

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

## Code documentation

Gustavo Barros<sup>†</sup>

2022-02-11

### 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><code>zref</code> 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 <code>infra</code> . . . . .     | 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      | <code>\zcsetup</code> . . . . .         | 66        |
| 5.2      | <code>\zcRefTypeSetup</code> . . . . .  | 67        |
| 5.3      | <code>\zcLanguageSetup</code> . . . . . | 72        |
| <b>6</b> | <b>User interface</b>                   | <b>82</b> |
| 6.1      | <code>\zceref</code> . . . . .          | 82        |
| 6.2      | <code>\zcpageref</code> . . . . .       | 84        |
| <b>7</b> | <b>Sorting</b>                          | <b>84</b> |
| <b>8</b> | <b>Typesetting</b>                      | <b>91</b> |

---

<sup>\*</sup>This file describes v0.2.2-alpha, released 2022-02-11.

<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      | English               | 138        |
| 10.2      | German                | 142        |
| 10.3      | French                | 151        |
| 10.4      | Portuguese            | 155        |
| 10.5      | Spanish               | 159        |
| 10.6      | Dutch                 | 163        |
|           | <b>Index</b>          | <b>168</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} {2022-02-11} {0.2.2-alpha}
15  {Clever LaTeX cross-references based on zref}

```

## 2 Dependencies

Required packages. Besides these, `zref-hyperref`, `zref-titleref`, and `zref-check` may also be loaded depending on user options.

```

16  \RequirePackage { zref-base }
17  \RequirePackage { zref-user }
18  \RequirePackage { zref-abspage }
19  \RequirePackage { l3keys2e }
20  \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`.

```

21  \zref@newprop { zc@counter } { \l__zrefclever_current_counter_tl }
22  \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.

```

23  \zref@newprop { thecounter }
24  {
25    \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
26    { \use:c { the \l__zrefclever_current_counter_tl } }
27    {
28      \cs_if_exist:cT { c@ \@currentcounter }

```

```

29         { \use:c { the \@currentcounter } }
30     }
31 }
32 \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`.

```

33 \zref@newprop { zc@type }
34 {
35     \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
36         \l__zrefclever_current_counter_tl
37     {
38         \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
39             { \l__zrefclever_current_counter_tl }
40     }
41     { \l__zrefclever_current_counter_tl }
42 }
43 \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`’).

```

44 \zref@newprop { zc@cntval } [0]
45 {
46     \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
47     { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
48     {
49         \cs_if_exist:cT { c@ \@currentcounter }
50         { \int_use:c { c@ \@currentcounter } }
51     }
52 }
53 \zref@addprop \ZREF@mainlist { zc@cntval }
54 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
55 \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>}

56 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
57 {
58   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }

```

```

59   {
60     { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
61     \_zrefclever_get_enclosing_counters_value:e
62     { \_zrefclever_counter_reset_by:n {#1} }
63   }
64 }

```

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)).

```

65 \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  $\langle counter \rangle$ .

```

\_zrefclever_counter_reset_by:n { \langle counter \rangle }

66 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
67   {
68     \bool_if:nTF
69       { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
70       { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
71       {
72         \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
73           { \_zrefclever_counter_reset_by_aux:nn {#1} }
74       }
75   }
76 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
77   {
78     \cs_if_exist:cT { c@ #2 }
79     {
80       \tl_if_empty:cF { c1@ #2 }
81       {
82         \tl_map_tokens:cn { c1@ #2 }
83         { \_zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
84       }
85     }
86   }
87 \cs_new:Npn \_zrefclever_counter_reset_by_auxi:nnn #1#2#3
88   {
89     \str_if_eq:nnT {#2} {#3}
90     { \tl_map_break:n { \seq_map_break:n {#1} } }
91   }

```

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

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

```

92 \zref@newprop { zc@enclval }
93 {

```

```

94     \__zrefclever_get_enclosing_counters_value:e
95     \l__zrefclever_current_counter_tl
96   }
97   \zref@addprop \ZREF@mainlist { zc@enclval }

```

Another piece of information we need is the page numbering format being used by `\thepage`, so that we know when we can (or not) group a set of page references in a range. Unfortunately, `page` is not a typical counter in ways which complicates things. First, it does commonly get reset along the document, not necessarily by the usual counter reset chains, but rather with `\pagenumbering` or variations thereof. Second, the format of the page number commonly changes in the document (roman, arabic, etc.), not necessarily, though usually, together with a reset. Trying to “parse” `\thepage` to retrieve such information is bound to go wrong: we don’t know, and can’t know, what is within that macro, and that’s the business of the user, or of the documentclass, or of the loaded packages. The technique used by `cleveref`, which we borrow here, is simple and smart: store with the label what `\thepage` would return, if the counter `\c@page` was “1”. That does not allow us to *sort* the references, luckily however, we have `abspage` which solves this problem. But we can decide whether two labels can be compressed into a range or not based on this format: if they are identical, we can compress them, otherwise, we can’t. To do so, we locally redefine `\c@page` to return “1”, thus avoiding any global spillovers of this trick. Since this operation is not expandable we cannot run it directly from the property definition. Hence, we use a shipout hook, and set `\g__zrefclever_page_format_tl`, which can then be retrieved by the starred definition of `\zref@newprop*{zc@pgfmt}`.

```

98 \tl_new:N \g__zrefclever_page_format_tl
99 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
100 \AddToHook { shipout / before }
101 {
102   \group_begin:
103   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
104   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
105   \group_end:
106 }
107 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
108 \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.

```

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

```

113 \msg_new:nnn { zref-clever } { option-not-type-specific }
114 {
115   Option~'#1'~is-not-type-specific~\msg_line_context:~
116   Set-it-in~'\iow_char:N\zcLanguageSetup'~before-first~'type'~
117   switch-or-as~package~option.
118 }
119 \msg_new:nnn { zref-clever } { option-only-type-specific }
120 {
121   No~type~specified~for~option~'#1'~\msg_line_context:~
122   Set-it~after~'type'~switch.
123 }
124 \msg_new:nnn { zref-clever } { key-requires-value }
125 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:~ }
126 \msg_new:nnn { zref-clever } { language-declared }
127 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing~to~do. }
128 \msg_new:nnn { zref-clever } { unknown-language-alias }
129 {
130   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
131   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
132   '\iow_char:N\zcDeclareLanguageAlias'.
133 }
134 \msg_new:nnn { zref-clever } { unknown-language-setup }
135 {
136   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
137   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
138   '\iow_char:N\zcDeclareLanguageAlias'.
139 }
140 \msg_new:nnn { zref-clever } { unknown-language-opt }
141 {
142   Language~'#1'~is~unknown~\msg_line_context:~
143   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
144   '\iow_char:N\zcDeclareLanguageAlias'.
145 }
146 \msg_new:nnn { zref-clever } { unknown-language-decl }
147 {
148   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
149   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
150   '\iow_char:N\zcDeclareLanguageAlias'.
151 }
152 \msg_new:nnn { zref-clever } { language-no-decl-ref }
153 {
154   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
155   Nothing~to~do~with~option~'d=#2'.
156 }
157 \msg_new:nnn { zref-clever } { language-no-gender }
158 {
159   Language~'#1'~has~no~declared~gender~\msg_line_context:~
160   Nothing~to~do~with~option~'#2=#3'.
161 }
162 \msg_new:nnn { zref-clever } { language-no-decl-setup }

```



```

163 {
164   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
165   Nothing~to~do~with~option~'case=#2'.
166 }
167 \msg_new:nnn { zref-clever } { unknown-decl-case }
168 {
169   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:~
170   Using~default~declension~case.
171 }
172 \msg_new:nnn { zref-clever } { nudge-multitype }
173 {
174   Reference~with~multiple~types~\msg_line_context:~
175   You~may~wish~to~separate~them~or~review~language~around~it.
176 }
177 \msg_new:nnn { zref-clever } { nudge-comptosing }
178 {
179   Multiple~labels~have~been~compressed~into~singular~type~name~
180   for~type~'#1'~\msg_line_context:.
181 }
182 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
183 {
184   Option~'sg'~signals~that~a~singular~type~name~was~expected~
185   \msg_line_context:~But~type~'#1'~has~plural~type~name.
186 }
187 \msg_new:nnn { zref-clever } { gender-not-declared }
188 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
189 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
190 {
191   Gender~mismatch~for~type~'#1'~\msg_line_context:~
192   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
193 }
194 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
195 {
196   You've~specified~'g=#1'~\msg_line_context:~
197   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
198 }
199 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
200 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
201 \msg_new:nnn { zref-clever } { option-document-only }
202 { Option~'#1'~is~only~available~after~\iow_char:N\\begin\{document\}. }
203 \msg_new:nnn { zref-clever } { langfile-loaded }
204 { Loaded~'#1'~language~file. }
205 \msg_new:nnn { zref-clever } { zref-property-undefined }
206 {
207   Option~'ref=#1'~requested~\msg_line_context:~
208   But~the~property~'#1'~is~not~declared,~falling~back~to~'default'.
209 }
210 \msg_new:nnn { zref-clever } { endrange-property-undefined }
211 {
212   Option~'endrange=#1'~requested~\msg_line_context:~
213   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
214 }
215 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
216 {

```

```

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

```

### 4.3 Data extraction

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

```

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

268 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
269   {
270     \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
271       { \zref@extractdefault {#2} {#3} {#4} }
272   }
273 \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}

274 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
275   {
276     \exp_args:NNo \exp_args:No
277       \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
278   }
279 \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}

280 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
281   { \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}

282 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
283 { 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}

284 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
285 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
286 \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}

287 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
288 {
289   \_zrefclever_language_if_declared:nTF {#1}
290   {
291     g\_zrefclever_opt_language_
292     \tl_use:c { \_zrefclever_language_varname:n {#1} }
293     _ #2 _ #3
294   }
295   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
296 }
297 \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>}}
298 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
299 {
300   \_zrefclever_language_if_declared:nTF {#1}
301   {
302     g__zrefclever_opt_lang_
303     \tl_use:c { \_zrefclever_language_varname:n {#1} }
304     _default_ #2 _ #3
305   }
306   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
307 }
308 \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>}}
309 \cs_new:Npn \_zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
310 {
311   \_zrefclever_language_if_declared:nTF {#1}
312   {
313     g__zrefclever_opt_lang_
314     \tl_use:c { \_zrefclever_language_varname:n {#1} }
315     _type_ #2 _ #3 _ #4
316   }
317   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
318 }
319 \cs_generate_variant:Nn
320 \_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>}}
321 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
322 { 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}}

```

```

323 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
324 { \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

```

```

325 \cs_new_protected:Npn \_zrefclever_opt_tl_set:Nn #1#2
326 {
327   \tl_if_exist:NF #1
328   { \tl_new:N #1 }
329   \tl_set:Nn #1 {#2}
330   \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
331   { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
332   \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} }
333 }
334 \cs_generate_variant:Nn \_zrefclever_opt_tl_set:Nn { cn }
335 \cs_new_protected:Npn \_zrefclever_opt_tl_clear:N #1
336 {
337   \tl_if_exist:NF #1
338   { \tl_new:N #1 }
339   \tl_clear:N #1
340   \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
341   { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
342   \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} }
343 }
344 \cs_generate_variant:Nn \_zrefclever_opt_tl_clear:N { c }
345 \cs_new_protected:Npn \_zrefclever_opt_tl_gset:Nn #1#2
346 {
347   \tl_if_exist:NF #1
348   { \tl_new:N #1 }
349   \tl_gset:Nn #1 {#2}
350 }
351 \cs_generate_variant:Nn \_zrefclever_opt_tl_gset:Nn { cn }
352 \cs_new_protected:Npn \_zrefclever_opt_tl_gclear:N #1
353 {
354   \tl_if_exist:NF #1
355   { \tl_new:N #1 }
356   \tl_gclear:N #1
357 }
358 \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}

```

```

359 \cs_new_protected:Npn \_zrefclever_opt_tl_unset:N #1
360 {
361   \tl_if_exist:NT #1

```

```

362     {
363       \tl_clear:N #1 % ?
364       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
365       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
366       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
367     }
368   }
369   \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>}
370 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
371 {
372   \tl_if_exist:NTF #1
373   {
374     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
375     {
376       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
377       { \prg_return_true: }
378       { \prg_return_false: }
379     }
380     { \prg_return_true: }
381   }
382   { \prg_return_false: }
383 }

```

(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>}
384 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
385 {
386   \__zrefclever_opt_tl_if_set:NF #1
387   {
388     \tl_if_exist:NF #1
389     { \tl_new:N #1 }
390     \tl_gset:Nn #1 {#2}
391   }
392 }
393 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
394 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
395 {
396   \__zrefclever_opt_tl_if_set:NF #1
397   {
398     \tl_if_exist:NF #1
399     { \tl_new:N #1 }
400     \tl_gclear:N #1
401   }

```

```

402 }
403 \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>}}
404 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
405 {
406     \__zrefclever_opt_tl_if_set:NTF #1
407     {
408         \tl_set_eq:NN #2 #1
409         \prg_return_true:
410     }
411     { \prg_return_false: }
412 }
413 \prg_generate_conditional_variant:Nnn
414 \__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>}
415 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
416 { \seq_set_split:Nnn #1 { , } {#2} }
417 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
418 { \seq_gset_split:Nnn #1 { , } {#2} }
419 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
420 {
421     \seq_if_exist:NF #1
422     { \seq_new:N #1 }
423     \seq_set_eq:NN #1 #2
424     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
425     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
426     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
427 }
428 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
429 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
430 {
431     \seq_if_exist:NF #1
432     { \seq_new:N #1 }
433     \seq_gset_eq:NN #1 #2
434 }
435 \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>}

```



```

436 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
437 {
438   \seq_if_exist:NT #1
439   {
440     \seq_clear:N #1 % ?
441     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
442     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
443     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
444   }
445 }
446 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

(End definition for \\_\_zrefclever\_opt\_seq\_unset:N.)

\\_\_zrefclever\_opt\_seq\_if\_set:N~~TF~~ 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}
447 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
448 {
449   \seq_if_exist:NTF #1
450   {
451     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
452     {
453       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
454       { \prg_return_true: }
455       { \prg_return_false: }
456     }
457     { \prg_return_true: }
458   }
459   { \prg_return_false: }
460 }
461 \prg_generate_conditional_variant:Nnn
462 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End definition for \\_\_zrefclever\_opt\_seq\_if\_set:N~~TF~~.)

```

\__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}
463 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
464 {
465   \__zrefclever_opt_seq_if_set:NTF #1
466   {
467     \seq_set_eq:NN #2 #1
468     \prg_return_true:
469   }
470   { \prg_return_false: }
471 }
472 \prg_generate_conditional_variant:Nnn
473 \__zrefclever_opt_seq_get:NN { cN } { F }

```

(End definition for \\_\_zrefclever\_opt\_seq\_get:NN~~TF~~.)

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

```

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

```

```

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

```

```

515     \bool_set_false:N #1
516     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
517       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
518     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
519   }
520 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
521 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
522 {
523   \bool_if_exist:NF #1
524     { \bool_new:N #1 }
525   \bool_gset_true:N #1
526 }
527 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
528 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
529 {
530   \bool_if_exist:NF #1
531     { \bool_new:N #1 }
532   \bool_gset_false:N #1
533 }
534 \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>}}
535 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
536 {
537   \__zrefclever_opt_bool_if_set:NTF #1
538   {
539     \bool_set_eq:NN #2 #1
540     \prg_return_true:
541   }
542   { \prg_return_false: }
543 }
544 \prg_generate_conditional_variant:Nnn
545   \__zrefclever_opt_bool_get:NN { cN } { F }

```

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

```

\__zrefclever_opt_bool_if:NNTF      \__zrefclever_opt_bool_if:N(TF) {<option bool>} {<true>} {<false>}}
546 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
547 {
548   \__zrefclever_opt_bool_if_set:NTF #1
549   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
550   { \prg_return_false: }
551 }
552 \prg_generate_conditional_variant:Nnn
553   \__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 `\zcref`, 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>     |  |

```

554 \tl_new:N \l__zrefclever_setup_type_tl
555 \tl_new:N \l__zrefclever_setup_language_tl
556 \tl_new:N \l__zrefclever_lang_decl_case_tl
557 \seq_new:N \l__zrefclever_lang_declension_seq
558 \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.

```

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

```

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:.

```

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

```

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

```

616 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
617 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
618 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
619 \g__zrefclever_rf_opts_tl_type_names_seq
620 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
621 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
622 \g__zrefclever_rf_opts_tl_not_type_specific_seq
623 \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.

```

624 \clist_map_inline:nn
625 {
626   reference ,
627   typesetup ,
628   langsetup ,
629   langfile ,
630 }
631 {
632   \keys_define:nn { zref-clever/ #1 }
633   {
634     +refbounds-first .meta:n =
635     {
636       refbounds-first = {##1} ,
637       refbounds-first-sg = {##1} ,
638       refbounds-first-pb = {##1} ,
639       refbounds-first-rb = {##1} ,
640     } ,
641     +refbounds-mid .meta:n =
642     {
643       refbounds-mid = {##1} ,
644       refbounds-mid-rb = {##1} ,
645       refbounds-mid-re = {##1} ,
646     } ,
647     +refbounds-last .meta:n =
648     {
649       refbounds-last = {##1} ,

```

```

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

```

```

704     }
705 }

```

## 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.

```

706 \tl_new:N \l__zrefclever_ref_language_tl
707 \tl_new:N \l__zrefclever_current_language_tl
708 \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`.

```

709 \tl_new:N \l_zrefclever_ref_language_tl
710 \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>}

711 \cs_new:Npn \_zrefclever_language_varname:n #1
712 { 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`.

