ..... ..... 711, 5394, 5400, 5487, 5492 \cs_set_nopar:Npn ..... 5477 \cs_to_str:N ..... 322
\bool_until_do:Nn ..... ..... 1852, 1903, 1945, 3560, 3785 \bool_while_do:nn ..... 5640 \l_tmpa_bool ..... 1753, 1775, 1797, 1801, 1851, 1852, 1860, 1862, 1896, 1900, 1902, 1903, 1911, 1913, 1938, 1942, 1944, 1945, 1957, 1959	D \d ..... 1874
C \caption ..... 131 \chaptername ..... 140 clist commands: \clist_map_inline:nn ..... ..... 622, 670, 687, 1006, 2200, 2345, 2406, 2910, 5511, 5598 \contsubbottom ..... 129 \contsubcaption ..... 129 \counterwithin ..... 5 \crefstripprefix ..... 45 cs commands: \cs_generate_variant:Nn ..... 64, 271, 277, 284, 295, 306, 317, 332, 342, 349, 356, 367, 391, 401, 426, 433, 444, 482, 508, 518, 525, 532, 960, 1834, 1870, 1921, 1967, 2569, 4720, 4758, 5124, 5228, 5261, 5294 \cs_if_exist:NTF ..... 24, 27, 45, 48, 57, 77, 5321, 5327, 5448 \cs_if_exist_p:N 4191, 4262, 4640, 5642 \cs_if_exist_use:N ..... 5419, 5680 \cs_new:Npn ..... 55, 65, 75, 86, 272, 278, 280, 282, 285, 296, 307, 319, 321, 709, 4681, 4721, 4759, 5114 \cs_new_eq:NN ..... 5486, 5491 \cs_new_protected:Npn ..... ..... 266, 323, 333, 343, 350, 357, 382, 392, 413, 415, 417, 427, 434, 472, 499, 509, 519, 526, 802, 904, 1513, 1532, 1736, 1835, 1880, 1922, 2436, 2567, 3377, 3441, 3483, 3494, 3507, 3637, 3689, 3751, 3965, 4430, 4677, 4679, 4873, 5108, 5125, 5196, 5229, 5262, 5392, 5547	E \endinput ..... 12 \endsidecaption ..... 128 exp commands: \exp_args:NNe ..... 34, 37 \exp_args:NNNo ..... 268 \exp_args:NNNV 1815, 1864, 1915, 1961 \exp_args:NNo ..... 268, 274 \exp_args:No ..... 274 \exp_args:Nx ..... 914, 5552, 5560 \exp_args:Nxx ..... 1754, 1766, 1778, 1788, 1854, 1905, 1947, 5127, 5133, 5155, 5159, 5174 \exp_not:N ..... 112, 4332, 4345, 4348, 4351, 4697, 4701, 4709, 4713, 4738, 4741, 4749, 4752, 4780, 4783, 4787, 4792, 4798, 4801, 4813, 4816, 4844, 4849, 4859, 4864 \exp_not:n ..... 275, 3997, 4020, 4028, 4046, 4059, 4063, 4098, 4107, 4129, 4137, 4144, 4168, 4182, 4199, 4209, 4251, 4274, 4284, 4333, 4344, 4349, 4350, 4507, 4530, 4542, 4576, 4598, 4607, 4627, 4648, 4658, 4698, 4710, 4739, 4740, 4750, 4751, 4781, 4782, 4784, 4788, 4799, 4800, 4802, 4810, 4814, 4815, 4818, 4845, 4860 \ExplSyntaxOn ..... 30, 920
F \figurename ..... 140 file commands: \file_get:nnNTF ..... 914 \fmtversion ..... 3 \footnote ..... 2, 129, 130	G group commands: \group_begin: ..... 100, 725, 906, 1752, 1839, 1884, 1926, 2834, 3379, 3393, 3425, 4332, 4348, 4697, 4709, 4738, 4749, 4780, 4787, 4798, 4813, 4844, 4859, 5407, 5411, 5415 \group_end: ..... 103, 737, 958, 1816, 1865, 1916, 1962, 2862, 3396, 3419, 3429, 4345, 4351, 4701,



4713, 4741, 4752, 4783, 4792, 4801, 4816, 4849, 4864, 5409, 5413, 5417	
<b>H</b>	
\hyperlink . . . . .	5111
<b>I</b>	
\IfBooleanT . . . . .	3426
\IfClassLoadedTF . . . . .	110
\ifdraft . . . . .	2181
\IfFormatAtLeastTF . . . . .	3, 4
\ifoptionfinal . . . . .	2187
\IfPackageAtLeastTF . . . . .	2277
\IfPackageLoadedTF . . . . .	108
\input . . . . .	29, 30
int commands:	
\int_case:nnTF . . . . .	3968, 4010, 4072, 4383, 4498, 4555
\int_compare:nNnTF . . . . .	3571, 3586, 3601, 3613, 3625, 3645, 3647, 3691, 3865, 3988, 4016, 4038, 4093, 4163, 4368, 4370, 4447, 4477, 4536, 5137, 5143, 5163, 5169, 5658
\int_compare_p:nNn . . . . .	1370, 1403, 1951, 1952, 2512, 2776, 3276, 3301, 3661, 3669, 4408, 4894, 4905, 4934, 5101
\int_incr:N . . . . .	4423, 4455, 4467, 4469, 4484, 4486, 4490, 4492, 4504, 4527, 4539, 4573, 4595, 4604, 4624, 4675, 5656
\int_new:N . . . . .	3438, 3439, 3701, 3702, 3703, 3716, 3717
\int_rand:n . . . . .	293, 304, 315
\int_set:Nn . . . . .	101, 3646, 3648, 3652, 3655, 5639
\int_to_roman:n . . . . .	5643, 5650, 5651, 5654
\int_use:N . . . . .	46, 49, 53, 59
\int_zero:N . . . . .	3639, 3640, 3761, 3762, 3763, 3764, 3765, 4421, 4422, 4424, 4425, 4670, 4671
\l_tmpa_int . . . . .	5639, 5643, 5650, 5651, 5654, 5656, 5658
iow commands:	
\iow_char:N . . . . .	114, 129, 130, 135, 136, 141, 142, 147, 148, 200, 217, 264, 2248, 2257
\iow_newline: . . . . .	258
<b>J</b>	
\jobname . . . . .	5441
<b>K</b>	
keys commands:	
\l_keys_choice_tl . . . . .	794
\keys_define:nn . . . . .	20, 630, 676, 693, 754, 961, 1050, 1075, 1103, 1312, 1351, 1429, 1539, 1566, 1593, 1602, 1617, 1626, 1969, 1976, 1983, 1989, 1997, 2032, 2041, 2055, 2083, 2122, 2153, 2160, 2172, 2233, 2240, 2242, 2251, 2261, 2268, 2280, 2292, 2304, 2312, 2341, 2367, 2391, 2399, 2419, 2448, 2466, 2496, 2530, 2553, 2579, 2591, 2615, 2727, 2757, 2797, 2865, 2936, 2960, 2987, 3193, 3223, 3263, 3325
\keys_set:nn . . . . .	27, 30, 57, 58, 83, 734, 882, 945, 2285, 2568, 2573, 2859, 3380
keyval commands:	
\keyval_parse:nnn . . . . .	1518, 2316, 2371
\KOMAClassName . . . . .	5448, 5470
<b>L</b>	
\label . . . . .	128, 131, 132, 135, 5394, 5400
\labelformat . . . . .	3
\languagename . . . . .	24, 141, 2068
<b>M</b>	
\mainbabelname . . . . .	24, 2075
\MessageBreak . . . . .	10
MH commands:	
\MH_if_boolean:nTF . . . . .	5544
msg commands:	
\msg_info:nnn . . . . .	948, 1001, 1066, 1259, 1265, 1319, 5371, 5443, 5470, 5536, 5571, 5612, 5632, 5659
\msg_info:nnnn . . . . .	974, 981, 1011, 1383, 1417
\msg_info:nnnnn . . . . .	995
\msg_line_context: . . . . .	113, 119, 123, 125, 128, 134, 140, 146, 152, 157, 162, 167, 172, 178, 183, 186, 189, 194, 198, 205, 210, 215, 222, 231, 236, 241, 245, 247, 249, 251, 258
\msg_new:nnn . . . . .	111, 117, 122, 124, 126, 132, 138, 144, 150, 155, 160, 165, 170, 175, 180, 185, 187, 192, 197, 199, 201, 203, 208, 213, 219, 221, 223, 228, 234, 239, 244, 246, 248, 250, 252, 254, 256, 261
\msg_warning:nn . . . . .	2029, 2035, 2307, 2564, 4410
\msg_warning:nnn . . . . .	729, 750, 1545, 1552, 1708, 1714, 2107, 2144, 2156, 2218, 2229, 2271, 2296, 2373, 2423, 2433, 2583, 2697, 2703, 2861, 2905, 2951, 3143, 3149, 3230, 3849, 4229, 4923, 4939