```

713 \cs_set_eq:NN \zrefclever_language_varname:n
714 \_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>}

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

```

723 \prg_set_eq_conditional:Nn \zrefclever_language_if_declared:n
724 \__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>}

725 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
726 {
727   \group_begin:
728   \tl_if_empty:nF {#2}
729   {
730     \__zrefclever_language_if_declared:nTF {#2}
731     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
732     {
733       \tl_new:c { \__zrefclever_language_varname:n {#2} }
734       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
735       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
736       \keys_set:nn { zref-clever/declarelang } {#1}
737     }
738   }
739   \group_end:
740 }
741 \@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>}

742 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
743 {
744   \tl_if_empty:nF {#1}
745   {

```

```

746     \_zrefclever_language_if_declared:nTF {#2}
747     {
748         \tl_new:c { \_zrefclever_language_varname:n {#1} }
749         \tl_gset:cx { \_zrefclever_language_varname:n {#1} }
750         { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
751     }
752     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
753 }
754 }
755 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

756 \keys_define:nn { zref-clever/declarelang }
757 {
758     declension .code:n =
759     {
760         \seq_new:c
761         {
762             \_zrefclever_opt_varname_language:enn
763             { \l__zrefclever_setup_language_tl } { declension } { seq }
764         }
765         \seq_gset_from_clist:cn
766         {
767             \_zrefclever_opt_varname_language:enn
768             { \l__zrefclever_setup_language_tl } { declension } { seq }
769         }
770         {#1}
771     } ,
772     declension .value_required:n = true ,
773     gender .code:n =
774     {
775         \seq_new:c
776         {
777             \_zrefclever_opt_varname_language:enn
778             { \l__zrefclever_setup_language_tl } { gender } { seq }
779         }
780         \seq_gset_from_clist:cn
781         {
782             \_zrefclever_opt_varname_language:enn
783             { \l__zrefclever_setup_language_tl } { gender } { seq }
784         }
785         {#1}
786     } ,
787     gender .value_required:n = true ,
788     allcaps .choices:nn =
789     { true , false }
790     {
791         \bool_new:c
792         {
793             \_zrefclever_opt_varname_language:enn
794             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
795         }
796         \use:c { bool_gset_ \l_keys_choice_tl :c }
797         {

```

```

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

```

`\_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.

```

804 \cs_new_protected:Npn \_zrefclever_process_language_settings:
805 {
806     \_zrefclever_language_if_declared:xTF
807     { \l__zrefclever_ref_language_tl }
808     {

```

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.

```

809     \_zrefclever_opt_seq_get:cNF
810     {
811         \_zrefclever_opt_varname_language:enn
812         { \l__zrefclever_ref_language_tl } { declension } { seq }
813     }
814     \l__zrefclever_lang_declension_seq
815     { \seq_clear:N \l__zrefclever_lang_declension_seq }
816     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
817     {
818         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
819         {
820             \msg_warning:nxxx { zref-clever }
821             { language-no-decl-ref }
822             { \l__zrefclever_ref_language_tl }
823             { \l__zrefclever_ref_decl_case_tl }
824             \tl_clear:N \l__zrefclever_ref_decl_case_tl
825         }
826     }
827     {
828         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
829         {
830             \seq_get_left:NN \l__zrefclever_lang_declension_seq
831             \l__zrefclever_ref_decl_case_tl
832         }
833         {
834             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

835         \l__zrefclever_ref_decl_case_tl
836     {
837         \msg_warning:nxxx { zref-clever }
838         { unknown-decl-case }
839         { \l__zrefclever_ref_decl_case_tl }
840         { \l__zrefclever_ref_language_tl }
841         \seq_get_left:NN \l__zrefclever_lang_declension_seq
842         \l__zrefclever_ref_decl_case_tl
843     }
844 }
845 }

```

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.

```

846     \__zrefclever_opt_seq_get:cNF
847     {
848         \__zrefclever_opt_varname_language:enn
849         { \l__zrefclever_ref_language_tl } { gender } { seq }
850     }
851     \l__zrefclever_lang_gender_seq
852     { \seq_clear:N \l__zrefclever_lang_gender_seq }
853     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
854     {
855         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
856         {
857             \msg_warning:nxxxx { zref-clever }
858             { language-no-gender }
859             { \l__zrefclever_ref_language_tl }
860             { g }
861             { \l__zrefclever_ref_gender_tl }
862             \tl_clear:N \l__zrefclever_ref_gender_tl
863         }
864     }
865     {
866         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
867         {
868             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
869             \l__zrefclever_ref_gender_tl
870             {
871                 \msg_warning:nxxx { zref-clever }
872                 { gender-not-declared }
873                 { \l__zrefclever_ref_language_tl }
874                 { \l__zrefclever_ref_gender_tl }
875                 \tl_clear:N \l__zrefclever_ref_gender_tl
876             }
877         }
878     }

```

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

```

879     \__zrefclever_opt_bool_if:cT
880     {
881         \__zrefclever_opt_varname_language:enn
882         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

883     }
884     { \keys_set:nn { zref-clever/reference } { cap = true } }
885   }
886   {

```

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

```

887     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
888     {
889         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
890         { \l__zrefclever_ref_decl_case_tl }
891         { \l__zrefclever_ref_language_tl }
892         \tl_clear:N \l__zrefclever_ref_decl_case_tl
893     }
894     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
895     {
896         \msg_warning:nxxxx { zref-clever }
897         { language-no-gender }
898         { \l__zrefclever_ref_language_tl }
899         { g }
900         { \l__zrefclever_ref_gender_tl }
901         \tl_clear:N \l__zrefclever_ref_gender_tl
902     }
903   }
904 }

```

(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.

```
905 \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 {\langle language \rangle}

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

```

```

935         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
936       {
937         \seq_get_left:NN \l__zrefclever_lang_declension_seq
938         \l__zrefclever_lang_decl_case_tl
939       }
940     \__zrefclever_opt_seq_get:cNF
941     {
942       \__zrefclever_opt_varname_language:nnn
943       {#1} { gender } { seq }
944     }
945     \l__zrefclever_lang_gender_seq
946     { \seq_clear:N \l__zrefclever_lang_gender_seq }
947   \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
948   \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
949   { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
950   \msg_info:nnx { zref-clever } { langfile-loaded }
951   { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
952 }
953 {

```

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.

```

954       \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
955       { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
956     }
957   }
958 }
959 \@esphack
960 \group_end:
961 }
962 \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.

```

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

```

```

978     }
979     {
980         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
981         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
982         {
983             \msg_info:nxxx { zref-clever } { unknown-decl-case }
984             {#1} { \l__zrefclever_setup_language_tl }
985             \seq_get_left:NN \l__zrefclever_lang_declension_seq
986             \l__zrefclever_lang_decl_case_tl
987         }
988     }
989 },
990 case .value_required:n = true ,
991
992 gender .value_required:n = true ,
993 gender .code:n =
994 {
995     \seq_if_empty:NnTF \l__zrefclever_lang_gender_seq
996     {
997         \msg_info:nxxxx { zref-clever } { language-no-gender }
998         { \l__zrefclever_setup_language_tl } { gender } {#1}
999     }
1000     {
1001         \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1002         {
1003             \msg_info:nnn { zref-clever }
1004             { option-only-type-specific } { gender }
1005         }
1006         {
1007             \seq_clear:N \l_tmpa_seq
1008             \clist_map_inline:nn {#1}
1009             {
1010                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1011                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
1012                 {
1013                     \msg_info:nxxx { zref-clever }
1014                     { gender-not-declared }
1015                     { \l__zrefclever_setup_language_tl } {##1}
1016                 }
1017             }
1018             \__zrefclever_opt_seq_if_set:cF
1019             {
1020                 \__zrefclever_opt_varname_lang_type:eenn
1021                 { \l__zrefclever_setup_language_tl }
1022                 { \l__zrefclever_setup_type_tl }
1023                 { gender }
1024                 { seq }
1025             }
1026             {
1027                 \seq_new:c
1028                 {
1029                     \__zrefclever_opt_varname_lang_type:eenn
1030                     { \l__zrefclever_setup_language_tl }
1031                     { \l__zrefclever_setup_type_tl }

```



```

1032         { gender }
1033         { seq }
1034     }
1035     \seq_gset_eq:cN
1036     {
1037         \__zrefclever_opt_varname_lang_type:eenn
1038         { \l__zrefclever_setup_language_tl }
1039         { \l__zrefclever_setup_type_tl }
1040         { gender }
1041         { seq }
1042     }
1043     \l_tmpa_seq
1044 }
1045 }
1046 }
1047 },
1048 }
1049 \seq_map_inline:Nn
1050 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1051 {
1052     \keys_define:nn { zref-clever/langfile }
1053     {
1054         #1 .value_required:n = true ,
1055         #1 .code:n =
1056         {
1057             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1058             {
1059                 \__zrefclever_opt_tl_gset_if_new:cn
1060                 {
1061                     \__zrefclever_opt_varname_lang_default:enn
1062                     { \l__zrefclever_setup_language_tl }
1063                     {#1} { tl }
1064                 }
1065                 {##1}
1066             }
1067             {
1068                 \msg_info:nnn { zref-clever }
1069                 { option-not-type-specific } {#1}
1070             }
1071         },
1072     }
1073 }
1074 \seq_map_inline:Nn
1075 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1076 {
1077     \keys_define:nn { zref-clever/langfile }
1078     {
1079         #1 .value_required:n = true ,
1080         #1 .code:n =
1081         {
1082             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1083             {
1084                 \__zrefclever_opt_tl_gset_if_new:cn
1085                 {

```

```

1086         \_zrefclever_opt_varname_lang_default:enn
1087         { \l__zrefclever_setup_language_tl }
1088         {#1} { tl }
1089     }
1090     {##1}
1091 }
1092 {
1093     \_zrefclever_opt_tl_gset_if_new:cn
1094     {
1095         \_zrefclever_opt_varname_lang_type:enn
1096         { \l__zrefclever_setup_language_tl }
1097         { \l__zrefclever_setup_type_tl }
1098         {#1} { tl }
1099     }
1100     {##1}
1101 }
1102 },
1103 }
1104 }
1105 \keys_define:nn { zref-clever/langfile }
1106 {
1107     endrange .value_required:n = true ,
1108     endrange .code:n =
1109     {
1110         \str_case:nnF {#1}
1111         {
1112             { ref }
1113             {
1114                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1115                 {
1116                     \_zrefclever_opt_tl_gclear_if_new:c
1117                     {
1118                         \_zrefclever_opt_varname_lang_default:enn
1119                         { \l__zrefclever_setup_language_tl }
1120                         { endrangefunc } { tl }
1121                     }
1122                     \_zrefclever_opt_tl_gclear_if_new:c
1123                     {
1124                         \_zrefclever_opt_varname_lang_default:enn
1125                         { \l__zrefclever_setup_language_tl }
1126                         { endrangeprop } { tl }
1127                     }
1128                 }
1129             }
1130             \_zrefclever_opt_tl_gclear_if_new:c
1131             {
1132                 \_zrefclever_opt_varname_lang_type:enn
1133                 { \l__zrefclever_setup_language_tl }
1134                 { \l__zrefclever_setup_type_tl }
1135                 { endrangefunc } { tl }
1136             }
1137             \_zrefclever_opt_tl_gclear_if_new:c
1138             {
1139                 \_zrefclever_opt_varname_lang_type:enn

```

```

1140         { \l__zrefclever_setup_language_tl }
1141         { \l__zrefclever_setup_type_tl }
1142         { endrangeprop } { tl }
1143     }
1144 }
1145 }
1146
1147 { stripprefix }
1148 {
1149     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1150     {
1151         \__zrefclever_opt_tl_gset_if_new:cn
1152         {
1153             \__zrefclever_opt_varname_lang_default:enn
1154             { \l__zrefclever_setup_language_tl }
1155             { endrangefunc } { tl }
1156         }
1157         { __zrefclever_get_endrange_stripprefix }
1158         \__zrefclever_opt_tl_gclear_if_new:c
1159         {
1160             \__zrefclever_opt_varname_lang_default:enn
1161             { \l__zrefclever_setup_language_tl }
1162             { endrangeprop } { tl }
1163         }
1164     }
1165     {
1166         \__zrefclever_opt_tl_gset_if_new:cn
1167         {
1168             \__zrefclever_opt_varname_lang_type:eenn
1169             { \l__zrefclever_setup_language_tl }
1170             { \l__zrefclever_setup_type_tl }
1171             { endrangefunc } { tl }
1172         }
1173         { __zrefclever_get_endrange_stripprefix }
1174         \__zrefclever_opt_tl_gclear_if_new:c
1175         {
1176             \__zrefclever_opt_varname_lang_type:eenn
1177             { \l__zrefclever_setup_language_tl }
1178             { \l__zrefclever_setup_type_tl }
1179             { endrangeprop } { tl }
1180         }
1181     }
1182 }
1183
1184 { pagecomp }
1185 {
1186     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1187     {
1188         \__zrefclever_opt_tl_gset_if_new:cn
1189         {
1190             \__zrefclever_opt_varname_lang_default:enn
1191             { \l__zrefclever_setup_language_tl }
1192             { endrangefunc } { tl }
1193         }

```

```

1194         { __zrefclever_get_endrange_pagecomp }
1195     \__zrefclever_opt_tl_gclear_if_new:c
1196     {
1197         \__zrefclever_opt_varname_lang_default:enn
1198         { \l__zrefclever_setup_language_tl }
1199         { endrangeprop } { t1 }
1200     }
1201 }
1202 {
1203     \__zrefclever_opt_tl_gset_if_new:cn
1204     {
1205         \__zrefclever_opt_varname_lang_type:een
1206         { \l__zrefclever_setup_language_tl }
1207         { \l__zrefclever_setup_type_tl }
1208         { endrangefunc } { t1 }
1209     }
1210     { __zrefclever_get_endrange_pagecomp }
1211     \__zrefclever_opt_tl_gclear_if_new:c
1212     {
1213         \__zrefclever_opt_varname_lang_type:een
1214         { \l__zrefclever_setup_language_tl }
1215         { \l__zrefclever_setup_type_tl }
1216         { endrangeprop } { t1 }
1217     }
1218 }
1219 }
1220
1221 { pagecomp2 }
1222 {
1223     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1224     {
1225         \__zrefclever_opt_tl_gset_if_new:cn
1226         {
1227             \__zrefclever_opt_varname_lang_default:enn
1228             { \l__zrefclever_setup_language_tl }
1229             { endrangefunc } { t1 }
1230         }
1231         { __zrefclever_get_endrange_pagecomptwo }
1232         \__zrefclever_opt_tl_gclear_if_new:c
1233         {
1234             \__zrefclever_opt_varname_lang_default:enn
1235             { \l__zrefclever_setup_language_tl }
1236             { endrangeprop } { t1 }
1237         }
1238     }
1239     {
1240         \__zrefclever_opt_tl_gset_if_new:cn
1241         {
1242             \__zrefclever_opt_varname_lang_type:een
1243             { \l__zrefclever_setup_language_tl }
1244             { \l__zrefclever_setup_type_tl }
1245             { endrangefunc } { t1 }
1246         }
1247         { __zrefclever_get_endrange_pagecomptwo }

```

```

1248         \__zrefclever_opt_tl_gclear_if_new:c
1249         {
1250             \__zrefclever_opt_varname_lang_type:enn
1251             { \l__zrefclever_setup_language_tl }
1252             { \l__zrefclever_setup_type_tl }
1253             { endrangeprop } { tl }
1254         }
1255     }
1256 }
1257 }
1258 {
1259     \tl_if_empty:nTF {#1}
1260     {
1261         \msg_info:nnn { zref-clever }
1262         { endrange-property-undefined } {#1}
1263     }
1264     {
1265         \zref@ifpropundefined {#1}
1266         {
1267             \msg_info:nnn { zref-clever }
1268             { endrange-property-undefined } {#1}
1269         }
1270         {
1271             \tl_if_empty:nTF \l__zrefclever_setup_type_tl
1272             {
1273                 \__zrefclever_opt_tl_gset_if_new:cn
1274                 {
1275                     \__zrefclever_opt_varname_lang_default:enn
1276                     { \l__zrefclever_setup_language_tl }
1277                     { endrangefunc } { tl }
1278                 }
1279                 { __zrefclever_get_endrange_property }
1280                 \__zrefclever_opt_tl_gset_if_new:cn
1281                 {
1282                     \__zrefclever_opt_varname_lang_default:enn
1283                     { \l__zrefclever_setup_language_tl }
1284                     { endrangeprop } { tl }
1285                 }
1286                 {#1}
1287             }
1288             {
1289                 \__zrefclever_opt_tl_gset_if_new:cn
1290                 {
1291                     \__zrefclever_opt_varname_lang_type:enn
1292                     { \l__zrefclever_setup_language_tl }
1293                     { \l__zrefclever_setup_type_tl }
1294                     { endrangefunc } { tl }
1295                 }
1296                 { __zrefclever_get_endrange_property }
1297                 \__zrefclever_opt_tl_gset_if_new:cn
1298                 {
1299                     \__zrefclever_opt_varname_lang_type:enn
1300                     { \l__zrefclever_setup_language_tl }
1301                     { \l__zrefclever_setup_type_tl }

```

```

1302             { endrangeprop } { t1 }
1303         }
1304         {#1}
1305     }
1306 }
1307 }
1308 }
1309 } ,
1310 }
1311 \seq_map_inline:Nn
1312 \g__zrefclever_rf_opts_tl_type_names_seq
1313 {
1314     \keys_define:nn { zref-clever/langfile }
1315     {
1316         #1 .value_required:n = true ,
1317         #1 .code:n =
1318         {
1319             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1320             {
1321                 \msg_info:nnn { zref-clever }
1322                 { option-only-type-specific } {#1}
1323             }
1324             {
1325                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1326                 {
1327                     \__zrefclever_opt_tl_gset_if_new:cn
1328                     {
1329                         \__zrefclever_opt_varname_lang_type:een
1330                         { \l__zrefclever_setup_language_tl }
1331                         { \l__zrefclever_setup_type_tl }
1332                         {#1} { t1 }
1333                     }
1334                     {##1}
1335                 }
1336                 {
1337                     \__zrefclever_opt_tl_gset_if_new:cn
1338                     {
1339                         \__zrefclever_opt_varname_lang_type:een
1340                         { \l__zrefclever_setup_language_tl }
1341                         { \l__zrefclever_setup_type_tl }
1342                         { \l__zrefclever_lang_decl_case_tl - #1 } { t1 }
1343                     }
1344                     {##1}
1345                 }
1346             }
1347         } ,
1348     }
1349 }
1350 \seq_map_inline:Nn
1351 \g__zrefclever_rf_opts_seq_refbounds_seq
1352 {
1353     \keys_define:nn { zref-clever/langfile }
1354     {
1355         #1 .value_required:n = true ,

```

```

1356 #1 .code:n =
1357 {
1358   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1359   {
1360     \__zrefclever_opt_seq_if_set:cF
1361     {
1362       \__zrefclever_opt_varname_lang_default:enn
1363       { \l__zrefclever_setup_language_tl } {#1} { seq }
1364     }
1365     {
1366       \seq_gclear:N \g_tmpa_seq
1367       \__zrefclever_opt_seq_gset_clist_split:Nn
1368       \g_tmpa_seq {##1}
1369       \bool_lazy_or:nnTF
1370       { \tl_if_empty_p:n {##1} }
1371       {
1372         \int_compare_p:nNn
1373         { \seq_count:N \g_tmpa_seq } = { 4 }
1374       }
1375       {
1376         \__zrefclever_opt_seq_gset_eq:cN
1377         {
1378           \__zrefclever_opt_varname_lang_default:enn
1379           { \l__zrefclever_setup_language_tl }
1380           {#1} { seq }
1381         }
1382         \g_tmpa_seq
1383       }
1384       {
1385         \msg_info:nxxx { zref-clever }
1386         { refbounds-must-be-four }
1387         {#1} { \seq_count:N \g_tmpa_seq }
1388       }
1389     }
1390   }
1391   {
1392     \__zrefclever_opt_seq_if_set:cF
1393     {
1394       \__zrefclever_opt_varname_lang_type:eenn
1395       { \l__zrefclever_setup_language_tl }
1396       { \l__zrefclever_setup_type_tl } {#1} { seq }
1397     }
1398     {
1399       \seq_gclear:N \g_tmpa_seq
1400       \__zrefclever_opt_seq_gset_clist_split:Nn
1401       \g_tmpa_seq {##1}
1402       \bool_lazy_or:nnTF
1403       { \tl_if_empty_p:n {##1} }
1404       {
1405         \int_compare_p:nNn
1406         { \seq_count:N \g_tmpa_seq } = { 4 }
1407       }
1408       {
1409         \__zrefclever_opt_seq_gset_eq:cN

```

```

1410         {
1411             \__zrefclever_opt_varname_lang_type:enn
1412             { \l__zrefclever_setup_language_tl }
1413             { \l__zrefclever_setup_type_tl }
1414             {#1} { seq }
1415         }
1416         \g_tmpa_seq
1417     }
1418     {
1419         \msg_info:nxxx { zref-clever }
1420         { refbounds-must-be-four }
1421         {#1} { \seq_count:N \g_tmpa_seq }
1422     }
1423 }
1424 }
1425 } ,
1426 }
1427 }
1428 \seq_map_inline:Nn
1429 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1430 {
1431     \keys_define:nn { zref-clever/langfile }
1432     {
1433         #1 .choice: ,
1434         #1 / true .code:n =
1435         {
1436             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1437             {
1438                 \__zrefclever_opt_bool_if_set:cF
1439                 {
1440                     \__zrefclever_opt_varname_lang_default:enn
1441                     { \l__zrefclever_setup_language_tl }
1442                     {#1} { bool }
1443                 }
1444                 {
1445                     \__zrefclever_opt_bool_gset_true:c
1446                     {
1447                         \__zrefclever_opt_varname_lang_default:enn
1448                         { \l__zrefclever_setup_language_tl }
1449                         {#1} { bool }
1450                     }
1451                 }
1452             }
1453         }
1454         \__zrefclever_opt_bool_if_set:cF
1455         {
1456             \__zrefclever_opt_varname_lang_type:enn
1457             { \l__zrefclever_setup_language_tl }
1458             { \l__zrefclever_setup_type_tl }
1459             {#1} { bool }
1460         }
1461         {
1462             \__zrefclever_opt_bool_gset_true:c
1463             {

```



```

1464         \_zrefclever_opt_varname_lang_type:eenn
1465         { \l__zrefclever_setup_language_tl }
1466         { \l__zrefclever_setup_type_tl }
1467         {#1} { bool }
1468     }
1469 }
1470 }
1471 },
1472 #1 / false .code:n =
1473 {
1474     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1475     {
1476         \__zrefclever_opt_bool_if_set:cF
1477         {
1478             \__zrefclever_opt_varname_lang_default:enn
1479             { \l__zrefclever_setup_language_tl }
1480             {#1} { bool }
1481         }
1482         {
1483             \__zrefclever_opt_bool_gset_false:c
1484             {
1485                 \__zrefclever_opt_varname_lang_default:enn
1486                 { \l__zrefclever_setup_language_tl }
1487                 {#1} { bool }
1488             }
1489         }
1490     }
1491     {
1492         \__zrefclever_opt_bool_if_set:cF
1493         {
1494             \__zrefclever_opt_varname_lang_type:eenn
1495             { \l__zrefclever_setup_language_tl }
1496             { \l__zrefclever_setup_type_tl }
1497             {#1} { bool }
1498         }
1499         {
1500             \__zrefclever_opt_bool_gset_false:c
1501             {
1502                 \__zrefclever_opt_varname_lang_type:eenn
1503                 { \l__zrefclever_setup_language_tl }
1504                 { \l__zrefclever_setup_type_tl }
1505                 {#1} { bool }
1506             }
1507         }
1508     }
1509 },
1510 #1 .default:n = true ,
1511 no #1 .meta:n = { #1 = false } ,
1512 no #1 .value_forbidden:n = true ,
1513 }
1514 }

```

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.

```

1515 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1516 {
1517   \tl_const:cn
1518     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1519 }
1520 \keyval_parse:nnn
1521 { }
1522 { \__zrefclever_opt_tl_cset_fallback:nn }
1523 {
1524   tpairsep = {,~} ,
1525   tlistsep = {,~} ,
1526   tlastsep = {,~} ,
1527   notesep  = {~} ,
1528   namesep  = {\nobreakspace} ,
1529   pairsep  = {,~} ,
1530   listsep  = {,~} ,
1531   lastsep  = {,~} ,
1532   rangesep = {\textendash} ,
1533 }

```

## 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 }

1534 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1535 {
1536   \tl_if_empty:nTF {#3}
1537     { \prop_remove:Nn #1 {#2} }
1538     { \prop_put:Nnn #1 {#2} {#3} }
1539 }

```

(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`.

```

1540 \tl_new:N \l__zrefclever_ref_property_tl
1541 \keys_define:nn { zref-clever/reference }
1542 {
1543   ref .code:n =
1544   {
1545     \tl_if_empty:nTF {#1}
1546     {
1547       \msg_warning:nnn { zref-clever }
1548       { zref-property-undefined } {#1}
1549       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1550     }
1551     {
1552       \zref@ifpropundefined {#1}
1553       {
1554         \msg_warning:nnn { zref-clever }
1555         { zref-property-undefined } {#1}
1556         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1557       }
1558       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1559     }
1560   } ,
1561   ref .initial:n = default ,
1562   ref .value_required:n = true ,
1563   page .meta:n = { ref = page } ,
1564   page .value_forbidden:n = true ,
1565 }

```

### typeset option

```

1566 \bool_new:N \l__zrefclever_typeset_ref_bool
1567 \bool_new:N \l__zrefclever_typeset_name_bool
1568 \keys_define:nn { zref-clever/reference }
1569 {
1570   typeset .choice: ,
1571   typeset / both .code:n =
1572   {
1573     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1574     \bool_set_true:N \l__zrefclever_typeset_name_bool
1575   } ,
1576   typeset / ref .code:n =
1577   {
1578     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1579     \bool_set_false:N \l__zrefclever_typeset_name_bool
1580   } ,
1581   typeset / name .code:n =
1582   {
1583     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1584     \bool_set_true:N \l__zrefclever_typeset_name_bool
1585   } ,
1586   typeset .initial:n = both ,
1587   typeset .value_required:n = true ,
1588
1589   noname .meta:n = { typeset = ref } ,
1590   noname .value_forbidden:n = true ,

```

```

1591     noref .meta:n = { typeset = name } ,
1592     noref .value_forbidden:n = true ,
1593 }

```

#### **sort option**

```

1594 \bool_new:N \l__zrefclever_typeset_sort_bool
1595 \keys_define:nn { zref-clever/reference }
1596 {
1597     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1598     sort .initial:n = true ,
1599     sort .default:n = true ,
1600     nosort .meta:n = { sort = false },
1601     nosort .value_forbidden:n = true ,
1602 }

```

#### **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`.

```

1603 \seq_new:N \l__zrefclever_typesort_seq
1604 \keys_define:nn { zref-clever/reference }
1605 {
1606     typesort .code:n =
1607     {
1608         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1609         \seq_reverse:N \l__zrefclever_typesort_seq
1610     } ,
1611     typesort .initial:n =
1612     { part , chapter , section , paragraph },
1613     typesort .value_required:n = true ,
1614     notypesort .code:n =
1615     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1616     notypesort .value_forbidden:n = true ,
1617 }

```

#### **comp option**

```

1618 \bool_new:N \l__zrefclever_typeset_compress_bool
1619 \keys_define:nn { zref-clever/reference }
1620 {
1621     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1622     comp .initial:n = true ,
1623     comp .default:n = true ,
1624     nocomp .meta:n = { comp = false },
1625     nocomp .value_forbidden:n = true ,
1626 }

```

#### **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`.

```

1627 \NewHook { zref-clever/endrange-setup }
1628 \keys_define:nn { zref-clever/reference }
1629 {
1630   endrange .code:n =
1631   {
1632     \str_case:nnF {#1}
1633     {
1634       { ref }
1635       {
1636         \_zrefclever_opt_tl_clear:c
1637         {
1638           \_zrefclever_opt_varname_general:nn
1639           { endrangefunc } { tl }
1640         }
1641         \_zrefclever_opt_tl_clear:c
1642         {
1643           \_zrefclever_opt_varname_general:nn
1644           { endrangeprop } { tl }
1645         }
1646       }
1647     }
1648     { stripprefix }
1649     {

```

```

1650 \_zrefclever_opt_tl_set:cn
1651 {
1652   \_zrefclever_opt_varname_general:nn
1653   { endrangefunc } { t1 }
1654 }
1655 { \_zrefclever_get_endrange_stripprefix }
1656 \_zrefclever_opt_tl_clear:c
1657 {
1658   \_zrefclever_opt_varname_general:nn
1659   { endrangeprop } { t1 }
1660 }
1661 }
1662
1663 { pagecomp }
1664 {
1665   \_zrefclever_opt_tl_set:cn
1666   {
1667     \_zrefclever_opt_varname_general:nn
1668     { endrangefunc } { t1 }
1669   }
1670   { \_zrefclever_get_endrange_pagecomp }
1671   \_zrefclever_opt_tl_clear:c
1672   {
1673     \_zrefclever_opt_varname_general:nn
1674     { endrangeprop } { t1 }
1675   }
1676 }
1677
1678 { pagecomp2 }
1679 {
1680   \_zrefclever_opt_tl_set:cn
1681   {
1682     \_zrefclever_opt_varname_general:nn
1683     { endrangefunc } { t1 }
1684   }
1685   { \_zrefclever_get_endrange_pagecomptwo }
1686   \_zrefclever_opt_tl_clear:c
1687   {
1688     \_zrefclever_opt_varname_general:nn
1689     { endrangeprop } { t1 }
1690   }
1691 }
1692
1693 { unset }
1694 {
1695   \_zrefclever_opt_tl_unset:c
1696   {
1697     \_zrefclever_opt_varname_general:nn
1698     { endrangefunc } { t1 }
1699   }
1700   \_zrefclever_opt_tl_unset:c
1701   {
1702     \_zrefclever_opt_varname_general:nn
1703     { endrangeprop } { t1 }

```

```

1704         }
1705     }
1706 }
1707 {
1708     \tl_if_empty:nTF {#1}
1709     {
1710         \msg_warning:nnn { zref-clever }
1711         { endrange-property-undefined } {#1}
1712     }
1713     {
1714         \zref@ifpropundefined {#1}
1715         {
1716             \msg_warning:nnn { zref-clever }
1717             { endrange-property-undefined } {#1}
1718         }
1719         {
1720             \__zrefclever_opt_tl_set:cn
1721             {
1722                 \__zrefclever_opt_varname_general:nn
1723                 { endrangefunc } { tl }
1724             }
1725             { __zrefclever_get_endrange_property }
1726             \__zrefclever_opt_tl_set:cn
1727             {
1728                 \__zrefclever_opt_varname_general:nn
1729                 { endrangeprop } { tl }
1730             }
1731             {#1}
1732         }
1733     }
1734 }
1735 },
1736 endrange .value_required:n = true ,
1737 }
1738 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1739 {
1740     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1741     {
1742         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1743         {
1744             \__zrefclever_extract_default:Nnvn #3
1745             {#2} { \l__zrefclever_ref_property_tl } { }
1746         }
1747         { \tl_set:Nn #3 { zc@missingproperty } }
1748     }
1749     {
1750         \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1751         {

```

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.

```

1752 \bool_if:NTF \l__zrefclever_typeset_range_bool
1753 {
1754   \group_begin:
1755   \bool_set_false:N \l_tmpa_bool
1756   \exp_args:Nxx \tl_if_eq:nnT
1757   {
1758     \__zrefclever_extract_unexp:nnn
1759     {#1} { externaldocument } { }
1760   }
1761   {
1762     \__zrefclever_extract_unexp:nnn
1763     {#2} { externaldocument } { }
1764   }
1765   {
1766     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1767     {
1768       \exp_args:Nxx \tl_if_eq:nnT
1769       {
1770         \__zrefclever_extract_unexp:nnn
1771         {#1} { zc@pgfmt } { }
1772       }
1773       {
1774         \__zrefclever_extract_unexp:nnn
1775         {#2} { zc@pgfmt } { }
1776       }
1777       { \bool_set_true:N \l_tmpa_bool }
1778     }
1779     {
1780       \exp_args:Nxx \tl_if_eq:nnT
1781       {
1782         \__zrefclever_extract_unexp:nnn
1783         {#1} { zc@counter } { }
1784       }
1785       {
1786         \__zrefclever_extract_unexp:nnn
1787         {#2} { zc@counter } { }
1788       }
1789       {
1790         \exp_args:Nxx \tl_if_eq:nnT
1791         {
1792           \__zrefclever_extract_unexp:nnn
1793           {#1} { zc@enclval } { }
1794         }
1795         {
1796           \__zrefclever_extract_unexp:nnn
1797           {#2} { zc@enclval } { }
1798         }
1799         { \bool_set_true:N \l_tmpa_bool }
1800       }
1801     }
1802   }
1803   \bool_if:NTF \l_tmpa_bool
1804   {
1805     \__zrefclever_extract_default:Nnvn \l_tmpb_tl

```



```

1806         {#2} { l__zrefclever_endrangeprop_tl } { }
1807     }
1808     {
1809         \zref@ifrefcontainsprop
1810         {#2} { \l__zrefclever_ref_property_tl }
1811         {
1812             \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1813             {#2} { l__zrefclever_ref_property_tl } { }
1814         }
1815         { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1816     }
1817     \exp_args:NNNV
1818     \group_end:
1819     \tl_set:Nn #3 \l_tmpb_tl
1820 }
1821 {
1822     \__zrefclever_extract_default:Nnvn #3
1823     {#2} { l__zrefclever_endrangeprop_tl } { }
1824 }
1825 }
1826 {
1827     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1828     {
1829         \__zrefclever_extract_default:Nnvn #3
1830         {#2} { l__zrefclever_ref_property_tl } { }
1831     }
1832     { \tl_set:Nn #3 { zc@missingproperty } }
1833 }
1834 }
1835 }
1836 \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>.