\msg_warning:nnnn	818, 835, 869, 887, 2060, 2247, 2256, 2318, 2471, 2477, 2483, 2489, 2519, 2732, 2738, 2744, 2750, 2786, 2878, 2885, 2915, 3198, 3204, 3210, 3216, 3289, 3314, 3857, 5008, 5068
\msg_warning:nnnnn	855, 894, 2899, 4958
\msg_warning:nnnnnn	4965
<b>N</b>	
\newcounter	5, 5317, 5318
\NewDocumentCommand	723, 740, 2565, 2570, 2832, 3375, 3423
\newfloat	129
\NewHook	1625
\newsfloat	129
\newtheorem	140
\nobreakspace	1526, 5716, 5717, 5719, 5720, 5722, 5724, 5921, 5922, 5924, 5925, 5927, 5929, 6368, 6369, 6371, 6372, 6374, 6376, 6585, 6586, 6588, 6589, 6591, 6593, 6811, 6812, 6814, 6815, 6817, 6819, 7025, 7026, 7028, 7029, 7031, 7033, 7243, 7244, 7246, 7247, 7249, 7251, 7252, 7377
\NumCheckSetup	45
\NumsCheckSetup	45
<b>P</b>	
\PackageError	7
\pagename	140
\pagenote	129, 130
\pagenumbering	7
\pageref	84
\PagesCheckSetup	45
\paragraph	60, 61
\part	139
\partname	140
prg commands:	
\prg_generate_conditional_	
variant:Nnn	411, 459, 470, 497, 542, 550, 719, 1878
\prg_new_conditional:Npnn	107, 109, 368, 445, 483, 544, 713
\prg_new_protected_conditional:Npnn	402, 461, 533, 1871
\prg_return_false:	108, 110, 376, 380, 409, 453, 457, 468, 491, 495, 540, 547, 548, 717, 1876
\prg_return_true:	108, 110, 375, 378, 407, 452, 455, 466, 490, 493, 538, 547, 716, 1875
\prg_set_eq_conditional:NNn	721
prop commands:	
\prop_if_in:NnTF	34
\prop_if_in_p:Nn	68
\prop_item:Nn	37, 69
\prop_new:N	2311, 2366
\prop_put:Nnn	1536
\prop_remove:Nn	1535
\providecommand	3
\ProvidesExplPackage	14
\ProvidesFile	29
<b>R</b>	
\refstepcounter	3, 128, 131–133, 135, 136
regex commands:	
\regex_match:nnTF	1874
\renewlist	136, 137
\RequirePackage	16, 17, 18, 19, 2025
<b>S</b>	
\scantokens	126
seq commands:	
\seq_clear:N	438, 813, 850, 931, 944, 1005, 1613, 2507, 2771, 2845, 2858, 2909, 3443, 5255
\seq_const_from_clist:Nn	21
\seq_count:N	1371, 1385, 1404, 1419, 2512, 2521, 2776, 2788, 3277, 3291, 3302, 3316
\seq_gclear:N	1364, 1397, 3270, 3295
\seq_gconcat:NNN	615, 619
\seq_get_left:NN	828, 839, 935, 983, 2849, 2887, 3795
\seq_gput_right:Nn	946, 952, 2410
\seq_gremove_all:Nn	2442
\seq_gset_eq:NN	431, 1033
\seq_gset_from_clist:Nn	558, 567, 579, 594, 602, 763, 778
\seq_gset_split:Nnn	416
\seq_if_empty:NTF	814, 851, 932, 972, 993, 2846, 2876, 2897, 3789, 4956
\seq_if_exist:NTF	419, 429, 436, 447
\seq_if_in:NnTF	832, 866, 910, 978, 1008, 2347, 2408, 2441, 2882, 2912, 3487, 4952
\seq_item:Nn	4691, 4696, 4702, 4704, 4707, 4708, 4714, 4715, 4732, 4737, 4742, 4744, 4747, 4748, 4753, 4754, 4785, 4786, 4793, 4795, 4830, 4842, 4850, 4853, 4857, 4858, 4865, 4866
\seq_map_break:n	89, 3680, 3683
\seq_map_function:NN	3446
\seq_map_indexed_inline:Nn	44, 3641
\seq_map_inline:Nn	1047, 1072, 1309, 1348, 1426, 2432, 2445, 2493,