```

1837 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1838 {
1839     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1840     {
1841         \group_begin:
1842         \UseHook { zref-clever/endrange-setup }
1843         \tl_set:Nx \l_tmpa_tl
1844         {
1845             \__zrefclever_extract:nnn
1846             {#1} { \l__zrefclever_ref_property_tl } { }
1847         }
1848         \tl_set:Nx \l_tmpb_tl
1849         {
1850             \__zrefclever_extract:nnn
1851             {#2} { \l__zrefclever_ref_property_tl } { }
1852         }
1853         \bool_set_false:N \l_tmpa_bool
1854         \bool_until_do:Nn \l_tmpa_bool
1855         {
1856             \exp_args:Nxx \tl_if_eq:nnTF

```

```

1857         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1858         {
1859             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1860             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1861             \tl_if_empty:NT \l_tmpb_tl
1862                 { \bool_set_true:N \l_tmpa_bool }
1863         }
1864         { \bool_set_true:N \l_tmpa_bool }
1865     }
1866     \exp_args:NNNV
1867     \group_end:
1868     \tl_set:Nn #3 \l_tmpb_tl
1869 }
1870 { \tl_set:Nn #3 { zc@missingproperty } }
1871 }
1872 \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>).

```

1873 \prg_new_protected_conditional:Npnn
1874   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1875 {
1876     \regex_match:nnTF { \A\d+\Z } {#1}
1877     { \prg_return_true: }
1878     { \prg_return_false: }
1879 }
1880 \prg_generate_conditional_variant:Nnn
1881   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

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

```

1882 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1883 {
1884     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1885     {
1886         \group_begin:
1887         \UseHook { zref-clever/endrange-setup }
1888         \tl_set:Nx \l_tmpa_tl
1889         {
1890             \__zrefclever_extract:nnn
1891             {#1} { \l__zrefclever_ref_property_tl } { }
1892         }
1893         \tl_set:Nx \l_tmpb_tl
1894         {
1895             \__zrefclever_extract:nnn
1896             {#2} { \l__zrefclever_ref_property_tl } { }
1897         }
1898         \bool_set_false:N \l_tmpa_bool
1899         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1900         {
1901             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1902             { \bool_set_true:N \l_tmpa_bool }
1903         }
1904         { \bool_set_true:N \l_tmpa_bool }
1905         \bool_until_do:Nn \l_tmpa_bool

```

```

1906     {
1907         \exp_args:Nxx \tl_if_eq:nnTF
1908         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1909         {
1910             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1911             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1912             \tl_if_empty:NT \l_tmpb_tl
1913             { \bool_set_true:N \l_tmpa_bool }
1914         }
1915         { \bool_set_true:N \l_tmpa_bool }
1916     }
1917     \exp_args:NNNV
1918     \group_end:
1919     \tl_set:Nn #3 \l_tmpb_tl
1920 }
1921 { \tl_set:Nn #3 { zc@missingproperty } }
1922 }
1923 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1924 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1925 {
1926     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1927     {
1928         \group_begin:
1929         \UseHook { zref-clever/endrange-setup }
1930         \tl_set:Nx \l_tmpa_tl
1931         {
1932             \__zrefclever_extract:nnn
1933             {#1} { \l__zrefclever_ref_property_tl } { }
1934         }
1935         \tl_set:Nx \l_tmpb_tl
1936         {
1937             \__zrefclever_extract:nnn
1938             {#2} { \l__zrefclever_ref_property_tl } { }
1939         }
1940         \bool_set_false:N \l_tmpa_bool
1941         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1942         {
1943             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1944             { \bool_set_true:N \l_tmpa_bool }
1945         }
1946         { \bool_set_true:N \l_tmpa_bool }
1947         \bool_until_do:Nn \l_tmpa_bool
1948         {
1949             \exp_args:Nxx \tl_if_eq:nnTF
1950             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1951             {
1952                 \bool_lazy_or:nnTF
1953                 { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1954                 { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1955                 {
1956                     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1957                     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1958                 }
1959                 { \bool_set_true:N \l_tmpa_bool }

```

```

1960         }
1961         { \bool_set_true:N \l_tmpa_bool }
1962     }
1963     \exp_args:NNNV
1964     \group_end:
1965     \tl_set:Nn #3 \l_tmpb_tl
1966 }
1967 { \tl_set:Nn #3 { zc@missingproperty } }
1968 }
1969 \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`.

```

1970 \bool_new:N \l__zrefclever_typeset_range_bool
1971 \keys_define:nn { zref-clever/reference }
1972 {
1973     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1974     range .initial:n = false ,
1975     range .default:n = true ,
1976 }

```

### 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`.

```

1977 \bool_new:N \l__zrefclever_capfirst_bool
1978 \keys_define:nn { zref-clever/reference }
1979 {
1980     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1981     capfirst .initial:n = false ,
1982     capfirst .default:n = true ,
1983 }

```

### 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`.

```

1984 \bool_new:N \l__zrefclever_noabbrev_first_bool
1985 \keys_define:nn { zref-clever/reference }
1986 {
1987     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1988     noabbrevfirst .initial:n = false ,
1989     noabbrevfirst .default:n = true ,
1990 }

```

### S option

```

1991 \keys_define:nn { zref-clever/reference }
1992 {
1993     S .meta:n =

```

```

1994     { capfirst = {#1} , noabbrevfirst = {#1} },
1995     S .default:n = true ,
1996 }

```

### hyperref option

```

1997 \bool_new:N \l__zrefclever_hyperlink_bool
1998 \bool_new:N \l__zrefclever_hyperref_warn_bool
1999 \keys_define:nn { zref-clever/reference }
2000 {
2001     hyperref .choice: ,
2002     hyperref / auto .code:n =
2003     {
2004         \bool_set_true:N \l__zrefclever_hyperlink_bool
2005         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2006     } ,
2007     hyperref / true .code:n =
2008     {
2009         \bool_set_true:N \l__zrefclever_hyperlink_bool
2010         \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2011     } ,
2012     hyperref / false .code:n =
2013     {
2014         \bool_set_false:N \l__zrefclever_hyperlink_bool
2015         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2016     } ,
2017     hyperref .initial:n = auto ,
2018     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.

```

2019     nohyperref .meta:n = { hyperref = false } ,
2020     nohyperref .value_forbidden:n = true ,
2021 }
2022 \AddToHook { begindocument }
2023 {
2024     \__zrefclever_if_package_loaded:nTF { hyperref }
2025     {
2026         \bool_if:NT \l__zrefclever_hyperlink_bool
2027         { \RequirePackage { zref-hyperref } }
2028     }
2029     {
2030         \bool_if:NT \l__zrefclever_hyperref_warn_bool
2031         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2032         \bool_set_false:N \l__zrefclever_hyperlink_bool
2033     }
2034     \keys_define:nn { zref-clever/reference }
2035     {
2036         hyperref .code:n =
2037         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2038         nohyperref .code:n =
2039         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,

```

```

2040     }
2041 }

```

### nameinlink option

```

2042 \str_new:N \l__zrefclever_nameinlink_str
2043 \keys_define:nn { zref-clever/reference }
2044 {
2045   nameinlink .choice: ,
2046   nameinlink / true .code:n =
2047     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2048   nameinlink / false .code:n =
2049     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2050   nameinlink / single .code:n =
2051     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2052   nameinlink / tsingle .code:n =
2053     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2054   nameinlink .initial:n = tsingle ,
2055   nameinlink .default:n = true ,
2056 }

```

### preposinlink option (deprecated)

```

2057 \keys_define:nn { zref-clever/reference }
2058 {
2059   preposinlink .code:n =
2060   {
2061     % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2062     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2063     { preposinlink } { refbounds }
2064   } ,
2065 }

```

### 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`.

```

2066 \AddToHook { begindocument }
2067 {

```

```

2068 \__zrefclever_if_package_loaded:nTF { babel }
2069 {
2070   \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
2071   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2072 }
2073 {
2074   \__zrefclever_if_package_loaded:nTF { polyglossia }
2075   {
2076     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2077     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2078   }
2079   {
2080     \tl_set:Nn \l__zrefclever_current_language_tl { english }
2081     \tl_set:Nn \l__zrefclever_main_language_tl { english }
2082   }
2083 }
2084 }
2085 \keys_define:nn { zref-clever/reference }
2086 {
2087   lang .code:n =
2088   {
2089     \AddToHook { begindocument }
2090     {
2091       \str_case:nnF {#1}
2092       {
2093         { current }
2094         {
2095           \tl_set:Nn \l__zrefclever_ref_language_tl
2096             { \l__zrefclever_current_language_tl }
2097         }
2098
2099         { main }
2100         {
2101           \tl_set:Nn \l__zrefclever_ref_language_tl
2102             { \l__zrefclever_main_language_tl }
2103         }
2104       }
2105       {
2106         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2107         \__zrefclever_language_if_declared:nF {#1}
2108         {
2109           \msg_warning:nnn { zref-clever }
2110             { unknown-language-opt } {#1}
2111         }
2112       }
2113       \__zrefclever_provide_langfile:x
2114         { \l__zrefclever_ref_language_tl }
2115     }
2116   } ,
2117   lang .initial:n = current ,
2118   lang .value_required:n = true ,
2119 }
2120 \AddToHook { begindocument / before }

```

```

2121 {
2122   \AddToHook { begindocument }
2123   {

```

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.

```

2124     \keys_define:nn { zref-clever/reference }
2125     {
2126       lang .code:n =
2127       {
2128         \str_case:nnF {#1}
2129         {
2130           { current }
2131           {
2132             \tl_set:Nn \l__zrefclever_ref_language_tl
2133             { \l__zrefclever_current_language_tl }
2134           }
2135
2136           { main }
2137           {
2138             \tl_set:Nn \l__zrefclever_ref_language_tl
2139             { \l__zrefclever_main_language_tl }
2140           }
2141         }
2142         {
2143           \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2144           \__zrefclever_language_if_declared:nF {#1}
2145           {
2146             \msg_warning:nnn { zref-clever }
2147             { unknown-language-opt } {#1}
2148           }
2149         }
2150       } ,
2151     }
2152   }
2153 }

```

### 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.

```

2154 \tl_new:N \l__zrefclever_ref_decl_case_tl
2155 \keys_define:nn { zref-clever/reference }
2156 {
2157   d .code:n =
2158   { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2159 }
2160 \AddToHook { begindocument }

```



```

2161 {
2162   \keys_define:nn { zref-clever/reference }
2163   {

```

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

```

2164     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2165     d .value_required:n = true ,
2166   }
2167 }

```

### nudge & co. options

```

2168 \bool_new:N \l__zrefclever_nudge_enabled_bool
2169 \bool_new:N \l__zrefclever_nudge_multitype_bool
2170 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2171 \bool_new:N \l__zrefclever_nudge_singular_bool
2172 \bool_new:N \l__zrefclever_nudge_gender_bool
2173 \tl_new:N \l__zrefclever_ref_gender_tl
2174 \keys_define:nn { zref-clever/reference }
2175 {
2176   nudge .choice: ,
2177   nudge / true .code:n =
2178   { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2179   nudge / false .code:n =
2180   { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2181   nudge / ifdraft .code:n =
2182   {
2183     \ifdraft
2184       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2185       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2186   } ,
2187   nudge / iffinaal .code:n =
2188   {
2189     \ifoptionfinal
2190       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2191       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2192   } ,
2193   nudge .initial:n = false ,
2194   nudge .default:n = true ,
2195   nonnudge .meta:n = { nudge = false } ,
2196   nonnudge .value_forbidden:n = true ,
2197   nudgeif .code:n =
2198   {
2199     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2200     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2201     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2202     \clist_map_inline:nn {#1}
2203     {
2204       \str_case:nnF {##1}
2205       {
2206         { multitype }
2207         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2208         { comptosing }

```

```

2209         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2210         { gender }
2211         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2212         { all }
2213         {
2214             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2215             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2216             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2217         }
2218     }
2219     {
2220         \msg_warning:nnn { zref-clever }
2221         { nudgeif-unknown-value } {##1}
2222     }
2223 }
2224 },
2225 nudgeif .value_required:n = true ,
2226 nudgeif .initial:n = all ,
2227 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2228 sg .initial:n = false ,
2229 sg .default:n = true ,
2230 g .code:n =
2231     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2232 }
2233 \AddToHook { begindocument }
2234 {
2235     \keys_define:nn { zref-clever/reference }
2236     {

```

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

```

2237         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2238         g .value_required:n = true ,
2239     }
2240 }

```

### font option

`font` *can't be used as a package option*, since the options get expanded by L<sup>A</sup>T<sub>E</sub>X before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can be set in `\zceref` and, for global settings, with `\zcsetup`. Note that, technically, the “raw” options are already available as `\@raw@opt@{package}.sty` (helpful comment by David Carlisle at <https://tex.stackexchange.com/a/618439>).

```

2241 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2242 \keys_define:nn { zref-clever/reference }
2243 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

### titleref option

```

2244 \keys_define:nn { zref-clever/reference }
2245 {
2246     titleref .code:n = { \RequirePackage { zref-titleref } } ,
2247     titleref .value_forbidden:n = true ,
2248 }
2249 \AddToHook { begindocument }

```

```

2250 {
2251   \keys_define:nn { zref-clever/reference }
2252   {
2253     titleref .code:n =
2254       { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
2255   }
2256 }

```

#### vario option

```

2257 \keys_define:nn { zref-clever/reference }
2258 {
2259   vario .code:n = { \RequirePackage { zref-vario } } ,
2260   vario .value_forbidden:n = true ,
2261 }
2262 \AddToHook { begindocument }
2263 {
2264   \keys_define:nn { zref-clever/reference }
2265   {
2266     vario .code:n =
2267     {
2268       \msg_warning:nnn { zref-clever }
2269       { option-preamble-only } { vario }
2270     }
2271   }
2272 }

```

#### note option

```

2273 \tl_new:N \l__zrefclever_zcref_note_tl
2274 \keys_define:nn { zref-clever/reference }
2275 {
2276   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2277   note .value_required:n = true ,
2278 }

```

#### check option

Integration with zref-check.

```

2279 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2280 \bool_new:N \l__zrefclever_zcref_with_check_bool
2281 \keys_define:nn { zref-clever/reference }
2282 {
2283   check .code:n = { \RequirePackage { zref-check } } ,
2284   check .value_forbidden:n = true ,
2285 }
2286 \AddToHook { begindocument }
2287 {
2288   \__zrefclever_if_package_loaded:nTF { zref-check }
2289   {
2290     \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2291     {
2292       \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2293       \keys_define:nn { zref-clever/reference }
2294       {
2295         check .code:n =

```

```

2296         {
2297             \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2298             \keys_set:nn { zref-check / zcheck } {#1}
2299         } ,
2300         check .value_required:n = true ,
2301     }
2302 }
2303 {
2304     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2305     \keys_define:nn { zref-clever/reference }
2306     {
2307         check .code:n =
2308         {
2309             \msg_warning:nnn { zref-clever }
2310             { zref-check-too-old } { 2021-09-16~v0.2.1 }
2311         } ,
2312     }
2313 }
2314 }
2315 {
2316     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2317     \keys_define:nn { zref-clever/reference }
2318     {
2319         check .code:n =
2320         { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2321     }
2322 }
2323 }

```

### 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`.

```

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

```

```

2342 {
2343     subsection    = section ,
2344     subsubsection = section ,
2345     subparagraph  = paragraph ,
2346     enumi         = item ,
2347     enumii        = item ,
2348     enumiii       = item ,
2349     enumiv        = item ,
2350     mpfootnote    = footnote ,
2351 } ,
2352 }

```

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_reset_by:n` to populate the `zc@enclval` property, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resetters_seq` with the `counterresetby` option.

```

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

```

```

2369     part ,
2370     chapter ,
2371     section ,
2372     subsection ,
2373     subsubsection ,
2374     paragraph ,
2375     subparagraph ,
2376 },
2377 counterresetters .value_required:n = true ,
2378 }

```

### counterresetby option

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

```

2379 \prop_new:N \l__zrefclever_counter_resetby_prop
2380 \keys_define:nn { zref-clever/label }
2381 {
2382     counterresetby .code:n =
2383     {
2384         \keyval_parse:nnn
2385         {
2386             \msg_warning:nnn { zref-clever }
2387             { key-requires-value } { counterresetby }
2388         }
2389         {
2390             \__zrefclever_prop_put_non_empty:Nnn
2391             \l__zrefclever_counter_resetby_prop
2392         }
2393         {#1}
2394     } ,
2395     counterresetby .value_required:n = true ,
2396     counterresetby .initial:n =
2397     {

```

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.

```

2398     enumii = enumi ,
2399     enumiii = enumii ,
2400     enumiv = enumiii ,
2401 } ,
2402 }

```

### 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.

```

2403 \tl_new:N \l__zrefclever_current_counter_tl
2404 \keys_define:nn { zref-clever/label }
2405 {
2406   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2407   currentcounter .value_required:n = true ,
2408   currentcounter .initial:n = \@currentcounter ,
2409 }

nocompat option

2410 \bool_new:N \g__zrefclever_nocompat_bool
2411 \seq_new:N \g__zrefclever_nocompat_modules_seq
2412 \keys_define:nn { zref-clever/reference }
2413 {
2414   nocompat .code:n =
2415   {
2416     \tl_if_empty:nTF {#1}
2417     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2418     {
2419       \clist_map_inline:nn {#1}
2420       {
2421         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2422         {
2423           \seq_gput_right:Nn
2424             \g__zrefclever_nocompat_modules_seq {##1}
2425         }
2426       }
2427     }
2428   } ,
2429 }
2430 \AddToHook { begindocument }
2431 {
2432   \keys_define:nn { zref-clever/reference }
2433   {
2434     nocompat .code:n =
2435     {
2436       \msg_warning:nnn { zref-clever }
2437         { option-preamble-only } { nocompat }
2438     }
2439   }
2440 }
2441 \AtEndOfPackage
2442 {
2443   \AddToHook { begindocument }
2444   {
2445     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2446       { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2447   }
2448 }

```

`\__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 `\g__zrefclever_nocompat_modules_seq` is not in `\l__zrefclever_nocompat_modules_seq`. The `begindocument` hook is needed so that we can have the option functional along the whole preamble, not just at package load time. This re-

quirement 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>}

2449 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2450 {
2451   \AddToHook { begindocument }
2452   {
2453     \bool_if:NF \g__zrefclever_nocompat_bool
2454     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2455     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2456   }
2457 }

```

(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.

```

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

```



```

2485         { refpre } { refbounds }
2486     } ,
2487     refpos .code:n =
2488     {
2489         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2490         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2491         { refpos } { refbounds }
2492     } ,
2493     preref .code:n =
2494     {
2495         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2496         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2497         { preref } { refbounds }
2498     } ,
2499     postref .code:n =
2500     {
2501         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2502         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2503         { postref } { refbounds }
2504     } ,
2505 }
2506 \seq_map_inline:Nn
2507 \g__zrefclever_rf_opts_seq_refbounds_seq
2508 {
2509     \keys_define:nn { zref-clever/reference }
2510     {
2511         #1 .default:o = \c_novalue_tl ,
2512         #1 .code:n =
2513         {
2514             \tl_if_novalue:nTF {##1}
2515             {
2516                 \__zrefclever_opt_seq_unset:c
2517                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2518             }
2519             {
2520                 \seq_clear:N \l_tmpa_seq
2521                 \__zrefclever_opt_seq_set_clist_split:Nn
2522                 \l_tmpa_seq {##1}
2523                 \bool_lazy_or:nnTF
2524                 { \tl_if_empty_p:n {##1} }
2525                 { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2526                 {
2527                     \__zrefclever_opt_seq_set_eq:cN
2528                     { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2529                     \l_tmpa_seq
2530                 }
2531                 {
2532                     \msg_warning:nxxx { zref-clever }
2533                     { refbounds-must-be-four }
2534                     {#1} { \seq_count:N \l_tmpa_seq }
2535                 }
2536             }
2537         } ,
2538     }

```

```

2539 }
2540 \seq_map_inline:Nn
2541   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2542   {
2543     \keys_define:nn { zref-clever/reference }
2544     {
2545       #1 .choice: ,
2546       #1 / true .code:n =
2547       {
2548         \__zrefclever_opt_bool_set_true:c
2549         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2550       } ,
2551       #1 / false .code:n =
2552       {
2553         \__zrefclever_opt_bool_set_false:c
2554         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2555       } ,
2556       #1 / unset .code:n =
2557       {
2558         \__zrefclever_opt_bool_unset:c
2559         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2560       } ,
2561       #1 .default:n = true ,
2562       no #1 .meta:n = { #1 = false } ,
2563       no #1 .value_forbidden:n = true ,
2564     }
2565   }

```

## Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: **label** and **reference**. Currently, the only use of this selection is the ability to exclude label related options from `\zcref`'s options. Anyway, for load-time package options and for `\zcsetup` we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```

2566 \keys_define:nn { }
2567 {
2568   zref-clever/zcsetup .inherit:n =
2569   {
2570     zref-clever/label ,
2571     zref-clever/reference ,
2572   }
2573 }

```

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

```

2574 \ProcessKeysOptions { zref-clever/zcsetup }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

2575 \NewDocumentCommand \zcsetup { m }
2576 { \_zrefclever_zcsetup:n {#1} }

(End definition for \zcsetup.)