2527, 2576, 2588, 2754, 2794, 2933, 2957, 3220, 3260, 3322, 3677, 5550	\subsubsection . . . . . 60, 61
\seq_map_tokens:Nn . . . . . 71	\subsubsections . . . . . 139
\seq_new:N . . . . . 420, 430, 555, 556, 557, 566, 578, 593, 601, 614, 618, 758, 773, 903, 1025, 1601, 2340, 2398, 3421, 3440, 3698, 3715, 3738, 3739, 3740, 3741, 3742, 3743, 3744, 3745, 3746, 3747, 3748	\subsubsubsection . . . . . 61
\seq_pop_left:NN . . . . . 3787	\subtop . . . . . 129
\seq_put_right:Nn . . . . . . . . . . 1009, 2349, 2913, 3490	
\seq_reverse:N . . . . . 1607	
\seq_set_eq:NN . . . . . . 421, 465, 3753, 3979, 3990, 4001, 4051, 4087, 4123, 4157, 4173, 4255, 4289, 4300, 4521, 4566, 4588, 4617	
\seq_set_from_clist:Nn . . . 1606, 3381	
\seq_set_split:Nnn . . . . . 414	
\seq_sort:Nn . . . . . 86, 3449	
\seq_use:Nn . . . . . 4970	
\g_tmpa_seq . . . . . 1364, 1366, 1371, 1380, 1385, 1397, 1399, 1404, 1414, 1419, 3270, 3272, 3277, 3286, 3291, 3295, 3297, 3302, 3311, 3316	
\l_tmpa_seq 1005, 1009, 1041, 2507, 2509, 2512, 2516, 2521, 2771, 2773, 2776, 2783, 2788, 2909, 2913, 2928	
\setcounter . . 5319, 5320, 5336, 5351, 5355	
\sidefootnote . . . . . 129, 130	
sort commands:	
\sort_return_same: . . . . . 87, 91, 3456, 3461, 3535, 3555, 3576, 3591, 3605, 3630, 3665, 3680, 3696	
\sort_return_swapped: . . . . . . . 87, 91, 3469, 3545, 3554, 3575, 3590, 3606, 3629, 3673, 3683, 3695	
\stepcounter . . . . . 135, 5335, 5354	
str commands:	
\str_case:nn . . . . . 45	
\str_case:nnTF . . . . . 1108, 1630, 2089, 2126, 2202, 2619, 2992	
\str_compare:nNnTF . . . . . 3551	
\str_if_eq:nnTF . . . . . 88, 4723	
\str_if_eq_p:nn 5086, 5092, 5094, 5098	
\str_new:N . . . . . 2040	
\str_set:Nn . . . 2045, 2047, 2049, 2051	
\string . . . . . 5557, 5565	
\subbottom . . . . . 129	
\subcaption . . . . . 129	
\subcaptionref . . . . . 128	
\subparagraph . . . . . 139	
\subref . . . . . 138	
\subsections . . . . . 139	
	<b>T</b>
	\tablename . . . . . 140
	\tag . . . . . 123, 133, 135
	TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:
	\@@sidecaption . . . . . 128
	\@Alph . . . . . 126
	\@addtoreset . . . . . 5
	\@auxout . . . . . 5556, 5564
	\@bsphack . . . . . 907, 5549
	\@capttype . . . . . . . . 131, 5419, 5452, 5453, 5457, 5680
	\@chapapp . . . . . 126
	\@currentcounter . . . . . . . . . . 2, 3, 5, 62, 129, 131–133, 136, 27, 28, 48, 49, 2395, 5453, 5465
	\@currentlabel . . 3, 123, 129, 131, 136
	\@currenvir . . . . . 5460
	\@elt . . . . . 5
	\@esphack . . . . . 957, 5569
	\@ifl@t@r . . . . . 3
	\@mem@scap@afterhook . . . . . 128
	\@memsubcaption . . . . . 129
	\@onlypreamble . . . . . 739, 753, 2864
	\bbl@loaded . . . . . 54
	\bbl@main@language . . . . . 24, 2069
	\c@lstnumber . . . . . 136
	\c@page . . . . . 7, 101
	\caption@subtypehook . . . . . 5681
	\hyper@@link . . . . . 112, 121
	\hyper@linkfile . . . . . 5112
	\lst@AddToHook . . . . . 5628, 5630
	\lst@Init . . . . . 136
	\lst@label . . . . . 5629
	\lst@MakeCaption . . . . . 136
	\ltx@gobble . . . . . 132
	\ltx@label . 132, 5486, 5487, 5491, 5492
	\m@mscaplabel . . . . . 128
	\MT@newlabel . . . . . 5557, 5565
	\protected@write . . . . . 5556, 5564
	\zref@addprop 21, 31, 42, 52, 54, 96, 106
	\zref@default . . . . . 112, 4678, 4680
	\zref@extractdefault . . . . . . . . . . 10, 11, 121, 269, 275, 279
	\zref@ifpropundefined . . 42, 1263, 1550, 1712, 2701, 3147, 5116, 5595
	\zref@ifrefcontainsprop . . 42, 45, 1740, 1748, 1807, 1825, 1837, 1882, 1924, 3852, 4683, 4766, 4820, 5119

<code>\zref@ifrefundefined</code> .....	<code>\tl_if_eq:nnTF</code> .. 1754, 1766, 1778,
3451, 3453, 3465, 3820, 3822, 3827,	1788, 1854, 1905, 1947, 3643, 5127,
3844, 4226, 4237, 4438, 4761, 4875	5133, 5155, 5159, 5174, 5552, 5560
<code>\zref@label</code> ..... 131, 5480	<code>\tl_if_exist:NTF</code> ..... 325, 335,
<code>\zref@localaddprop</code> .....	345, 352, 359, 370, 386, 396, 715, 5452
..... 5421, 5533, 5609, 5682	<code>\tl_if_exist_p:N</code> ..... 2562
<code>\ZREF@mainlist</code> ..... 21, 31, 42,	<code>\tl_if_novalue:nTF</code> .....
52, 54, 96, 106, 5421, 5533, 5609, 5682	..... 2453, 2501, 2596, 2762
<code>\zref@newprop</code> .... 5, 7, 20, 22, 32,	<code>\tl_map_break:n</code> ..... 89
43, 53, 91, 105, 5418, 5510, 5596, 5679	<code>\tl_map_tokens:Nn</code> ..... 81
<code>\zref@refused</code> ..... 3842	<code>\tl_new:N</code> ..... 97, 326, 336,
<code>\zref@wrapper@babel</code> 82, 131, 3376, 5480	346, 353, 387, 397, 552, 553, 554,
<code>\textendash</code> ..... 1530,	704, 705, 706, 707, 731, 746, 1538,
5771, 6035, 6647, 6869, 7083, 7306	2152, 2171, 2239, 2260, 2390, 3431,
<code>\thechapter</code> ..... 126	3432, 3433, 3434, 3435, 3436, 3704,
<code>\thelstnumber</code> ..... 136	3705, 3706, 3707, 3708, 3709, 3710,
<code>\thepage</code> ..... 7, 102	3713, 3714, 3718, 3720, 3723, 3724,
<code>\thesection</code> ..... 126	3725, 3726, 3727, 3728, 3729, 3730,
tl commands:	3731, 3732, 3733, 3734, 5422, 5455
<code>\c_novalue_tl</code> .. 678, 679, 680, 681,	<code>\tl_put_left:Nn</code> . 4311, 4318, 4361,
682, 683, 684, 2450, 2498, 2593, 2759	4992, 4993, 5035, 5037, 5039, 5041
<code>\tl_clear:N</code> ..... 337, 361, 822,	<code>\tl_put_right:Nn</code> . 3995, 4018, 4026,
860, 873, 890, 899, 924, 933, 966,	4044, 4057, 4096, 4105, 4127, 4135,
2574, 2837, 2847, 2870, 3755, 3756,	4142, 4166, 4180, 4197, 4207, 4505,
3757, 3758, 3759, 3760, 3791, 4416,	4528, 4540, 4574, 4596, 4605, 4625,
4417, 4418, 4419, 4420, 4466, 4483,	4646, 4656, 4900, 4901, 4912, 5681
4877, 4884, 4914, 5006, 5065, 5222	<code>\tl_reverse:N</code> ..... 3511, 3514
<code>\tl_const:Nn</code> ..... 1515	<code>\tl_set:Nn</code> .....
<code>\tl_gclear:N</code> ..... 354, 398, 5468	. 268, 327, 708, 733, 923, 967, 979,
<code>\tl_gset:Nn</code> ... 102, 347, 388, 732, 747	1547, 1554, 1556, 1745, 1813, 1817,
<code>\tl_gset_eq:NN</code> ..... 5457	1830, 1841, 1846, 1857, 1858, 1866,
<code>\tl_head:N</code> .....	1868, 1886, 1891, 1908, 1909, 1917,
.. 3589, 3602, 3614, 3616, 3626, 3628	1919, 1928, 1933, 1954, 1955, 1963,
<code>\tl_head:n</code> .... 1855, 1906, 1948, 1952	1965, 2068, 2069, 2074, 2075, 2078,
<code>\tl_if_empty:NTF</code> .... 79, 816, 826,	2079, 2093, 2099, 2104, 2130, 2136,
853, 864, 885, 892, 999, 1055, 1080,	2141, 2572, 2838, 2871, 2883, 3618,
1112, 1147, 1184, 1221, 1269, 1317,	3620, 3801, 3802, 3975, 3977, 4249,
1323, 1356, 1434, 1472, 1738, 1859,	4272, 4282, 4328, 4451, 4453, 4464,
1910, 2903, 2941, 2965, 2996, 3031,	4481, 4896, 4897, 4910, 5423, 5425
3068, 3105, 3153, 3228, 3234, 3268,	<code>\tl_set_eq:NN</code> .. 406, 4414, 5453, 5464
3330, 3351, 3397, 4224, 4817, 4899,	<code>\tl_show:N</code> ..... 4377
4921, 4979, 4990, 5033, 5462, 5629	<code>\tl_tail:N</code> ..... 3619, 3621
<code>\tl_if_empty:nTF</code> .....	<code>\tl_tail:n</code> .....
.. 726, 742, 965, 1257, 1534, 1543,	.. 1857, 1858, 1908, 1909, 1954, 1955
1706, 2403, 2695, 2869, 3141, 5110	<code>\tl_use:N</code> ..... 290,
<code>\tl_if_empty_p:N</code> . 2563, 4190, 4261,	301, 312, 748, 912, 917, 947, 949, 953
4639, 4932, 4946, 5085, 5095, 5099	<code>\l_tmpa_tl</code> ..... 921, 945, 1841,
<code>\tl_if_empty_p:n</code> ..... 1368, 1401,	1855, 1857, 1886, 1897, 1906, 1908,
2511, 2775, 3274, 3299, 3530, 3531,	1928, 1939, 1948, 1954, 3402, 3403
3540, 3541, 3566, 3567, 3582, 3597	<code>\l_tmpb_tl</code> . 1803, 1810, 1813, 1817,
<code>\tl_if_eq:NNTF</code> ..... 3501, 3524, 3831	1846, 1855, 1858, 1859, 1866, 1891,
<code>\tl_if_eq:NnTF</code> .....	1899, 1906, 1909, 1910, 1917, 1933,
1764, 3444, 3476, 3651, 3654, 3679,	1941, 1948, 1951, 1952, 1955, 1963
3682, 3799, 3847, 4881, 5131, 5460	