```

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

```

\_zrefclever_zcsetup:n{<options>}

2577 \cs_new_protected:Npn \_zrefclever_zcsetup:n #1
2578 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2579 \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>}

2580 \NewDocumentCommand \zcRefTypeSetup { m m }
2581 {
2582   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2583   \keys_set:nn { zref-clever/typesetup } {#2}
2584   \tl_clear:N \l__zrefclever_setup_type_tl
2585 }

(End definition for \zcRefTypeSetup.)

2586 \seq_map_inline:Nn
2587 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2588 {
2589   \keys_define:nn { zref-clever/typesetup }
2590   {
2591     #1 .code:n =
2592     {
2593       \msg_warning:nnn { zref-clever }
2594       { option-not-type-specific } {#1}
2595     } ,
2596   }
2597 }
2598 \seq_map_inline:Nn
2599 \g__zrefclever_rf_opts_tl_typesetup_seq
2600 {
2601   \keys_define:nn { zref-clever/typesetup }
2602   {
2603     #1 .default:o = \c_novalue_tl ,
2604     #1 .code:n =

```

```

2605     {
2606         \tl_if_novalue:nTF {##1}
2607         {
2608             \__zrefclever_opt_tl_unset:c
2609             {
2610                 \__zrefclever_opt_varname_type:enn
2611                 { \l__zrefclever_setup_type_tl } {#1} { t1 }
2612             }
2613         }
2614         {
2615             \__zrefclever_opt_tl_set:cn
2616             {
2617                 \__zrefclever_opt_varname_type:enn
2618                 { \l__zrefclever_setup_type_tl } {#1} { t1 }
2619             }
2620             {##1}
2621         }
2622     } ,
2623 }
2624 }
2625 \keys_define:nn { zref-clever/typesetup }
2626 {
2627     endrange .code:n =
2628     {
2629         \str_case:nnF {#1}
2630         {
2631             { ref }
2632             {
2633                 \__zrefclever_opt_tl_clear:c
2634                 {
2635                     \__zrefclever_opt_varname_type:enn
2636                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2637                 }
2638                 \__zrefclever_opt_tl_clear:c
2639                 {
2640                     \__zrefclever_opt_varname_type:enn
2641                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2642                 }
2643             }
2644         }
2645         { stripprefix }
2646         {
2647             \__zrefclever_opt_tl_set:cn
2648             {
2649                 \__zrefclever_opt_varname_type:enn
2650                 { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2651             }
2652             { __zrefclever_get_endrange_stripprefix }
2653             \__zrefclever_opt_tl_clear:c
2654             {
2655                 \__zrefclever_opt_varname_type:enn
2656                 { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2657             }
2658         }

```

```

2659 { pagecomp }
2660 {
2661   \_zrefclever_opt_t1_set:cn
2662   {
2663     \_zrefclever_opt_varname_type:enn
2664     { \l\_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2665   }
2666   { \_zrefclever_get_endrange_pagecomp }
2667   \_zrefclever_opt_t1_clear:c
2668   {
2669     \_zrefclever_opt_varname_type:enn
2670     { \l\_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2671   }
2672 }
2673
2674 { pagecomp2 }
2675 {
2676   \_zrefclever_opt_t1_set:cn
2677   {
2678     \_zrefclever_opt_varname_type:enn
2679     { \l\_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2680   }
2681   { \_zrefclever_get_endrange_pagecomptwo }
2682   \_zrefclever_opt_t1_clear:c
2683   {
2684     \_zrefclever_opt_varname_type:enn
2685     { \l\_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2686   }
2687 }
2688
2689 { unset }
2690 {
2691   \_zrefclever_opt_t1_unset:c
2692   {
2693     \_zrefclever_opt_varname_type:enn
2694     { \l\_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2695   }
2696   \_zrefclever_opt_t1_unset:c
2697   {
2698     \_zrefclever_opt_varname_type:enn
2699     { \l\_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2700   }
2701 }
2702 }
2703 {
2704   \t1_if_empty:nTF {#1}
2705   {
2706     \msg_warning:nnn { zref-clever }
2707     { endrange-property-undefined } {#1}
2708   }
2709   {
2710     \zref@ifpropundefined {#1}
2711     {
2712

```

```

2713         \msg_warning:nnn { zref-clever }
2714         { endrange-property-undefined } {#1}
2715     }
2716     {
2717         \__zrefclever_opt_tl_set:cn
2718         {
2719             \__zrefclever_opt_varname_type:enn
2720             { \l__zrefclever_setup_type_tl }
2721             { endrangefunc } { tl }
2722         }
2723         { \__zrefclever_get_endrange_property }
2724         \__zrefclever_opt_tl_set:cn
2725         {
2726             \__zrefclever_opt_varname_type:enn
2727             { \l__zrefclever_setup_type_tl }
2728             { endrangeprop } { tl }
2729         }
2730         {#1}
2731     }
2732 }
2733 }
2734 } ,
2735 endrange .value_required:n = true ,
2736 }
2737 \keys_define:nn { zref-clever/typesetup }
2738 {
2739     refpre .code:n =
2740     {
2741         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2742         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2743         { refpre } { refbounds }
2744     } ,
2745     refpos .code:n =
2746     {
2747         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2748         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2749         { refpos } { refbounds }
2750     } ,
2751     preref .code:n =
2752     {
2753         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2754         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2755         { preref } { refbounds }
2756     } ,
2757     postref .code:n =
2758     {
2759         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2760         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2761         { postref } { refbounds }
2762     } ,
2763 }
2764 \seq_map_inline:Nn
2765 \g__zrefclever_rf_opts_seq_refbounds_seq
2766 {

```

```

2767 \keys_define:nn { zref-clever/typesetup }
2768 {
2769   #1 .default:o = \c_novalue_tl ,
2770   #1 .code:n =
2771   {
2772     \tl_if_novalue:nTF {##1}
2773     {
2774       \__zrefclever_opt_seq_unset:c
2775       {
2776         \__zrefclever_opt_varname_type:enn
2777         { \l__zrefclever_setup_type_tl } {#1} { seq }
2778       }
2779     }
2780     {
2781       \seq_clear:N \l_tmpa_seq
2782       \__zrefclever_opt_seq_set_clist_split:Nn
2783       \l_tmpa_seq {##1}
2784       \bool_lazy_or:nnTF
2785       { \tl_if_empty_p:n {##1} }
2786       { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2787       {
2788         \__zrefclever_opt_seq_set_eq:cN
2789         {
2790           \__zrefclever_opt_varname_type:enn
2791           { \l__zrefclever_setup_type_tl } {#1} { seq }
2792         }
2793         \l_tmpa_seq
2794       }
2795       {
2796         \msg_warning:nxxx { zref-clever }
2797         { refbounds-must-be-four }
2798         {#1} { \seq_count:N \l_tmpa_seq }
2799       }
2800     }
2801   } ,
2802 }
2803 }
2804 \seq_map_inline:Nn
2805 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2806 {
2807   \keys_define:nn { zref-clever/typesetup }
2808   {
2809     #1 .choice: ,
2810     #1 / true .code:n =
2811     {
2812       \__zrefclever_opt_bool_set_true:c
2813       {
2814         \__zrefclever_opt_varname_type:enn
2815         { \l__zrefclever_setup_type_tl }
2816         {#1} { bool }
2817       }
2818     } ,
2819     #1 / false .code:n =
2820     {

```

```

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

```

### 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>}
2842 \NewDocumentCommand \zcLanguageSetup { m m }
2843 {
2844     \group_begin:
2845     \__zrefclever_language_if_declared:nTF {#1}
2846     {
2847         \tl_clear:N \l__zrefclever_setup_type_tl
2848         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2849         \__zrefclever_opt_seq_get:cNF
2850         {
2851             \__zrefclever_opt_varname_language:nnn
2852             {#1} { declension } { seq }
2853         }
2854         \l__zrefclever_lang_declension_seq
2855         { \seq_clear:N \l__zrefclever_lang_declension_seq }
2856         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2857         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2858         {
2859             \seq_get_left:NN \l__zrefclever_lang_declension_seq
2860             \l__zrefclever_lang_decl_case_tl
2861         }

```



```

2862     \__zrefclever_opt_seq_get:cNF
2863     {
2864         \__zrefclever_opt_varname_language:nnn
2865         {#1} { gender } { seq }
2866     }
2867     \l__zrefclever_lang_gender_seq
2868     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2869     \keys_set:nn { zref-clever/langsetup } {#2}
2870 }
2871 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2872 \group_end:
2873 }
2874 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2875 \keys_define:nn { zref-clever/langsetup }
2876 {
2877     type .code:n =
2878     {
2879         \tl_if_empty:NTF {#1}
2880         { \tl_clear:N \l__zrefclever_setup_type_tl }
2881         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2882     } ,
2883
2884     case .code:n =
2885     {
2886         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2887         {
2888             \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2889             { \l__zrefclever_setup_language_tl } {#1}
2890         }
2891         {
2892             \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2893             { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2894             {
2895                 \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2896                 {#1} { \l__zrefclever_setup_language_tl }
2897                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
2898                 \l__zrefclever_lang_decl_case_tl
2899             }
2900         }
2901     } ,
2902     case .value_required:n = true ,
2903
2904     gender .value_required:n = true ,
2905     gender .code:n =
2906     {
2907         \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2908         {
2909             \msg_warning:nxxxx { zref-clever } { language-no-gender }
2910             { \l__zrefclever_setup_language_tl } { gender } {#1}
2911         }

```

```

2912     {
2913         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2914         {
2915             \msg_warning:nnn { zref-clever }
2916             { option-only-type-specific } { gender }
2917         }
2918         {
2919             \seq_clear:N \l_tmpa_seq
2920             \clist_map_inline:nn {#1}
2921             {
2922                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2923                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
2924                 {
2925                     \msg_warning:nnxx { zref-clever }
2926                     { gender-not-declared }
2927                     { \l__zrefclever_setup_language_tl } {##1}
2928                 }
2929             }
2930             \__zrefclever_opt_seq_gset_eq:cn
2931             {
2932                 \__zrefclever_opt_varname_lang_type:enn
2933                 { \l__zrefclever_setup_language_tl }
2934                 { \l__zrefclever_setup_type_tl }
2935                 { gender }
2936                 { seq }
2937             }
2938             \l_tmpa_seq
2939         }
2940     } ,
2941 } ,
2942 }
2943 \seq_map_inline:Nn
2944 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2945 {
2946     \keys_define:nn { zref-clever/langsetup }
2947     {
2948         #1 .value_required:n = true ,
2949         #1 .code:n =
2950         {
2951             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2952             {
2953                 \__zrefclever_opt_tl_gset:cn
2954                 {
2955                     \__zrefclever_opt_varname_lang_default:enn
2956                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2957                 }
2958                 {##1}
2959             }
2960             {
2961                 \msg_warning:nnn { zref-clever }
2962                 { option-not-type-specific } {#1}
2963             }
2964         } ,
2965     }

```

```

2966 }
2967 \seq_map_inline:Nn
2968 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2969 {
2970   \keys_define:nn { zref-clever/langsetup }
2971   {
2972     #1 .value_required:n = true ,
2973     #1 .code:n =
2974     {
2975       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2976       {
2977         \__zrefclever_opt_tl_gset:cn
2978         {
2979           \__zrefclever_opt_varname_lang_default:enn
2980           { \l__zrefclever_setup_language_tl } {#1} { t1 }
2981         }
2982         {##1}
2983       }
2984       {
2985         \__zrefclever_opt_tl_gset:cn
2986         {
2987           \__zrefclever_opt_varname_lang_type:enn
2988           { \l__zrefclever_setup_language_tl }
2989           { \l__zrefclever_setup_type_tl }
2990           {#1} { t1 }
2991         }
2992         {##1}
2993       }
2994     } ,
2995   }
2996 }
2997 \keys_define:nn { zref-clever/langsetup }
2998 {
2999   endrange .value_required:n = true ,
3000   endrange .code:n =
3001   {
3002     \str_case:nnF {#1}
3003     {
3004       { ref }
3005       {
3006         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3007         {
3008           \__zrefclever_opt_tl_gclear:c
3009           {
3010             \__zrefclever_opt_varname_lang_default:enn
3011             { \l__zrefclever_setup_language_tl }
3012             { endrangefunc } { t1 }
3013           }
3014           \__zrefclever_opt_tl_gclear:c
3015           {
3016             \__zrefclever_opt_varname_lang_default:enn
3017             { \l__zrefclever_setup_language_tl }
3018             { endrangeprop } { t1 }
3019           }

```

```

3020     }
3021     {
3022         \__zrefclever_opt_tl_gclear:c
3023         {
3024             \__zrefclever_opt_varname_lang_type:enn
3025             { \l__zrefclever_setup_language_tl }
3026             { \l__zrefclever_setup_type_tl }
3027             { endrangefunc } { tl }
3028         }
3029         \__zrefclever_opt_tl_gclear:c
3030         {
3031             \__zrefclever_opt_varname_lang_type:enn
3032             { \l__zrefclever_setup_language_tl }
3033             { \l__zrefclever_setup_type_tl }
3034             { endrangeprop } { tl }
3035         }
3036     }
3037 }
3038
3039 { stripprefix }
3040 {
3041     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3042     {
3043         \__zrefclever_opt_tl_gset:cn
3044         {
3045             \__zrefclever_opt_varname_lang_default:enn
3046             { \l__zrefclever_setup_language_tl }
3047             { endrangefunc } { tl }
3048         }
3049         { __zrefclever_get_endrange_stripprefix }
3050         \__zrefclever_opt_tl_gclear:c
3051         {
3052             \__zrefclever_opt_varname_lang_default:enn
3053             { \l__zrefclever_setup_language_tl }
3054             { endrangeprop } { tl }
3055         }
3056     }
3057     {
3058         \__zrefclever_opt_tl_gset:cn
3059         {
3060             \__zrefclever_opt_varname_lang_type:enn
3061             { \l__zrefclever_setup_language_tl }
3062             { \l__zrefclever_setup_type_tl }
3063             { endrangefunc } { tl }
3064         }
3065         { __zrefclever_get_endrange_stripprefix }
3066         \__zrefclever_opt_tl_gclear:c
3067         {
3068             \__zrefclever_opt_varname_lang_type:enn
3069             { \l__zrefclever_setup_language_tl }
3070             { \l__zrefclever_setup_type_tl }
3071             { endrangeprop } { tl }
3072         }
3073     }

```

```

3074 }
3075
3076 { pagecomp }
3077 {
3078   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3079   {
3080     \__zrefclever_opt_tl_gset:cn
3081     {
3082       \__zrefclever_opt_varname_lang_default:enn
3083       { \l__zrefclever_setup_language_tl }
3084       { endrangefunc } { tl }
3085     }
3086     { __zrefclever_get_endrange_pagecomp }
3087     \__zrefclever_opt_tl_gclear:c
3088     {
3089       \__zrefclever_opt_varname_lang_default:enn
3090       { \l__zrefclever_setup_language_tl }
3091       { endrangeprop } { tl }
3092     }
3093   }
3094   {
3095     \__zrefclever_opt_tl_gset:cn
3096     {
3097       \__zrefclever_opt_varname_lang_type:enn
3098       { \l__zrefclever_setup_language_tl }
3099       { \l__zrefclever_setup_type_tl }
3100       { endrangefunc } { tl }
3101     }
3102     { __zrefclever_get_endrange_pagecomp }
3103     \__zrefclever_opt_tl_gclear:c
3104     {
3105       \__zrefclever_opt_varname_lang_type:enn
3106       { \l__zrefclever_setup_language_tl }
3107       { \l__zrefclever_setup_type_tl }
3108       { endrangeprop } { tl }
3109     }
3110   }
3111 }
3112
3113 { pagecomp2 }
3114 {
3115   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3116   {
3117     \__zrefclever_opt_tl_gset:cn
3118     {
3119       \__zrefclever_opt_varname_lang_default:enn
3120       { \l__zrefclever_setup_language_tl }
3121       { endrangefunc } { tl }
3122     }
3123     { __zrefclever_get_endrange_pagecomptwo }
3124     \__zrefclever_opt_tl_gclear:c
3125     {
3126       \__zrefclever_opt_varname_lang_default:enn
3127       { \l__zrefclever_setup_language_tl }

```