**U**

use commands:

\use:N [25](#), [28](#), [794](#), [4193](#), [4268](#), [4642](#), [5288](#)

\UseHook ..... [1840](#), [1885](#), [1927](#)

**V**

\value ..... [5336](#), [5355](#)

\verbfootnote ..... [129](#), [130](#)

**Z**

\Z ..... [1874](#)

\zcDeclareLanguage . [12](#), [25](#), [723](#), [5706](#),  
[5911](#), [6364](#), [6579](#), [6808](#), [7022](#), [7240](#)

\zcDeclareLanguageAlias .....  
... [25](#), [740](#), [5707](#), [5708](#), [5709](#), [5710](#),  
[5711](#), [5712](#), [5713](#), [5914](#), [5915](#), [5916](#),  
[5917](#), [5918](#), [6365](#), [6580](#), [6581](#), [6582](#)

\zcLanguageSetup .....  
... [20](#), [29](#), [30](#), [67](#), [72](#), [73](#), [141](#), [2832](#)

\zcpageref ..... [84](#), [3423](#)

\zceref ..... [20](#), [64](#),  
[66](#), [82](#), [84–86](#), [92](#), [94](#), [134](#), [3375](#), [3428](#)

\zcRefTypeSetup ..... [20](#), [67](#), [2570](#)

\zcsetup ..... [20](#), [54](#), [64](#), [66](#), [2565](#)

\zlabel [128](#), [131](#), [133](#), [135](#), [136](#), [5398](#), [5629](#)

zrefcheck commands:

\zrefcheck\_zceref\_beg\_label: .. [3388](#)

\zrefcheck\_zceref\_end\_label\_-  
maybe: ..... [3410](#)

\zrefcheck\_zceref\_run\_checks\_on\_-  
labels:n ..... [3411](#)

zrefclever commands:

\zrefclever\_language\_if\_declared:n  
..... [721](#)

\zrefclever\_language\_if\_declared:nTF  
..... [721](#)

\zrefclever\_language\_varname:n ..  
..... [711](#), [711](#)

\l\_zrefclever\_ref\_language\_tl .. [707](#)

zrefclever internal commands:

\l\_zrefclever\_abbrev\_bool .....  
..... [3723](#), [3910](#), [4903](#)

\l\_zrefclever\_amsmath\_subequations\_-  
bool ..... [5494](#), [5508](#), [5532](#)

\l\_zrefclever\_breqn\_dgroup\_bool  
..... [5579](#), [5593](#), [5608](#)

\l\_zrefclever\_cap\_bool .....  
..... [3723](#), [3906](#), [4891](#)

\l\_zrefclever\_capfirst\_bool ...  
..... [1975](#), [1978](#), [4893](#)

\\_zrefclever\_compat\_module:nn ..  
..... [63](#), [2436](#),  
[2436](#), [5295](#), [5313](#), [5374](#), [5446](#), [5473](#),  
[5540](#), [5575](#), [5615](#), [5635](#), [5662](#), [5685](#)

\\_zrefclever\_counter\_reset\_by:n  
[6](#), [61](#), [62](#), [57](#), [59](#), [61](#), [65](#), [65](#), [5502](#), [5587](#)

\\_zrefclever\_counter\_reset\_by\_-  
aux:nn ..... [72](#), [75](#)

\\_zrefclever\_counter\_reset\_by\_-  
auxi:nnn ..... [82](#), [86](#)

\l\_zrefclever\_counter\_resetby\_-  
prop ..... [5](#), [61](#), [68](#), [69](#), [2366](#), [2378](#)

\l\_zrefclever\_counter\_resettters\_-  
seq . [5](#), [61](#), [62](#), [71](#), [2340](#), [2347](#), [2350](#)

\l\_zrefclever\_counter\_type\_prop  
..... [4](#), [60](#), [34](#), [37](#), [2311](#), [2323](#)

\l\_zrefclever\_current\_counter\_-  
tl ..... [3](#), [5](#), [62](#), [20](#), [24](#),  
[25](#), [35](#), [38](#), [40](#), [45](#), [46](#), [94](#), [2390](#), [2393](#)

\l\_zrefclever\_current\_language\_-  
tl ..... [24](#),  
[54](#), [705](#), [2068](#), [2074](#), [2078](#), [2094](#), [2131](#)

\l\_zrefclever\_endrangefunc\_tl ..  
... [3723](#), [3898](#), [4190](#), [4191](#), [4193](#),  
[4261](#), [4262](#), [4268](#), [4639](#), [4640](#), [4642](#)

\l\_zrefclever\_endrangeprop\_tl ..  
..... [47](#), [1738](#), [1748](#), [3723](#), [3902](#)

\\_zrefclever\_extract:nnn .....  
..... [11](#), [278](#), [278](#), [1843](#), [1848](#),  
[1888](#), [1893](#), [1930](#), [1935](#), [3572](#), [3574](#),  
[3587](#), [3604](#), [3692](#), [3694](#), [5138](#), [5140](#),  
[5144](#), [5146](#), [5164](#), [5166](#), [5170](#), [5172](#)

\\_zrefclever\_extract\_default:Nnnn  
... [11](#), [266](#), [266](#), [271](#), [1742](#), [1803](#),  
[1810](#), [1820](#), [1827](#), [3485](#), [3496](#), [3498](#),  
[3509](#), [3512](#), [3515](#), [3517](#), [3805](#), [3808](#)

\\_zrefclever\_extract\_unexp:nnn .  
..... [11](#), [121](#),  
[272](#), [272](#), [277](#), [1756](#), [1760](#), [1768](#),  
[1772](#), [1780](#), [1784](#), [1790](#), [1794](#), [4340](#),  
[4694](#), [4699](#), [4711](#), [4735](#), [4776](#), [4789](#),  
[4838](#), [4846](#), [4861](#), [5117](#), [5120](#), [5121](#),  
[5128](#), [5129](#), [5134](#), [5135](#), [5156](#), [5157](#),  
[5160](#), [5161](#), [5176](#), [5180](#), [5553](#), [5561](#)

\\_zrefclever\_extract\_url\_-  
unexp:n ..... [4336](#), [4693](#),  
[4734](#), [4772](#), [4834](#), [5114](#), [5114](#), [5124](#)

\\_zrefclever\_get\_enclosing\_-  
counters\_value:n .....  
..... [5](#), [6](#), [55](#), [55](#), [60](#), [64](#), [93](#)

\\_zrefclever\_get\_endrange\_-  
pagecomp:nnN ..... [1880](#), [1921](#)

\\_zrefclever\_get\_endrange\_-  
pagecomptwo:nnN ..... [1922](#), [1967](#)

\\_zrefclever\_get\_endrange\_-  
property:nnN ..... [45](#), [1736](#), [1834](#)

\\_zrefclever\_get\_endrange\_-  
stripprefix:nnN ..... [1835](#), [1870](#)

```

\__zrefclever_get_ref:nN .....
. 112, 113, 3998, 4021, 4029, 4047,
4060, 4064, 4099, 4108, 4130, 4138,
4145, 4169, 4183, 4210, 4252, 4285,
4320, 4508, 4531, 4543, 4577, 4599,
4608, 4628, 4659, 4681, 4681, 4720
\__zrefclever_get_ref_endrange:nnN
..... 45,
113, 4200, 4275, 4649, 4721, 4721, 4758
\__zrefclever_get_ref_first: ...
112, 113, 117, 4312, 4362, 4759, 4759
\__zrefclever_get_rf_opt_bool:nN 125
\__zrefclever_get_rf_opt_-
bool:nnnnN ..... 20,
3903, 3907, 3911, 5262, 5262, 5294
\__zrefclever_get_rf_opt_-
seq:nnnN .. 20, 124, 3915, 3919,
3923, 3927, 3931, 3935, 3939, 3943,
3947, 3951, 4948, 5229, 5229, 5261
\__zrefclever_get_rf_opt_tl:nnnN
..... 20,
22, 45, 124, 3399, 3770, 3774, 3778,
3867, 3871, 3875, 3879, 3883, 3887,
3891, 3895, 3899, 5196, 5196, 5228
\__zrefclever_hyperlink:nnn ....
..... 121, 4334,
4692, 4733, 4770, 4832, 5108, 5108
\l__zrefclever_hyperlink_bool ...
1995, 2002, 2007, 2012, 2024, 2030,
2037, 3427, 4687, 4728, 4826, 5083
\l__zrefclever_hyperref_warn_-
bool ... 1996, 2003, 2008, 2013, 2028
\__zrefclever_if_class_loaded:n .
..... 107, 109
\__zrefclever_if_class_loaded:nTF
..... 5376
\__zrefclever_if_package_-
loaded:n ..... 107, 107
\__zrefclever_if_package_-
loaded:nTF ..... 2022,
2066, 2072, 2275, 5315, 5475, 5482,
5542, 5577, 5617, 5637, 5664, 5687
\__zrefclever_is_integer_rgx:n ..
..... 1871, 1872, 1879
\__zrefclever_is_integer_rgx:nTF
..... 1897, 1899, 1939, 1941
\g__zrefclever_koma_captionofbeside_-
capytype_tl ..... 5455
\g__zrefclever_koma_capytype_tl ..
..... 5457, 5462, 5465, 5468
\l__zrefclever_label_a_tl .....
. 91, 3704, 3788, 3807, 3820, 3842,
3844, 3850, 3853, 3859, 3976, 3998,
4021, 4029, 4064, 4130, 4145, 4195,
4201, 4210, 4242, 4252, 4270, 4276,
4285, 4438, 4442, 4452, 4465, 4482,
4508, 4544, 4608, 4644, 4650, 4659
\l__zrefclever_label_b_tl .....
..... 91, 3704,
3791, 3796, 3810, 3822, 3827, 4442
\l__zrefclever_label_count_int ..
..... 92, 3701, 3761,
3865, 3968, 4421, 4447, 4675, 4935
\l__zrefclever_label_enclval_a_-
tl ..... 3431, 3509, 3511, 3566,
3582, 3602, 3614, 3618, 3619, 3626
\l__zrefclever_label_enclval_b_-
tl ..... 3431, 3512, 3514, 3567,
3589, 3597, 3616, 3620, 3621, 3628
\l__zrefclever_label_extdoc_a_tl
.. 3431, 3515, 3525, 3530, 3540, 3553
\l__zrefclever_label_extdoc_b_tl
.. 3431, 3517, 3526, 3531, 3541, 3552
\l__zrefclever_label_type_a_tl ..
..... 3400, 3431, 3486, 3488,
3491, 3497, 3502, 3651, 3679, 3771,
3775, 3779, 3801, 3806, 3832, 3847,
3868, 3872, 3876, 3880, 3884, 3888,
3892, 3896, 3900, 3904, 3908, 3912,
3916, 3920, 3924, 3928, 3932, 3936,
3940, 3944, 3948, 3952, 3978, 4454
\l__zrefclever_label_type_b_tl ..
..... 3431, 3499,
3503, 3654, 3682, 3802, 3809, 3833
\__zrefclever_label_type_put_-
new_right:n 85, 86, 3447, 3483, 3483
\l__zrefclever_label_types_seq ..
..... 86, 3440, 3443, 3487, 3490, 3677
\__zrefclever_labels_in_sequence:nn
.. 47, 92, 122, 4240, 4441, 5125, 5125
\l__zrefclever_lang_decl_case_tl
552, 933, 936, 979, 984, 1323, 1340,
2847, 2850, 2883, 2888, 3234, 3251
\l__zrefclever_lang_declension_-
seq ..... 552,
812, 813, 814, 828, 832, 839, 930,
931, 932, 935, 972, 978, 983, 2844,
2845, 2846, 2849, 2876, 2882, 2887
\l__zrefclever_lang_gender_seq ..
..... 552, 849, 850, 851, 866, 943,
944, 993, 1008, 2857, 2858, 2897, 2912
\__zrefclever_language_if_-
declared:n ..... 25, 713, 720, 722
\__zrefclever_language_if_-
declared:n(TF) ..... 24
\__zrefclever_language_if_-
declared:nTF 287, 298, 309, 713,
728, 744, 804, 908, 2105, 2142, 2835