```

3128         { endrangeprop } { t1 }
3129     }
3130 }
3131 {
3132     \__zrefclever_opt_tl_gset:cn
3133     {
3134         \__zrefclever_opt_varname_lang_type:enn
3135         { \l__zrefclever_setup_language_tl }
3136         { \l__zrefclever_setup_type_tl }
3137         { endrangefunc } { t1 }
3138     }
3139     { __zrefclever_get_endrange_pagecomptwo }
3140     \__zrefclever_opt_tl_gclear:c
3141     {
3142         \__zrefclever_opt_varname_lang_type:enn
3143         { \l__zrefclever_setup_language_tl }
3144         { \l__zrefclever_setup_type_tl }
3145         { endrangeprop } { t1 }
3146     }
3147 }
3148 }
3149 }
3150 {
3151     \tl_if_empty:nTF {#1}
3152     {
3153         \msg_warning:nnn { zref-clever }
3154         { endrange-property-undefined } {#1}
3155     }
3156     {
3157         \zref@ifpropundefined {#1}
3158         {
3159             \msg_warning:nnn { zref-clever }
3160             { endrange-property-undefined } {#1}
3161         }
3162         {
3163             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3164             {
3165                 \__zrefclever_opt_tl_gset:cn
3166                 {
3167                     \__zrefclever_opt_varname_lang_default:enn
3168                     { \l__zrefclever_setup_language_tl }
3169                     { endrangefunc } { t1 }
3170                 }
3171                 { __zrefclever_get_endrange_property }
3172                 \__zrefclever_opt_tl_gset:cn
3173                 {
3174                     \__zrefclever_opt_varname_lang_default:enn
3175                     { \l__zrefclever_setup_language_tl }
3176                     { endrangeprop } { t1 }
3177                 }
3178                 {#1}
3179             }
3180             {
3181                 \__zrefclever_opt_tl_gset:cn

```

```

3182         {
3183             \_zrefclever_opt_varname_lang_type:eenn
3184             { \l__zrefclever_setup_language_tl }
3185             { \l__zrefclever_setup_type_tl }
3186             { endrangefunc } { tl }
3187         }
3188         { __zrefclever_get_endrange_property }
3189         \_zrefclever_opt_tl_gset:cn
3190         {
3191             \_zrefclever_opt_varname_lang_type:eenn
3192             { \l__zrefclever_setup_language_tl }
3193             { \l__zrefclever_setup_type_tl }
3194             { endrangeprop } { tl }
3195         }
3196         {#1}
3197     }
3198 }
3199 }
3200 }
3201 },
3202 }
3203 \keys_define:nn { zref-clever/langsetup }
3204 {
3205     refpre .code:n =
3206     {
3207         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3208         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3209         { refpre } { refbounds }
3210     } ,
3211     refpos .code:n =
3212     {
3213         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3214         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3215         { refpos } { refbounds }
3216     } ,
3217     preref .code:n =
3218     {
3219         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3220         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3221         { preref } { refbounds }
3222     } ,
3223     postref .code:n =
3224     {
3225         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3226         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3227         { postref } { refbounds }
3228     } ,
3229 }
3230 \seq_map_inline:Nn
3231   \g__zrefclever_rf_opts_tl_type_names_seq
3232   {
3233     \keys_define:nn { zref-clever/langsetup }
3234     {
3235         #1 .value_required:n = true ,

```

```

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

```



```

3290         \__zrefclever_opt_seq_gset_eq:cN
3291         {
3292             \__zrefclever_opt_varname_lang_default:enn
3293             { \l__zrefclever_setup_language_tl }
3294             {#1} { seq }
3295         }
3296         \g_tmpa_seq
3297     }
3298     {
3299         \msg_warning:nxxx { zref-clever }
3300         { rebounds-must-be-four }
3301         {#1} { \seq_count:N \g_tmpa_seq }
3302     }
3303 }
3304 {
3305     \seq_gclear:N \g_tmpa_seq
3306     \__zrefclever_opt_seq_gset_clist_split:Nn
3307     \g_tmpa_seq {##1}
3308     \bool_lazy_or:nnTF
3309     { \tl_if_empty_p:n {##1} }
3310     {
3311         \int_compare_p:nNn
3312         { \seq_count:N \g_tmpa_seq } = { 4 }
3313     }
3314     {
3315         \__zrefclever_opt_seq_gset_eq:cN
3316         {
3317             \__zrefclever_opt_varname_lang_type:enn
3318             { \l__zrefclever_setup_language_tl }
3319             { \l__zrefclever_setup_type_tl } {#1} { seq }
3320         }
3321         \g_tmpa_seq
3322     }
3323     {
3324         \msg_warning:nxxx { zref-clever }
3325         { rebounds-must-be-four }
3326         {#1} { \seq_count:N \g_tmpa_seq }
3327     }
3328 }
3329 } ,
3330 }
3331 }
3332 \seq_map_inline:Nn
3333 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3334 {
3335     \keys_define:nn { zref-clever/langsetup }
3336     {
3337         #1 .choice: ,
3338         #1 / true .code:n =
3339         {
3340             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3341             {
3342                 \__zrefclever_opt_bool_gset_true:c
3343                 {

```

```

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

```

## 6 User interface

### 6.1 \zcref

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

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

```

3385 \NewDocumentCommand \zcref { s O { } m }
3386 { \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}
```

```
3387 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3388 {
3389   \group_begin:
```

Set options.

```
3390   \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```
3391   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3392   \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.

```
3393   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3394   \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3395   \bool_lazy_and:nnT
3396   { \l__zrefclever_zrefcheck_available_bool }
3397   { \l__zrefclever_zcref_with_check_bool }
3398   { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3399   \bool_lazy_or:nnT
3400   { \l__zrefclever_typeset_sort_bool }
3401   { \l__zrefclever_typeset_range_bool }
3402   { \__zrefclever_sort_labels: }
```

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

```
3403   \group_begin:
3404   \l__zrefclever_ref_typeset_font_tl
3405   \__zrefclever_typeset_refs:
3406   \group_end:
```

Typeset note.

```
3407   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3408   {
3409     \__zrefclever_get_rf_opt_tl:nxxN { noteseq }
3410     { \l__zrefclever_label_type_a_tl }
3411     { \l__zrefclever_ref_language_tl }
3412     \l_tmpa_tl
3413     \l_tmpa_tl
3414     \l__zrefclever_zcref_note_tl
3415   }
```

Integration with zref-check.

```

3416 \bool_lazy_and:nnT
3417 { \l__zrefclever_zrefcheck_available_bool }
3418 { \l__zrefclever_zcref_with_check_bool }
3419 {
3420 \zrefcheck_zcref_end_label_maybe:
3421 \zrefcheck_zcref_run_checks_on_labels:n
3422 { \l__zrefclever_zcref_labels_seq }
3423 }

```

Integration with mathtools.

```

3424 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3425 {
3426 \__zrefclever_mathtools_showonlyrefs:n
3427 { \l__zrefclever_zcref_labels_seq }
3428 }
3429 \group_end:
3430 }

```

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

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3431 \seq_new:N \l__zrefclever_zcref_labels_seq
3432 \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{\*}[\langle options \rangle]{\langle labels \rangle}

```

3433 \NewDocumentCommand \zcpageref { s O { } m }
3434 {
3435 \group_begin:
3436 \IfBooleanT {#1}
3437 { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3438 \zcref [#2, ref = page] {#3}
3439 \group_end:
3440 }

```

(End definition for \zcpageref.)

## 7 Sorting

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

single reference type. Because of this, sorting must take into account the whole chain of “enclosing counters” for the counters of the labels at hand.

```

\l_zrefclever_label_type_a_tl Auxiliary variables, for use in sorting, and some also in typesetting. Used to store refer-
\l_zrefclever_label_type_b_tl ence information – label properties – of the “current” (a) and “next” (b) labels.
\l_zrefclever_label_enclval_a_tl 3441 \tl_new:N \l__zrefclever_label_type_a_tl
\l_zrefclever_label_enclval_b_tl 3442 \tl_new:N \l__zrefclever_label_type_b_tl
\l_zrefclever_label_extdoc_a_tl 3443 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l_zrefclever_label_extdoc_b_tl 3444 \tl_new:N \l__zrefclever_label_enclval_b_tl
3445 \tl_new:N \l__zrefclever_label_extdoc_a_tl
3446 \tl_new:N \l__zrefclever_label_extdoc_b_tl

```

*(End definition for \l\_\_zrefclever\_label\_type\_a\_tl and others.)*

```

\l_zrefclever_sort_decided_bool Auxiliary variable for \__zrefclever_sort_default_same_type:nn, signals if the sort-
ing between two labels has been decided or not.
3447 \bool_new:N \l__zrefclever_sort_decided_bool

```

*(End definition for \l\_\_zrefclever\_sort\_decided\_bool.)*

```

\l_zrefclever_sort_prior_a_int Auxiliary variables for \__zrefclever_sort_default_different_types:nn. Store the
\l_zrefclever_sort_prior_b_int sort priority of the “current” and “next” labels.
3448 \int_new:N \l__zrefclever_sort_prior_a_int
3449 \int_new:N \l__zrefclever_sort_prior_b_int

```

*(End definition for \l\_\_zrefclever\_sort\_prior\_a\_int and \l\_\_zrefclever\_sort\_prior\_b\_int.)*

```

\l_zrefclever_label_types_seq Stores the order in which reference types appear in the label list supplied by the user in
\zcref. This variable is populated by \__zrefclever_label_type_put_new_right:n
at the start of \__zrefclever_sort_labels:. This order is required as a “last resort”
sort criterion between the reference types, for use in \__zrefclever_sort_default_-
different_types:nn.
3450 \seq_new:N \l__zrefclever_label_types_seq

```

*(End definition for \l\_\_zrefclever\_label\_types\_seq.)*

```

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

```

```

3451 \cs_new_protected:Npn \__zrefclever_sort_labels:
3452 {

```

Store label types sequence.

```

3453 \seq_clear:N \l__zrefclever_label_types_seq
3454 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
3455 {
3456 \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3457 \__zrefclever_label_type_put_new_right:n
3458 }

```

Sort.

```

3459 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3460 {
3461   \zref@ifrefundefined {##1}
3462   {
3463     \zref@ifrefundefined {##2}
3464     {
3465       % Neither label is defined.
3466       \sort_return_same:
3467     }
3468     {
3469       % The second label is defined, but the first isn't, leave the
3470       % undefined first (to be more visible).
3471       \sort_return_same:
3472     }
3473   }
3474   {
3475     \zref@ifrefundefined {##2}
3476     {
3477       % The first label is defined, but the second isn't, bring the
3478       % second forward.
3479       \sort_return_swapped:
3480     }
3481     {
3482       % The interesting case: both labels are defined. References
3483       % to the "default" property or to the "page" are quite
3484       % different with regard to sorting, so we branch them here to
3485       % specialized functions.
3486       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3487       { \__zrefclever_sort_page:nn {##1} {##2} }
3488       { \__zrefclever_sort_default:nn {##1} {##2} }
3489     }
3490   }
3491 }
3492 }
```

(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>}

3493 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3494 {
3495   \__zrefclever_extract_default:Nnnn
3496   \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3497   \seq_if_in:NVF \l__zrefclever_label_types_seq
```

```

3498     \l__zrefclever_label_type_a_tl
3499     {
3500         \seq_put_right:NV \l__zrefclever_label_types_seq
3501         \l__zrefclever_label_type_a_tl
3502     }
3503 }

```

(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}}

3504 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3505 {
3506     \__zrefclever_extract_default:Nnnn
3507     \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3508     \__zrefclever_extract_default:Nnnn
3509     \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3510
3511     \tl_if_eq:NNTF
3512     \l__zrefclever_label_type_a_tl
3513     \l__zrefclever_label_type_b_tl
3514     { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3515     { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3516 }

```

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

```

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

3517 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3518 {
3519     \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3520     {#1} {zc@enclval} { }
3521     \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3522     \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3523     {#2} {zc@enclval} { }
3524     \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3525     \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3526     {#1} {externaldocument} { }
3527     \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3528     {#2} {externaldocument} { }
3529
3530     \bool_set_false:N \l__zrefclever_sort_decided_bool
3531
3532     % First we check if there's any "external document" difference (coming
3533     % from 'zref-xr') and, if so, sort based on that.
3534     \tl_if_eq:NNTF
3535     \l__zrefclever_label_extdoc_a_tl
3536     \l__zrefclever_label_extdoc_b_tl
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_same:
3546 }
3547 {
3548   \bool_if:nTF
3549   {
3550     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3551     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3552   }
3553   {
3554     \bool_set_true:N \l__zrefclever_sort_decided_bool
3555     \sort_return_swapped:
3556   }
3557   {
3558     \bool_set_true:N \l__zrefclever_sort_decided_bool
3559     % Two different "external documents": last resort, sort by the
3560     % document name itself.
3561     \str_compare:eNeTF
3562     { \l__zrefclever_label_extdoc_b_tl } <
3563     { \l__zrefclever_label_extdoc_a_tl }
3564     { \sort_return_swapped: }
3565     { \sort_return_same: }
3566   }
3567 }
3568 }
3569
3570 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3571 {
3572   \bool_if:nTF
3573   {
3574     % Both are empty: neither label has any (further) "enclosing
3575     % counters" (left).
3576     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3577     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3578   }
3579   {
3580     \bool_set_true:N \l__zrefclever_sort_decided_bool
3581     \int_compare:nNnTF
3582     { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
3583     >
3584     { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
3585     { \sort_return_swapped: }
3586     { \sort_return_same: }
3587   }
3588   {
3589     \bool_if:nTF
3590     {
3591       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.

```



```

3592     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3593   }
3594   {
3595     \bool_set_true:N \l__zrefclever_sort_decided_bool
3596     \int_compare:nNnTF
3597       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3598       >
3599       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3600       { \sort_return_swapped: }
3601       { \sort_return_same: }
3602   }
3603   {
3604     \bool_if:nTF
3605     {
3606       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3607       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3608     }
3609     {
3610       \bool_set_true:N \l__zrefclever_sort_decided_bool
3611       \int_compare:nNnTF
3612         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3613         <
3614         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3615         { \sort_return_same: }
3616         { \sort_return_swapped: }
3617     }
3618     {
3619       % Neither is empty: we can compare the values of the
3620       % current enclosing counter in the loop, if they are
3621       % equal, we are still in the loop, if they are not, a
3622       % sorting decision can be made directly.
3623       \int_compare:nNnTF
3624         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3625         =
3626         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3627         {
3628           \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3629             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3630           \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3631             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3632         }
3633         {
3634           \bool_set_true:N \l__zrefclever_sort_decided_bool
3635           \int_compare:nNnTF
3636             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3637             >
3638             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3639             { \sort_return_swapped: }
3640             { \sort_return_same: }
3641         }
3642     }
3643   }
3644 }
3645 }

```

```
3646 }
```

(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}
```

```
3647 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
```

```
3648 {
```

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”.

```
3649 \int_zero:N \l__zrefclever_sort_prior_a_int
```

```
3650 \int_zero:N \l__zrefclever_sort_prior_b_int
```

```
3651 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
```

```
3652 {
```

```
3653 \tl_if_eq:nnTF {##2} {{othertypes}}
```

```
3654 {
```

```
3655 \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
```

```
3656 { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
```

```
3657 \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
```

```
3658 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
```

```
3659 }
```

```
3660 {
```

```
3661 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
```

```
3662 { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
```

```
3663 {
```

```
3664 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
```

```
3665 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
```

```
3666 }
```

```
3667 }
```

```
3668 }
```

Then do the actual sorting.

```
3669 \bool_if:nTF
```

```
3670 {
```

```
3671 \int_compare_p:nNn
```

```
3672 { \l__zrefclever_sort_prior_a_int } <
```

```
3673 { \l__zrefclever_sort_prior_b_int }
```

```
3674 }
```

```
3675 { \sort_return_same: }
```

```
3676 {
```

```
3677 \bool_if:nTF
```

```
3678 {
```

```
3679 \int_compare_p:nNn
```

```
3680 { \l__zrefclever_sort_prior_a_int } >
```

```
3681 { \l__zrefclever_sort_prior_b_int }
```

```
3682 }
```

```
3683 { \sort_return_swapped: }
```

```
3684 {
```

```
3685 % Sort priorities are equal: the type that occurs first in
```

```
3686 % ‘labels’, as given by the user, is kept (or brought) forward.
```

```
3687 \seq_map_inline:Nn \l__zrefclever_label_types_seq
```

```
3688 {
```

```
3689 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
```

```

3690         { \seq_map_break:n { \sort_return_same: } }
3691         {
3692             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3693             { \seq_map_break:n { \sort_return_swapped: } }
3694         }
3695     }
3696 }
3697 }
3698 }

```

(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}

3699 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3700 {
3701     \int_compare:nNnTF
3702     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3703     >
3704     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3705     { \sort_return_swapped: }
3706     { \sort_return_same: }
3707 }

```

(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
```

```
3708 \seq_new:N \l__zrefclever_typeset_labels_seq
3709 \bool_new:N \l__zrefclever_typeset_last_bool
3710 \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
```

```
3711 \int_new:N \l__zrefclever_type_count_int
3712 \int_new:N \l__zrefclever_label_count_int
3713 \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
```

```
3714 \tl_new:N \l__zrefclever_label_a_tl
3715 \tl_new:N \l__zrefclever_label_b_tl
3716 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
3717 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
3718 \tl_new:N \l__zrefclever_type_first_label_tl
3719 \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
```

```
3720 \tl_new:N \l__zrefclever_type_name_tl
3721 \bool_new:N \l_zrefclever_name_in_link_bool
3722 \bool_new:N \l__zrefclever_type_name_missing_bool
3723 \tl_new:N \l__zrefclever_name_format_tl
3724 \tl_new:N \l__zrefclever_name_format_fallback_tl
3725 \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
```

```
3726 \int_new:N \l__zrefclever_range_count_int
3727 \int_new:N \l__zrefclever_range_same_count_int
3728 \tl_new:N \l__zrefclever_range_beg_label_tl
3729 \bool_new:N \l__zrefclever_range_beg_is_first_bool
3730 \tl_new:N \l__zrefclever_range_end_ref_tl
3731 \bool_new:N \l__zrefclever_next_maybe_range_bool
3732 \bool_new:N \l__zrefclever_next_is_same_bool
```

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

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

```

\l__zrefclever_tpairsep_tl 3733 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl 3734 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl 3735 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl 3736 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl 3737 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl 3738 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl 3739 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_rangeseq_tl 3740 \tl_new:N \l__zrefclever_rangeseq_tl
\l__zrefclever_namefont_tl 3741 \tl_new:N \l__zrefclever_namefont_tl
\l__zrefclever_reffont_tl 3742 \tl_new:N \l__zrefclever_reffont_tl
\l__zrefclever_endrangefunc_tl 3743 \tl_new:N \l__zrefclever_endrangefunc_tl
\l__zrefclever_endrangeprop_tl 3744 \tl_new:N \l__zrefclever_endrangeprop_tl
\l__zrefclever_cap_bool 3745 \bool_new:N \l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool 3746 \bool_new:N \l__zrefclever_abbrev_bool
\l__zrefclever_rangetopair_bool 3747 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

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

```

\l__zrefclever_refbounds_first_seq 3748 \seq_new:N \l__zrefclever_refbounds_first_seq
\l__zrefclever_refbounds_first_sg_seq 3749 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
\l__zrefclever_refbounds_first_pb_seq 3750 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
\l__zrefclever_refbounds_first_rb_seq 3751 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
\l__zrefclever_refbounds_mid_seq 3752 \seq_new:N \l__zrefclever_refbounds_mid_seq
\l__zrefclever_refbounds_mid_rb_seq 3753 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
\l__zrefclever_refbounds_mid_re_seq 3754 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
\l__zrefclever_refbounds_last_seq 3755 \seq_new:N \l__zrefclever_refbounds_last_seq
\l__zrefclever_refbounds_last_pe_seq 3756 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_refbounds_last_re_seq 3757 \seq_new:N \l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_seq 3758 \seq_new:N \l__zrefclever_type_first_refbounds_seq
\l__zrefclever_type_first_refbounds_set_bool 3759 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

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

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`.