```

```

\__zrefclever_language_varname:n
    ..... 24, 290, 301,
    312, 709, 709, 712, 715, 731, 732,
    746, 747, 748, 912, 917, 947, 949, 953
\l_zrefclever_last_of_type_bool
    ..... 92, 3698, 3818,
    3823, 3824, 3828, 3834, 3835, 3958
\l_zrefclever_lastsep_tl . 3723,
    3882, 4028, 4063, 4107, 4144, 4182
\l_zrefclever_link_star_bool ...
    .. 3382, 3421, 4688, 4729, 4827, 5084
\l_zrefclever_listsep_tl .....
    ... 3723, 3878, 4059, 4137, 4507,
    4530, 4542, 4576, 4598, 4607, 4627
\g_zrefclever_loaded_langfiles_-
    seq ..... 903, 911, 946, 952
\__zrefclever_ltxlabel:n .....
    ..... 132, 5477, 5487, 5492
\l_zrefclever_main_language_tl .
    ..... 24,
    54, 706, 2069, 2075, 2079, 2100, 2137
\__zrefclever_mathtools_showonlyrefs:n
    ..... 3416, 5547
\l_zrefclever_mathtools_-
    showonlyrefs_bool 3414, 5539, 5546
\__zrefclever_memoir_both_-
    labels: .....
    .. 5392, 5403, 5405, 5407, 5411, 5415
\l_zrefclever_memoir_footnote_-
    type_tl .... 5422, 5423, 5425, 5429
\__zrefclever_memoir_label_and_-
    zlabel:n ..... 5395, 5400
\__zrefclever_memoir_orig_-
    label:n ..... 5394, 5397
\__zrefclever_name_default: ....
    ..... 4677, 4679, 4809
\l_zrefclever_name_format_-
    fallback_tl ..... 3710, 4910,
    4914, 4979, 5028, 5040, 5042, 5060
\l_zrefclever_name_format_tl ...
    ... 3710, 4896, 4897, 4900, 4901,
    4911, 4912, 4985, 4992, 4993, 5001,
    5009, 5019, 5036, 5037, 5050, 5070
\l_zrefclever_name_in_link_bool
    ..... 114,
    117, 3710, 4330, 4764, 5088, 5104, 5105
\l_zrefclever_namefont_tl 3723,
    3890, 4333, 4349, 4781, 4799, 4814
\l_zrefclever_nameinlink_str ...
    ..... 2040, 2045, 2047,
    2049, 2051, 5086, 5092, 5094, 5098
\l_zrefclever_namesep_tl .....
    .. 3723, 3870, 4784, 4802, 4810, 4818

\l_zrefclever_next_is_same_bool
    ..... 92, 122, 3716,
    4435, 4468, 4485, 4491, 5149, 5187
\l_zrefclever_next_maybe_range_-
    bool .....
    . 92, 122, 3716, 4234, 4247, 4434,
    4461, 4474, 5141, 5148, 5167, 5185
\l_zrefclever_noabbrev_first_-
    bool ..... 1982, 1985, 4907
\g_zrefclever_nocompat_bool ...
    ..... 2397, 2404, 2440
\l_zrefclever_nocompat_bool ... 63
\g_zrefclever_nocompat_modules_-
    seq 2398, 2408, 2411, 2432, 2441, 2442
\l_zrefclever_nocompat_modules_-
    seq ..... 63
\l_zrefclever_nudge_comptosing_-
    bool ... 2168, 2198, 2207, 2213, 4931
\l_zrefclever_nudge_enabled_-
    bool ..... 2166, 2176, 2178,
    2182, 2183, 2188, 2189, 4406, 4917
\l_zrefclever_nudge_gender_bool
    ..... 2170, 2199, 2209, 2214, 4945
\l_zrefclever_nudge_multitype_-
    bool ... 2167, 2197, 2205, 2212, 4407
\l_zrefclever_nudge_singular_-
    bool ..... 2169, 2225, 4919
\__zrefclever_opt_bool_get:NN ...
    ..... 533, 543
\__zrefclever_opt_bool_get:NN(TF)
    ..... 19
\__zrefclever_opt_bool_get:NNTF .
    ... 533, 5265, 5270, 5275, 5280, 5285
\__zrefclever_opt_bool_gset_-
    false:N ..... 18,
    499, 526, 532, 1481, 1498, 3353, 3361
\__zrefclever_opt_bool_gset_-
    true:N ..... 18,
    499, 519, 525, 1443, 1460, 3332, 3340
\__zrefclever_opt_bool_if:N 544, 551
\__zrefclever_opt_bool_if:N(TF) . 19
\__zrefclever_opt_bool_if:NTF ...
    ..... 544, 877
\__zrefclever_opt_bool_if_set:N .
    ..... 483, 498
\__zrefclever_opt_bool_if_-
    set:N(TF) ..... 18
\__zrefclever_opt_bool_if_-
    set:NTF ..... 483,
    535, 546, 1436, 1452, 1474, 1490
\__zrefclever_opt_bool_set_-
    false:N 18, 499, 509, 518, 2540, 2811
\__zrefclever_opt_bool_set_-
    true:N 18, 499, 499, 508, 2535, 2802

```



```

\__zrefclever_opt_bool_unset:N . .
    . . . . . 17, 472, 472, 482, 2545, 2820
\__zrefclever_opt_seq_get:NN 461, 471
\__zrefclever_opt_seq_get:NN(TF) 17
\__zrefclever_opt_seq_get:NNTF . .
    . . . 461, 807, 844, 925, 938, 2839,
    2852, 5232, 5237, 5242, 5247, 5252
\__zrefclever_opt_seq_gset_-
    clist_split:Nn . . . . . 16,
    413, 415, 1365, 1398, 3271, 3296
\__zrefclever_opt_seq_gset_eq:NN
    . . . . . 16, 413,
    427, 433, 1374, 1407, 2920, 3280, 3305
\__zrefclever_opt_seq_if_set:N . .
    . . . . . 445, 460
\__zrefclever_opt_seq_if_-
    set:N(TF) . . . . . 17
\__zrefclever_opt_seq_if_set:NTF
    . . . . . 445, 463, 1016, 1358, 1390
\__zrefclever_opt_seq_set_clist_-
    split:Nn . . 16, 413, 413, 2508, 2772
\__zrefclever_opt_seq_set_eq:NN .
    . . . . . 16, 413, 417, 426, 2514, 2778
\__zrefclever_opt_seq_unset:N . . .
    . . . . . 16, 434, 434, 444, 2503, 2764
\__zrefclever_opt_tl_clear:N . . .
    . . . . . 14, 323,
    333, 342, 1634, 1639, 1654, 1669,
    1684, 2623, 2628, 2643, 2658, 2673
\__zrefclever_opt_tl_cset_-
    fallback:nn . . . . . 1513, 1520
\__zrefclever_opt_tl_gclear:N . . .
    . . . . . 14, 323,
    350, 356, 2998, 3004, 3012, 3019,
    3040, 3056, 3077, 3093, 3114, 3130
\__zrefclever_opt_tl_gclear_if_-
    new:N . . . . . 15, 382,
    392, 401, 1114, 1120, 1128, 1135,
    1156, 1172, 1193, 1209, 1230, 1246
\__zrefclever_opt_tl_get:NN 402, 412
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF . . .
    402, 4981, 4996, 5015, 5024, 5045,
    5055, 5199, 5204, 5209, 5214, 5219
\__zrefclever_opt_tl_gset:N . . . . 14
\__zrefclever_opt_tl_gset:Nn . . .
    . . 323, 343, 349, 2943, 2967, 2975,
    3033, 3048, 3070, 3085, 3107, 3122,
    3155, 3162, 3171, 3179, 3236, 3246
\__zrefclever_opt_tl_gset_if_-
    new:Nn . . . . . 15,
    382, 382, 391, 1057, 1082, 1091,
    1149, 1164, 1186, 1201, 1223, 1238,
    1271, 1278, 1287, 1295, 1325, 1335
\__zrefclever_opt_tl_if_set:N . . 368
\__zrefclever_opt_tl_if_set:N(TF)
    . . . . . 15
\__zrefclever_opt_tl_if_set:NTF .
    . . . . . 368, 384, 394, 404
\__zrefclever_opt_tl_set:N . . . . 14
\__zrefclever_opt_tl_set:Nn . . . .
    . . . . . 323, 323, 332,
    1648, 1663, 1678, 1718, 1724, 2459,
    2605, 2637, 2652, 2667, 2707, 2714
\__zrefclever_opt_tl_unset:N . . .
    . . . . . 14, 357, 357,
    367, 1693, 1698, 2455, 2598, 2682, 2687
\__zrefclever_opt_var_set_bool:n
    . . . . . 13, 14, 321, 321, 328,
    329, 330, 338, 339, 340, 362, 363,
    364, 372, 374, 422, 423, 424, 439,
    440, 441, 449, 451, 477, 478, 479,
    487, 489, 504, 505, 506, 514, 515, 516
\__zrefclever_opt_varname_-
    fallback:nn . . . . . 13,
    319, 319, 1516, 5220, 5253, 5286
\__zrefclever_opt_varname_-
    general:nn . . . . . 12,
    280, 280, 1636, 1641, 1650, 1656,
    1665, 1671, 1680, 1686, 1695, 1700,
    1720, 1726, 2456, 2460, 2504, 2515,
    2536, 2541, 2546, 5200, 5233, 5266
\__zrefclever_opt_varname_lang_-
    default:nnn . . 12, 296, 296, 306,
    1059, 1084, 1116, 1122, 1151, 1158,
    1188, 1195, 1225, 1232, 1273, 1280,
    1360, 1376, 1438, 1445, 1476, 1483,
    2945, 2969, 3000, 3006, 3035, 3042,
    3072, 3079, 3109, 3116, 3157, 3164,
    3282, 3334, 3355, 5215, 5248, 5281
\__zrefclever_opt_varname_lang_-
    type:nnnn . . . . . 13,
    307, 307, 318, 1018, 1027, 1035,
    1093, 1130, 1137, 1166, 1174, 1203,
    1211, 1240, 1248, 1289, 1297, 1327,
    1337, 1392, 1409, 1454, 1462, 1492,
    1500, 2922, 2977, 3014, 3021, 3050,
    3058, 3087, 3095, 3124, 3132, 3173,
    3181, 3238, 3248, 3307, 3342, 3363,
    4998, 5047, 5057, 5210, 5243, 5276
\__zrefclever_opt_varname_-
    language:nnn . . . . . 12, 285,
    285, 295, 760, 765, 775, 780, 791,
    796, 809, 846, 879, 927, 940, 2841, 2854
\__zrefclever_opt_varname_-
    type:nnn 12, 282, 282, 284, 2600,
    2607, 2625, 2630, 2639, 2645, 2654,
    2660, 2669, 2675, 2684, 2689, 2709,