```
3760 \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`.

```

3761 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3762 {
3763   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3764   \l__zrefclever_zceref_labels_seq
3765   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3766   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3767   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

3768 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3769 \tl_clear:N \l__zrefclever_range_beg_label_tl
3770 \tl_clear:N \l__zrefclever_range_end_ref_tl
3771 \int_zero:N \l__zrefclever_label_count_int
3772 \int_zero:N \l__zrefclever_type_count_int
3773 \int_zero:N \l__zrefclever_ref_count_int
3774 \int_zero:N \l__zrefclever_range_count_int
3775 \int_zero:N \l__zrefclever_range_same_count_int
3776 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3777 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3778
3779 % Get type block options (not type-specific).
3780 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3781   { \l__zrefclever_label_type_a_tl }
3782   { \l__zrefclever_ref_language_tl }
3783   \l__zrefclever_tpairsep_tl
3784 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3785   { \l__zrefclever_label_type_a_tl }
3786   { \l__zrefclever_ref_language_tl }
3787   \l__zrefclever_tlistsep_tl
3788 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3789   { \l__zrefclever_label_type_a_tl }
3790   { \l__zrefclever_ref_language_tl }
3791   \l__zrefclever_tlastsep_tl
3792
3793 % Process label stack.
3794 \bool_set_false:N \l__zrefclever_typeset_last_bool
3795 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3796   {
3797     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3798     \l__zrefclever_label_a_tl
3799     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3800       {
3801         \tl_clear:N \l__zrefclever_label_b_tl
3802         \bool_set_true:N \l__zrefclever_typeset_last_bool
3803       }
3804       {
3805         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3806         \l__zrefclever_label_b_tl
3807       }
3808
3809     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3810       {
3811         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3812         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3813       }
3814       {
3815         \__zrefclever_extract_default:NVnn
3816         \l__zrefclever_label_type_a_tl
3817         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3818         \__zrefclever_extract_default:NVnn
3819         \l__zrefclever_label_type_b_tl
3820         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3821       }

```

```

3822
3823 % First, we establish whether the "current label" (i.e. 'a') is the
3824 % last one of its type. This can happen because the "next label"
3825 % (i.e. 'b') is of a different type (or different definition status),
3826 % or because we are at the end of the list.
3827 \bool_if:NTF \l__zrefclever_typeset_last_bool
3828 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3829 {
3830   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3831   {
3832     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3833     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3834     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3835   }
3836   {
3837     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3838     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3839     {
3840       % Neither is undefined, we must check the types.
3841       \tl_if_eq:NNTF
3842         \l__zrefclever_label_type_a_tl
3843         \l__zrefclever_label_type_b_tl
3844         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3845         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3846     }
3847   }
3848 }
3849
3850 % Handle warnings in case of reference or type undefined.
3851 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3852 \zref@refused { \l__zrefclever_label_a_tl }
3853 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3854 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3855 {}
3856 {
3857   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3858   {
3859     \msg_warning:nxx { zref-clever } { missing-type }
3860     { \l__zrefclever_label_a_tl }
3861   }
3862   \zref@ifrefcontainsprop
3863     { \l__zrefclever_label_a_tl }
3864     { \l__zrefclever_ref_property_tl }
3865     { }
3866     {
3867       \msg_warning:nxxx { zref-clever } { missing-property }
3868       { \l__zrefclever_ref_property_tl }
3869       { \l__zrefclever_label_a_tl }
3870     }
3871 }
3872
3873 % Get possibly type-specific separators, refbounds, font and other
3874 % options, once per type.
3875 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```



```

3876 {
3877   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3878   { \l__zrefclever_label_type_a_tl }
3879   { \l__zrefclever_ref_language_tl }
3880   \l__zrefclever_namesep_tl
3881   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3882   { \l__zrefclever_label_type_a_tl }
3883   { \l__zrefclever_ref_language_tl }
3884   \l__zrefclever_pairsep_tl
3885   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3886   { \l__zrefclever_label_type_a_tl }
3887   { \l__zrefclever_ref_language_tl }
3888   \l__zrefclever_listsep_tl
3889   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3890   { \l__zrefclever_label_type_a_tl }
3891   { \l__zrefclever_ref_language_tl }
3892   \l__zrefclever_lastsep_tl
3893   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3894   { \l__zrefclever_label_type_a_tl }
3895   { \l__zrefclever_ref_language_tl }
3896   \l__zrefclever_rangesep_tl
3897   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3898   { \l__zrefclever_label_type_a_tl }
3899   { \l__zrefclever_ref_language_tl }
3900   \l__zrefclever_namefont_tl
3901   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3902   { \l__zrefclever_label_type_a_tl }
3903   { \l__zrefclever_ref_language_tl }
3904   \l__zrefclever_reffont_tl
3905   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3906   { \l__zrefclever_label_type_a_tl }
3907   { \l__zrefclever_ref_language_tl }
3908   \l__zrefclever_endrangefunc_tl
3909   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3910   { \l__zrefclever_label_type_a_tl }
3911   { \l__zrefclever_ref_language_tl }
3912   \l__zrefclever_endrangeprop_tl
3913   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3914   { \l__zrefclever_label_type_a_tl }
3915   { \l__zrefclever_ref_language_tl }
3916   \l__zrefclever_cap_bool
3917   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3918   { \l__zrefclever_label_type_a_tl }
3919   { \l__zrefclever_ref_language_tl }
3920   \l__zrefclever_abbrev_bool
3921   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3922   { \l__zrefclever_label_type_a_tl }
3923   { \l__zrefclever_ref_language_tl }
3924   \l__zrefclever_rangetopair_bool
3925   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3926   { \l__zrefclever_label_type_a_tl }
3927   { \l__zrefclever_ref_language_tl }
3928   \l__zrefclever_refbounds_first_seq
3929   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3930         { \l__zrefclever_label_type_a_tl }
3931         { \l__zrefclever_ref_language_tl }
3932         \l__zrefclever_refbounds_first_sg_seq
3933     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3934         { \l__zrefclever_label_type_a_tl }
3935         { \l__zrefclever_ref_language_tl }
3936         \l__zrefclever_refbounds_first_pb_seq
3937     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3938         { \l__zrefclever_label_type_a_tl }
3939         { \l__zrefclever_ref_language_tl }
3940         \l__zrefclever_refbounds_first_rb_seq
3941     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3942         { \l__zrefclever_label_type_a_tl }
3943         { \l__zrefclever_ref_language_tl }
3944         \l__zrefclever_refbounds_mid_seq
3945     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3946         { \l__zrefclever_label_type_a_tl }
3947         { \l__zrefclever_ref_language_tl }
3948         \l__zrefclever_refbounds_mid_rb_seq
3949     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3950         { \l__zrefclever_label_type_a_tl }
3951         { \l__zrefclever_ref_language_tl }
3952         \l__zrefclever_refbounds_mid_re_seq
3953     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3954         { \l__zrefclever_label_type_a_tl }
3955         { \l__zrefclever_ref_language_tl }
3956         \l__zrefclever_refbounds_last_seq
3957     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3958         { \l__zrefclever_label_type_a_tl }
3959         { \l__zrefclever_ref_language_tl }
3960         \l__zrefclever_refbounds_last_pe_seq
3961     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3962         { \l__zrefclever_label_type_a_tl }
3963         { \l__zrefclever_ref_language_tl }
3964         \l__zrefclever_refbounds_last_re_seq
3965     }
3966
3967     % Here we send this to a couple of auxiliary functions.
3968     \bool_if:NTF \l__zrefclever_last_of_type_bool
3969     {
3970         % There exists no next label of the same type as the current.
3971         { \__zrefclever_typeset_refs_last_of_type: }
3972         % There exists a next label of the same type as the current.
3973         { \__zrefclever_typeset_refs_not_last_of_type: }
3974     }

```

(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.

```

3975 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3976 {
3977   % Process the current label to the current queue.
3978   \int_case:nnF { \l__zrefclever_label_count_int }
3979   {
3980     % It is the last label of its type, but also the first one, and that's
3981     % what matters here: just store it.
3982     % Test: 'zc-typeset01.lvt': "Last of type: single"
3983     { 0 }
3984     {
3985       \tl_set:NV \l__zrefclever_type_first_label_tl
3986       \l__zrefclever_label_a_tl
3987       \tl_set:NV \l__zrefclever_type_first_label_type_tl
3988       \l__zrefclever_label_type_a_tl
3989       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3990       \l__zrefclever_refbounds_first_sg_seq
3991       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3992     }
3993
3994     % The last is the second: we have a pair (if not repeated).
3995     % Test: 'zc-typeset01.lvt': "Last of type: pair"
3996     { 1 }
3997     {
3998       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
3999       {
4000         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4001         \l__zrefclever_refbounds_first_sg_seq
4002         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4003       }
4004       {
4005         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4006         {
4007           \exp_not:V \l__zrefclever_pairsep_tl
4008           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4009           \l__zrefclever_refbounds_last_pe_seq
4010         }
4011         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4012         \l__zrefclever_refbounds_first_pb_seq
4013         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4014       }
4015     }
4016   }
4017   % Last is third or more of its type: without repetition, we'd have the
4018   % last element on a list, but control for possible repetition.
4019   {
4020     \int_case:nnF { \l__zrefclever_range_count_int }
4021     {
4022       % There was no range going on.
4023       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4024       { 0 }

```

```

4025 {
4026   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4027   {
4028     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4029     {
4030       \exp_not:V \l__zrefclever_pairsep_tl
4031       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4032       \l__zrefclever_refbounds_last_pe_seq
4033     }
4034   }
4035   {
4036     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4037     {
4038       \exp_not:V \l__zrefclever_lastsep_tl
4039       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4040       \l__zrefclever_refbounds_last_seq
4041     }
4042   }
4043 }
4044 % Last in the range is also the second in it.
4045 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4046 { 1 }
4047 {
4048   \int_compare:nNnTF
4049   { \l__zrefclever_range_same_count_int } = { 1 }
4050   {
4051     % We know 'range_beg_is_first_bool' is false, since this is
4052     % the second element in the range, but the third or more in
4053     % the type list.
4054     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4055     {
4056       \exp_not:V \l__zrefclever_pairsep_tl
4057       \__zrefclever_get_ref:VN
4058       \l__zrefclever_range_beg_label_tl
4059       \l__zrefclever_refbounds_last_pe_seq
4060     }
4061     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4062     \l__zrefclever_refbounds_first_pb_seq
4063     \bool_set_true:N
4064     \l__zrefclever_type_first_refbounds_set_bool
4065   }
4066   {
4067     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4068     {
4069       \exp_not:V \l__zrefclever_listsep_tl
4070       \__zrefclever_get_ref:VN
4071       \l__zrefclever_range_beg_label_tl
4072       \l__zrefclever_refbounds_mid_seq
4073       \exp_not:V \l__zrefclever_lastsep_tl
4074       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4075       \l__zrefclever_refbounds_last_seq
4076     }
4077   }
4078 }

```

```

4079 }
4080 % Last in the range is third or more in it.
4081 {
4082   \int_case:nnF
4083   {
4084     \l__zrefclever_range_count_int -
4085     \l__zrefclever_range_same_count_int
4086   }
4087   {
4088     % Repetition, not a range.
4089     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4090     { 0 }
4091     {
4092       % If 'range_beg_is_first_bool' is true, it means it was also
4093       % the first of the type, and hence its typesetting was
4094       % already handled, and we just have to set rebounds.
4095       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4096       {
4097         \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4098         \l__zrefclever_rebounds_first_sg_seq
4099         \bool_set_true:N
4100         \l__zrefclever_type_first_rebounds_set_bool
4101       }
4102       {
4103         \int_compare:nNnTF
4104         { \l__zrefclever_ref_count_int } < { 2 }
4105         {
4106           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4107           {
4108             \exp_not:V \l__zrefclever_pairsep_tl
4109             \__zrefclever_get_ref:VN
4110             \l__zrefclever_range_beg_label_tl
4111             \l__zrefclever_rebounds_last_pe_seq
4112           }
4113         }
4114         {
4115           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4116           {
4117             \exp_not:V \l__zrefclever_lastsep_tl
4118             \__zrefclever_get_ref:VN
4119             \l__zrefclever_range_beg_label_tl
4120             \l__zrefclever_rebounds_last_seq
4121           }
4122         }
4123       }
4124     }
4125     % A 'range', but with no skipped value, treat as pair if range
4126     % started with first of type, otherwise as list.
4127     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4128     { 1 }
4129     {
4130       % Ditto.
4131       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4132       {

```

```

4133         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4134         \l__zrefclever_refbounds_first_pb_seq
4135     \bool_set_true:N
4136         \l__zrefclever_type_first_refbounds_set_bool
4137     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4138     {
4139         \exp_not:V \l__zrefclever_pairsep_tl
4140         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4141         \l__zrefclever_refbounds_last_pe_seq
4142     }
4143 }
4144 {
4145     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4146     {
4147         \exp_not:V \l__zrefclever_listsep_tl
4148         \__zrefclever_get_ref:VN
4149         \l__zrefclever_range_beg_label_tl
4150         \l__zrefclever_refbounds_mid_seq
4151     }
4152     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4153     {
4154         \exp_not:V \l__zrefclever_lastsep_tl
4155         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4156         \l__zrefclever_refbounds_last_seq
4157     }
4158 }
4159 }
4160 }
4161 {
4162     % An actual range.
4163     % Test: 'zc-typeset01.lvt': "Last of type: range"
4164     % Ditto.
4165     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4166     {
4167         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4168         \l__zrefclever_refbounds_first_rb_seq
4169         \bool_set_true:N
4170         \l__zrefclever_type_first_refbounds_set_bool
4171     }
4172     {
4173         \int_compare:nNnTF
4174         { \l__zrefclever_ref_count_int } < { 2 }
4175         {
4176             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4177             {
4178                 \exp_not:V \l__zrefclever_pairsep_tl
4179                 \__zrefclever_get_ref:VN
4180                 \l__zrefclever_range_beg_label_tl
4181                 \l__zrefclever_refbounds_mid_rb_seq
4182             }
4183             \seq_set_eq:NN
4184             \l__zrefclever_type_first_refbounds_seq
4185             \l__zrefclever_refbounds_first_pb_seq
4186             \bool_set_true:N

```

```

4187         \l__zrefclever_type_first_refbounds_set_bool
4188     }
4189     {
4190         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4191         {
4192             \exp_not:V \l__zrefclever_lastsep_tl
4193             \__zrefclever_get_ref:VN
4194             \l__zrefclever_range_beg_label_tl
4195             \l__zrefclever_refbounds_mid_rb_seq
4196         }
4197     }
4198 }
4199 \bool_lazy_and:nnTF
4200 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4201 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4202 {
4203     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4204     \l__zrefclever_range_beg_label_tl
4205     \l__zrefclever_label_a_tl
4206     \l__zrefclever_range_end_ref_tl
4207     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4208     {
4209         \exp_not:V \l__zrefclever_rangesep_tl
4210         \__zrefclever_get_ref_endrange:VVN
4211         \l__zrefclever_label_a_tl
4212         \l__zrefclever_range_end_ref_tl
4213         \l__zrefclever_refbounds_last_re_seq
4214     }
4215 }
4216 {
4217     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4218     {
4219         \exp_not:V \l__zrefclever_rangesep_tl
4220         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4221         \l__zrefclever_refbounds_last_re_seq
4222     }
4223 }
4224 }
4225 }
4226 }
4227
4228 % Handle "range" option. The idea is simple: if the queue is not empty,
4229 % we replace it with the end of the range (or pair). We can still
4230 % retrieve the end of the range from 'label_a' since we know to be
4231 % processing the last label of its type at this point.
4232 \bool_if:NT \l__zrefclever_typeset_range_bool
4233 {
4234     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4235     {
4236         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4237         { }
4238         {
4239             \msg_warning:nmx { zref-clever } { single-element-range }
4240             { \l__zrefclever_type_first_label_type_tl }

```