```



2716, 2766, 2780, 2804, 2813, 2822,  
 4983, 5017, 5026, 5205, 5238, 5271  
 \\_zrefclever\_orig\_ltxlabel:n ...  
 ..... 5479, 5486, 5491  
 \g\_zrefclever\_page\_format\_tl ...  
 ..... 7, 97, 102, 105  
 \l\_zrefclever\_pairsep\_tl .....  
 ..... 3723, 3874, 3997,  
 4020, 4046, 4098, 4129, 4168, 4251  
 \\_zrefclever\_process\_language\_-  
 settings: .. 57, 58, 802, 802, 3384  
 \\_zrefclever\_prop\_put\_non\_-  
 empty:Nnn 42, 1532, 1532, 2322, 2377  
 \\_zrefclever\_provide\_langfile:n  
 ..... 20,  
 30, 31, 83, 904, 904, 960, 2111, 3383  
 \l\_zrefclever\_range\_beg\_is\_-  
 first\_bool ..... 3716,  
 3766, 4085, 4121, 4155, 4426,  
 4463, 4519, 4564, 4586, 4615, 4668  
 \l\_zrefclever\_range\_beg\_label\_-  
 tl ..... 92,  
 3716, 3759, 4048, 4061, 4100, 4109,  
 4139, 4170, 4184, 4194, 4419, 4464,  
 4481, 4532, 4578, 4600, 4629, 4643  
 \l\_zrefclever\_range\_count\_int ..  
 ..... 92,  
 3716, 3764, 4010, 4074, 4424, 4467,  
 4478, 4484, 4490, 4498, 4557, 4670  
 \l\_zrefclever\_range\_end\_ref\_tl .  
 ... 3716, 3760, 4196, 4202, 4271,  
 4277, 4420, 4466, 4483, 4645, 4651  
 \l\_zrefclever\_range\_same\_count\_-  
 int ..... 92,  
 3716, 3765, 3988, 4039, 4075, 4425,  
 4469, 4486, 4492, 4537, 4558, 4671  
 \l\_zrefclever\_rangeseq\_tl .....  
 ..... 3723, 3886,  
 4199, 4209, 4274, 4284, 4648, 4658  
 \l\_zrefclever\_rangetopair\_bool .  
 ..... 3723, 3914, 4235  
 \l\_zrefclever\_ref\_count\_int ...  
 ..... 3701, 3763,  
 4016, 4094, 4164, 4422, 4455, 4504,  
 4527, 4539, 4573, 4595, 4604, 4624  
 \l\_zrefclever\_ref\_decl\_case\_tl .  
 ..... 27, 816, 821, 822, 826, 829,  
 833, 837, 840, 885, 888, 890, 2152,  
 2162, 4990, 4994, 5033, 5038, 5043  
 \\_zrefclever\_ref\_default: 4677,  
 4677, 4718, 4724, 4762, 4803, 4869  
 \l\_zrefclever\_ref\_gender\_tl ...  
 ..... 28, 853, 859,  
 860, 864, 867, 872, 873, 892, 898,  
 899, 2171, 2235, 4946, 4954, 4960, 4968  
 \l\_zrefclever\_ref\_language\_tl ..  
 ..... 24, 27, 54, 704, 708, 805,  
 810, 820, 838, 847, 857, 871, 880,  
 889, 896, 2093, 2099, 2104, 2112,  
 2130, 2136, 2141, 3383, 3401, 3772,  
 3776, 3780, 3869, 3873, 3877, 3881,  
 3885, 3889, 3893, 3897, 3901, 3905,  
 3909, 3913, 3917, 3921, 3925, 3929,  
 3933, 3937, 3941, 3945, 3949, 3953,  
 4950, 4962, 4973, 4999, 5048, 5058  
 \l\_zrefclever\_ref\_property\_tl ..  
 ..... 42, 47, 1538,  
 1547, 1554, 1556, 1740, 1764, 1808,  
 1825, 1837, 1844, 1849, 1882, 1889,  
 1894, 1924, 1931, 1936, 3476, 3799,  
 3854, 3858, 4683, 4768, 4822, 5131  
 \l\_zrefclever\_ref\_propserity\_tl 3444  
 \l\_zrefclever\_ref\_typeset\_font\_-  
 tl ..... 2239, 2241, 3394  
 \l\_zrefclever\_refbounds\_first\_-  
 pb\_seq ..... 3738,  
 3926, 4002, 4052, 4124, 4175, 4256  
 \l\_zrefclever\_refbounds\_first\_-  
 rb\_seq . 3738, 3930, 4158, 4290, 4619  
 \l\_zrefclever\_refbounds\_first\_-  
 seq 3738, 3918, 4301, 4522, 4568, 4590  
 \l\_zrefclever\_refbounds\_first\_-  
 sg\_seq . 3738, 3922, 3980, 3991, 4088  
 \l\_zrefclever\_refbounds\_last\_-  
 pe\_seq ..... 3738, 3950,  
 3999, 4022, 4049, 4101, 4131, 4253  
 \l\_zrefclever\_refbounds\_last\_-  
 re\_seq .....  
 .. 3738, 3954, 4203, 4211, 4278, 4286  
 \l\_zrefclever\_refbounds\_last\_-  
 seq 3738, 3946, 4030, 4065, 4110, 4146  
 \l\_zrefclever\_refbounds\_mid\_rb\_-  
 seq ... 3738, 3938, 4171, 4185, 4630  
 \l\_zrefclever\_refbounds\_mid\_re\_-  
 seq ..... 3738, 3942, 4652, 4660  
 \l\_zrefclever\_refbounds\_mid\_seq  
 ..... 3738, 3934, 4062, 4140,  
 4509, 4533, 4545, 4579, 4601, 4609  
 \l\_zrefclever\_reffont\_tl .....  
 ..... 3723, 3894, 4698,  
 4710, 4739, 4750, 4788, 4845, 4860  
 \g\_zrefclever\_rf\_opts\_bool\_-  
 maybe\_type\_specific\_seq ....  
 ..... 52, 557, 1427, 2528, 2795, 3323  
 \g\_zrefclever\_rf\_opts\_seq\_-  
 refbounds\_seq .....  
 ..... 557, 1349, 2494, 2755, 3261

```

\g__zrefclever_rf_opts_tl_maybe-
  type_specific_seq 557, 1073, 2958
\g__zrefclever_rf_opts_tl_not-
  type_specific_seq ..... 557, 1048, 2577, 2934
\g__zrefclever_rf_opts_tl-
  reference_seq ..... 557, 2446
\g__zrefclever_rf_opts_tl_type-
  names_seq ..... 557, 1310, 3221
\g__zrefclever_rf_opts_tl-
  typesetup_seq ..... 557, 2589
\l__zrefclever_setup_language_tl
  ..... 552, 733, 761, 766, 776,
  781, 792, 797, 923, 975, 982, 996,
  1013, 1019, 1028, 1036, 1060, 1085,
  1094, 1117, 1123, 1131, 1138, 1152,
  1159, 1167, 1175, 1189, 1196, 1204,
  1212, 1226, 1233, 1241, 1249, 1274,
  1281, 1290, 1298, 1328, 1338, 1361,
  1377, 1393, 1410, 1439, 1446, 1455,
  1463, 1477, 1484, 1493, 1501, 2838,
  2879, 2886, 2900, 2917, 2923, 2946,
  2970, 2978, 3001, 3007, 3015, 3022,
  3036, 3043, 3051, 3059, 3073, 3080,
  3088, 3096, 3110, 3117, 3125, 3133,
  3158, 3165, 3174, 3182, 3239, 3249,
  3283, 3308, 3335, 3343, 3356, 3364
\l__zrefclever_setup_type_tl ...
  552, 924, 966, 967, 999, 1020, 1029,
  1037, 1055, 1080, 1095, 1112, 1132,
  1139, 1147, 1168, 1176, 1184, 1205,
  1213, 1221, 1242, 1250, 1269, 1291,
  1299, 1317, 1329, 1339, 1356, 1394,
  1411, 1434, 1456, 1464, 1472, 1494,
  1502, 2572, 2574, 2601, 2608, 2626,
  2631, 2640, 2646, 2655, 2661, 2670,
  2676, 2685, 2690, 2710, 2717, 2767,
  2781, 2805, 2814, 2823, 2837, 2870,
  2871, 2903, 2924, 2941, 2965, 2979,
  2996, 3016, 3023, 3031, 3052, 3060,
  3068, 3089, 3097, 3105, 3126, 3134,
  3153, 3175, 3183, 3228, 3240, 3250,
  3268, 3309, 3330, 3344, 3351, 3365
\l__zrefclever_sort_decided_bool
  ..... 3437, 3520, 3534, 3544,
  3548, 3560, 3570, 3585, 3600, 3624
\_zrefclever_sort_default:nn ...
  ..... 87, 3478, 3494, 3494
\_zrefclever_sort_default-
  different_types:nn .....
  ..... 44, 85, 90, 3505, 3637, 3637
\_zrefclever_sort_default_same-
  type:nn ... 85, 87, 3504, 3507, 3507
\_zrefclever_sort_labels: .....
  ..... 85-87, 91, 3392, 3441, 3441
\_zrefclever_sort_page:nn .....
  ..... 91, 3477, 3689, 3689
\l__zrefclever_sort_prior_a_int .
  ..... 3438,
  3639, 3645, 3646, 3652, 3662, 3670
\l__zrefclever_sort_prior_b_int .
  ..... 3438,
  3640, 3647, 3648, 3655, 3663, 3671
\l__zrefclever_tlastsep_tl .....
  ..... 3723, 3781, 4400
\l__zrefclever_tlistsep_tl .....
  ..... 3723, 3777, 4371
\l__zrefclever_tpairsep_tl .....
  ..... 3723, 3773, 4393
\l__zrefclever_type_count_int ...
  . 92, 117, 3701, 3762, 4368, 4370,
  4383, 4408, 4423, 4894, 4906, 5101
\l__zrefclever_type_first_label-
  tl ..... 92,
  114, 3704, 3757, 3975, 4226, 4237,
  4241, 4269, 4320, 4337, 4341, 4417,
  4451, 4761, 4767, 4773, 4777, 4790,
  4821, 4835, 4839, 4847, 4862, 4875
\l__zrefclever_type_first_label-
  type_tl ... 92, 117, 3704, 3758,
  3977, 4230, 4418, 4453, 4882, 4925,
  4941, 4949, 4961, 4967, 4984, 5000,
  5010, 5018, 5027, 5049, 5059, 5071
\l__zrefclever_type_first-
  refbounds_seq .....
  ... 3738, 3979, 3990, 4001, 4051,
  4087, 4123, 4157, 4174, 4255, 4289,
  4300, 4321, 4521, 4567, 4589, 4618,
  4785, 4786, 4793, 4795, 4831, 4843,
  4851, 4854, 4857, 4858, 4865, 4866
\l__zrefclever_type_first-
  refbounds_set_bool ..... 3738,
  3767, 3981, 3992, 4003, 4054, 4090,
  4126, 4160, 4177, 4257, 4291,
  4298, 4427, 4524, 4570, 4592, 4621
\l__zrefclever_type_name_gender-
  seq ... 3710, 4951, 4953, 4956, 4971
\l__zrefclever_type_name-
  missing_bool .....
  .. 3710, 4807, 4878, 4885, 5007, 5067
\_zrefclever_type_name_setup: ..
  ..... 20, 22, 114, 4307, 4873, 4873
\l__zrefclever_type_name_tl .....
  ..... 114, 117,
  3710, 4344, 4350, 4782, 4800, 4815,
  4817, 4877, 4884, 4988, 5004, 5006,
  5022, 5031, 5053, 5063, 5065, 5085

```

```

\l_zrefclever_typeset_compress_-
  bool ..... 1616, 1619, 4436
\l_zrefclever_typeset_labels_-
  seq 91, 3698, 3753, 3787, 3789, 3795
\l_zrefclever_typeset_last_bool
  ..... 92, 3698,
  3784, 3785, 3792, 3817, 4380, 5100
\l_zrefclever_typeset_name_bool
  .. 1565, 1572, 1577, 1582, 4309, 4325
\l_zrefclever_typeset_queue_-
  curr_tl ..... 92,
  94, 112, 117, 3704, 3756, 3995,
  4018, 4026, 4044, 4057, 4096,
  4105, 4127, 4135, 4142, 4166, 4180,
  4197, 4207, 4224, 4249, 4272, 4282,
  4311, 4318, 4328, 4361, 4377, 4388,
  4394, 4401, 4415, 4416, 4505, 4528,
  4540, 4574, 4596, 4605, 4625, 4646,
  4656, 4899, 4921, 4932, 5095, 5099
\l_zrefclever_typeset_queue_-
  prev_tl . 92, 3704, 3755, 4372, 4414
\l_zrefclever_typeset_range_-
  bool ... 1750, 1968, 1971, 3391, 4222
\l_zrefclever_typeset_ref_bool .
  .. 1564, 1571, 1576, 1581, 4309, 4315
\__zrefclever_typeset_refs: ....
  ..... 91, 93, 94, 3395, 3751, 3751
\__zrefclever_typeset_refs_last_-
  of_type: .....
  .. 98, 112, 114, 117, 3960, 3965, 3965
\__zrefclever_typeset_refs_not_-
  last_of_type: .....
  ... 92, 99, 112, 122, 3962, 4430, 4430
\l_zrefclever_typeset_sort_bool
  ..... 1592, 1595, 3390
\l_zrefclever_typesort_seq ....
  . 44, 90, 1601, 1606, 1607, 1613, 3641
\l_zrefclever_verbose_testing_-
  bool ..... 3750, 4376
\__zrefclever_zcref:nnn .....
  ..... 27, 56, 3376, 3377
\__zrefclever_zcref:nnnn 82, 85, 3377
\l_zrefclever_zcref_labels_seq .
  ..... 85, 86, 3381,
  3412, 3417, 3421, 3446, 3449, 3754
\l_zrefclever_zcref_note_tl ...
  ..... 2260, 2263, 3397, 3404
\l_zrefclever_zcref_with_check_-
  bool ..... 2267, 2284, 3387, 3408
\__zrefclever_zcsetup:n .....
  . 67, 2566, 2567, 2567, 2569, 5299,
  5323, 5329, 5337, 5359, 5378, 5428,
  5432, 5433, 5442, 5497, 5531, 5582,
  5607, 5619, 5631, 5646, 5666, 5689
\l_zrefclever_zrefcheck_-
  available_bool .....
  .. 2266, 2279, 2291, 2303, 3386, 3407

```