```

4241     }
4242   }
4243   {
4244     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4245     \bool_if:NT \l__zrefclever_rangetopair_bool
4246     {
4247       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4248       { }
4249       {
4250         \__zrefclever_labels_in_sequence:nn
4251         { \l__zrefclever_type_first_label_tl }
4252         { \l__zrefclever_label_a_tl }
4253       }
4254     }
4255     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4256     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4257     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4258     {
4259       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4260       {
4261         \exp_not:V \l__zrefclever_pairsep_tl
4262         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4263         \l__zrefclever_refbounds_last_pe_seq
4264       }
4265       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4266       \l__zrefclever_refbounds_first_pb_seq
4267       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4268     }
4269     {
4270       \bool_lazy_and:nnTF
4271       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4272       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4273       {
4274         % We must get 'type_first_label_tl' instead of
4275         % 'range_beg_label_tl' here, since it is not necessary
4276         % that the first of type was actually starting a range for
4277         % the 'range' option to be used.
4278         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4279         \l__zrefclever_type_first_label_tl
4280         \l__zrefclever_label_a_tl
4281         \l__zrefclever_range_end_ref_tl
4282         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4283         {
4284           \exp_not:V \l__zrefclever_rangesep_tl
4285           \__zrefclever_get_ref_endrange:VVN
4286           \l__zrefclever_label_a_tl
4287           \l__zrefclever_range_end_ref_tl
4288           \l__zrefclever_refbounds_last_re_seq
4289         }
4290       }
4291     }
4292     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4293     {
4294       \exp_not:V \l__zrefclever_rangesep_tl

```



```

4295         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4296         \l__zrefclever_refbounds_last_re_seq
4297     }
4298 }
4299 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4300 \l__zrefclever_refbounds_first_rb_seq
4301 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4302 }
4303 }
4304 }
4305
4306 % If none of the special cases for the first of type refbounds have been
4307 % set, do it.
4308 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4309 {
4310     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4311     \l__zrefclever_refbounds_first_seq
4312 }
4313
4314 % Now that the type block is finished, we can add the name and the first
4315 % ref to the queue. Also, if "typeset" option is not "both", handle it
4316 % here as well.
4317 \_zrefclever_type_name_setup:
4318 \bool_if:nTF
4319 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4320 {
4321     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4322     { \_zrefclever_get_ref_first: }
4323 }
4324 {
4325     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4326     {
4327         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4328         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4329         {
4330             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4331             \l__zrefclever_type_first_refbounds_seq
4332         }
4333     }
4334     {
4335         \bool_if:NTF \l__zrefclever_typeset_name_bool
4336         {
4337             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4338             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4339             {
4340                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4341                 {
4342                     \exp_not:N \group_begin:
4343                     \exp_not:V \l__zrefclever_namefont_tl
4344                     % It's two '@s', but escaped for DocStrip.
4345                     \exp_not:N \hyper@@link
4346                     {
4347                         \_zrefclever_extract_url_unexp:V
4348                         \l__zrefclever_type_first_label_tl

```

```

4349     }
4350     {
4351         \__zrefclever_extract_unexp:Vnn
4352         \l__zrefclever_type_first_label_tl
4353         { anchor } { }
4354     }
4355     { \exp_not:V \l__zrefclever_type_name_tl }
4356     \exp_not:N \group_end:
4357 }
4358 {
4359     \exp_not:N \group_begin:
4360     \exp_not:V \l__zrefclever_namefont_tl
4361     \exp_not:V \l__zrefclever_type_name_tl
4362     \exp_not:N \group_end:
4363 }
4364 }
4365 }
4366 {
4367     % Logically, this case would correspond to "typeset=none", but
4368     % it should not occur, given that the options are set up to
4369     % typeset either "ref" or "name". Still, leave here a
4370     % sensible fallback, equal to the behavior of "both".
4371     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4372     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4373     { \__zrefclever_get_ref_first: }
4374 }
4375 }
4376 }
4377
4378 % Typeset the previous type block, if there is one.
4379 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4380 {
4381     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4382     { \l__zrefclever_tlistsep_tl }
4383     \l__zrefclever_typeset_queue_prev_tl
4384 }
4385
4386 % Extra log for testing.
4387 \bool_if:NT \l__zrefclever_verbose_testing_bool
4388 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4389
4390 % Wrap up loop, or prepare for next iteration.
4391 \bool_if:NTF \l__zrefclever_typeset_last_bool
4392 {
4393     % We are finishing, typeset the current queue.
4394     \int_case:nnF { \l__zrefclever_type_count_int }
4395     {
4396         % Single type.
4397         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4398         { 0 }
4399         { \l__zrefclever_typeset_queue_curr_tl }
4400         % Pair of types.
4401         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4402         { 1 }

```

```

4403         {
4404             \l__zrefclever_tpairsep_tl
4405             \l__zrefclever_typeset_queue_curr_tl
4406         }
4407     }
4408     {
4409         % Last in list of types.
4410         % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4411         \l__zrefclever_tlastsep_tl
4412         \l__zrefclever_typeset_queue_curr_tl
4413     }
4414     % And nudge in case of multitype reference.
4415     \bool_lazy_all:nT
4416     {
4417         { \l__zrefclever_nudge_enabled_bool }
4418         { \l__zrefclever_nudge_multitype_bool }
4419         { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4420     }
4421     { \msg_warning:nn { zref-clever } { nudge-multitype } }
4422 }
4423 {
4424     % There are further labels, set variables for next iteration.
4425     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4426         \l__zrefclever_typeset_queue_curr_tl
4427     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4428     \tl_clear:N \l__zrefclever_type_first_label_tl
4429     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4430     \tl_clear:N \l__zrefclever_range_beg_label_tl
4431     \tl_clear:N \l__zrefclever_range_end_ref_tl
4432     \int_zero:N \l__zrefclever_label_count_int
4433     \int_zero:N \l__zrefclever_ref_count_int
4434     \int_incr:N \l__zrefclever_type_count_int
4435     \int_zero:N \l__zrefclever_range_count_int
4436     \int_zero:N \l__zrefclever_range_same_count_int
4437     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4438     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4439 }
4440 }

```

(End definition for \\_\_zrefclever\_typeset\_refs\_last\_of\_type:.)

\\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type: Handles typesetting when the current label is not the last of its type.

```

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

```

```

4453         { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
4454     }
4455 }
4456
4457 % Process the current label to the current queue.
4458 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4459 {
4460     % Current label is the first of its type (also not the last, but it
4461     % doesn't matter here): just store the label.
4462     \tl_set:NV \l__zrefclever_type_first_label_tl
4463         \l__zrefclever_label_a_tl
4464     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4465         \l__zrefclever_label_type_a_tl
4466     \int_incr:N \l__zrefclever_ref_count_int
4467
4468     % If the next label may be part of a range, signal it (we deal with it
4469     % as the "first", and must do it there, to handle hyperlinking), but
4470     % also step the range counters.
4471     % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4472     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4473     {
4474         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4475         \tl_set:NV \l__zrefclever_range_beg_label_tl
4476             \l__zrefclever_label_a_tl
4477         \tl_clear:N \l__zrefclever_range_end_ref_tl
4478         \int_incr:N \l__zrefclever_range_count_int
4479         \bool_if:NT \l__zrefclever_next_is_same_bool
4480             { \int_incr:N \l__zrefclever_range_same_count_int }
4481     }
4482 }
4483 {
4484     % Current label is neither the first (nor the last) of its type.
4485     \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
4486     {
4487         % Starting, or continuing a range.
4488         \int_compare:nNnTF
4489             { \l__zrefclever_range_count_int } = { 0 }
4490         {
4491             % There was no range going, we are starting one.
4492             \tl_set:NV \l__zrefclever_range_beg_label_tl
4493                 \l__zrefclever_label_a_tl
4494             \tl_clear:N \l__zrefclever_range_end_ref_tl
4495             \int_incr:N \l__zrefclever_range_count_int
4496             \bool_if:NT \l__zrefclever_next_is_same_bool
4497                 { \int_incr:N \l__zrefclever_range_same_count_int }
4498         }
4499         {
4500             % Second or more in the range, but not the last.
4501             \int_incr:N \l__zrefclever_range_count_int
4502             \bool_if:NT \l__zrefclever_next_is_same_bool
4503                 { \int_incr:N \l__zrefclever_range_same_count_int }
4504         }
4505     }
4506 }

```

```

4507 % Next element is not in sequence: there was no range, or we are
4508 % closing one.
4509 \int_case:nNF { \l__zrefclever_range_count_int }
4510 {
4511   % There was no range going on.
4512   % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4513   { 0 }
4514   {
4515     \int_incr:N \l__zrefclever_ref_count_int
4516     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4517     {
4518       \exp_not:V \l__zrefclever_listsep_tl
4519       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4520       \l__zrefclever_refbounds_mid_seq
4521     }
4522   }
4523   % Last is second in the range: if 'range_same_count' is also
4524   % '1', it's a repetition (drop it), otherwise, it's a "pair
4525   % within a list", treat as list.
4526   % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4527   % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4528   { 1 }
4529   {
4530     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4531     {
4532       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4533       \l__zrefclever_refbounds_first_seq
4534       \bool_set_true:N
4535       \l__zrefclever_type_first_refbounds_set_bool
4536     }
4537     {
4538       \int_incr:N \l__zrefclever_ref_count_int
4539       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4540       {
4541         \exp_not:V \l__zrefclever_listsep_tl
4542         \__zrefclever_get_ref:VN
4543         \l__zrefclever_range_beg_label_tl
4544         \l__zrefclever_refbounds_mid_seq
4545       }
4546     }
4547     \int_compare:nNnF
4548     { \l__zrefclever_range_same_count_int } = { 1 }
4549     {
4550       \int_incr:N \l__zrefclever_ref_count_int
4551       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4552       {
4553         \exp_not:V \l__zrefclever_listsep_tl
4554         \__zrefclever_get_ref:VN
4555         \l__zrefclever_label_a_tl
4556         \l__zrefclever_refbounds_mid_seq
4557       }
4558     }
4559   }
4560 }

```

```

4561 {
4562 % Last is third or more in the range: if 'range_count' and
4563 % 'range_same_count' are the same, its a repetition (drop it),
4564 % if they differ by '1', its a list, if they differ by more,
4565 % it is a real range.
4566 \int_case:nnF
4567 {
4568   \l__zrefclever_range_count_int -
4569   \l__zrefclever_range_same_count_int
4570 }
4571 {
4572 % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4573 { 0 }
4574 {
4575   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4576   {
4577     \seq_set_eq:NN
4578     \l__zrefclever_type_first_refbounds_seq
4579     \l__zrefclever_refbounds_first_seq
4580     \bool_set_true:N
4581     \l__zrefclever_type_first_refbounds_set_bool
4582   }
4583   {
4584     \int_incr:N \l__zrefclever_ref_count_int
4585     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4586     {
4587       \exp_not:V \l__zrefclever_listsep_tl
4588       \__zrefclever_get_ref:VN
4589       \l__zrefclever_range_beg_label_tl
4590       \l__zrefclever_refbounds_mid_seq
4591     }
4592   }
4593 }
4594 % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4595 { 1 }
4596 {
4597   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4598   {
4599     \seq_set_eq:NN
4600     \l__zrefclever_type_first_refbounds_seq
4601     \l__zrefclever_refbounds_first_seq
4602     \bool_set_true:N
4603     \l__zrefclever_type_first_refbounds_set_bool
4604   }
4605   {
4606     \int_incr:N \l__zrefclever_ref_count_int
4607     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4608     {
4609       \exp_not:V \l__zrefclever_listsep_tl
4610       \__zrefclever_get_ref:VN
4611       \l__zrefclever_range_beg_label_tl
4612       \l__zrefclever_refbounds_mid_seq
4613     }
4614   }

```

```

4615 \int_incr:N \l__zrefclever_ref_count_int
4616 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4617 {
4618   \exp_not:V \l__zrefclever_listsep_tl
4619   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4620   \l__zrefclever_refbounds_mid_seq
4621 }
4622 }
4623 }
4624 {
4625   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4626   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4627   {
4628     \seq_set_eq:NN
4629     \l__zrefclever_type_first_refbounds_seq
4630     \l__zrefclever_refbounds_first_rb_seq
4631     \bool_set_true:N
4632     \l__zrefclever_type_first_refbounds_set_bool
4633   }
4634   {
4635     \int_incr:N \l__zrefclever_ref_count_int
4636     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4637     {
4638       \exp_not:V \l__zrefclever_listsep_tl
4639       \__zrefclever_get_ref:VN
4640       \l__zrefclever_range_beg_label_tl
4641       \l__zrefclever_refbounds_mid_rb_seq
4642     }
4643   }
4644   % For the purposes of the serial comma, and thus for the
4645   % distinction of 'lastsep' and 'pairsep', a "range" counts
4646   % as one. Since 'range_beg' has already been counted
4647   % (here or with the first of type), we refrain from
4648   % incrementing 'ref_count_int'.
4649   \bool_lazy_and:nnTF
4650   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4651   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4652   {
4653     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4654     \l__zrefclever_range_beg_label_tl
4655     \l__zrefclever_label_a_tl
4656     \l__zrefclever_range_end_ref_tl
4657     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4658     {
4659       \exp_not:V \l__zrefclever_rangesep_tl
4660       \__zrefclever_get_ref_endrange:VVN
4661       \l__zrefclever_label_a_tl
4662       \l__zrefclever_range_end_ref_tl
4663       \l__zrefclever_refbounds_mid_re_seq
4664     }
4665   }
4666   {
4667     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4668     {

```

```

4669             \exp_not:V \l__zrefclever_rangesep_tl
4670             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4671             \l__zrefclever_refbounds_mid_re_seq
4672         }
4673     }
4674 }
4675 }
4676 % Reset counters.
4677 \int_zero:N \l__zrefclever_range_count_int
4678 \int_zero:N \l__zrefclever_range_same_count_int
4679 }
4680 }
4681 % Step label counter for next iteration.
4682 \int_incr:N \l__zrefclever_label_count_int
4683 }

```

(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.

```

4684 \cs_new_protected:Npn \__zrefclever_ref_default:
4685 { \zref@default }
4686 \cs_new_protected:Npn \__zrefclever_name_default:
4687 { \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}

4688 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4689 {
4690   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4691   {
4692     \bool_if:nTF
4693     {
4694       \l__zrefclever_hyperlink_bool &&
4695       ! \l__zrefclever_link_star_bool
4696     }
4697     {
4698       \exp_not:N \group_begin:
4699       \exp_not:N \l__zrefclever_reffont_tl
4700       \seq_item:Nn #2 { 1 }
4701       % It's two '@s', but escaped for DocStrip.
4702       \exp_not:N \hyper@@link
4703       { \__zrefclever_extract_url_unexp:n {#1} }
4704       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4705       {
4706         \seq_item:Nn #2 { 2 }
4707         \__zrefclever_extract_unexp:nvn {#1}
4708         { \l__zrefclever_ref_property_tl } { }
4709         \seq_item:Nn #2 { 3 }
4710       }
4711       \seq_item:Nn #2 { 4 }
4712       \exp_not:N \group_end:
4713     }
4714     {
4715       \exp_not:N \group_begin:
4716       \exp_not:N \l__zrefclever_reffont_tl
4717       \seq_item:Nn #2 { 1 }
4718       \seq_item:Nn #2 { 2 }
4719       \__zrefclever_extract_unexp:nvn {#1}
4720       { \l__zrefclever_ref_property_tl } { }
4721       \seq_item:Nn #2 { 3 }
4722       \seq_item:Nn #2 { 4 }
4723       \exp_not:N \group_end:
4724     }
4725   }
4726   { \__zrefclever_ref_default: }
4727 }
4728 \cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }

(End definition for \__zrefclever_get_ref:nN.)

```

```

\__zrefclever_get_ref_endrange:nnN {\label} {\reference} {\refbounds}

4729 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3
4730 {

```

```

4731 \str_if_eq:nnTF {#2} { zc@missingproperty }
4732 { \__zrefclever_ref_default: }
4733 {
4734   \bool_if:nTF
4735   {
4736     \l__zrefclever_hyperlink_bool &&
4737     ! \l__zrefclever_link_star_bool
4738   }
4739   {
4740     \exp_not:N \group_begin:
4741     \exp_not:V \l__zrefclever_reffont_tl
4742     \seq_item:Nn #3 { 1 }
4743     % It's two '@s', but escaped for DocStrip.
4744     \exp_not:N \hyper@@link
4745     { \__zrefclever_extract_url_unexp:n {#1} }
4746     { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4747     {
4748       \seq_item:Nn #3 { 2 }
4749       \exp_not:n {#2}
4750       \seq_item:Nn #3 { 3 }
4751     }
4752     \seq_item:Nn #3 { 4 }
4753     \exp_not:N \group_end:
4754   }
4755   {
4756     \exp_not:N \group_begin:
4757     \exp_not:V \l__zrefclever_reffont_tl
4758     \seq_item:Nn #3 { 1 }
4759     \seq_item:Nn #3 { 2 }
4760     \exp_not:n {#2}
4761     \seq_item:Nn #3 { 3 }
4762     \seq_item:Nn #3 { 4 }
4763     \exp_not:N \group_end:
4764   }
4765 }
4766 }
4767 \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.

```

4768 \cs_new:Npn \__zrefclever_get_ref_first:
4769 {
4770   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4771   { \__zrefclever_ref_default: }
4772   {
4773     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4774 {
4775   \zref@ifrefcontainsprop
4776   { \l__zrefclever_type_first_label_tl }
4777   { \l__zrefclever_ref_property_tl }
4778   {
4779     \exp_not:N \group_begin:
4780     % It's two '@s', but escaped for DocStrip.
4781     \exp_not:N \hyper@@link
4782     {
4783       \__zrefclever_extract_url_unexp:V
4784       \l__zrefclever_type_first_label_tl
4785     }
4786     {
4787       \__zrefclever_extract_unexp:Vnn
4788       \l__zrefclever_type_first_label_tl { anchor } { }
4789     }
4790     {
4791       \exp_not:N \group_begin:
4792       \exp_not:V \l__zrefclever_namefont_tl
4793       \exp_not:V \l__zrefclever_type_name_tl
4794       \exp_not:N \group_end:
4795       \exp_not:V \l__zrefclever_namesep_tl
4796       \exp_not:N \group_begin:
4797       \exp_not:V \l__zrefclever_reffont_tl
4798       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4799       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4800       \__zrefclever_extract_unexp:Vvn
4801       \l__zrefclever_type_first_label_tl
4802       { \l__zrefclever_ref_property_tl } { }
4803       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4804       \exp_not:N \group_end:
4805     }
4806     \exp_not:V \l__zrefclever_reffont_tl
4807     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4808     \exp_not:N \group_end:
4809   }
4810   {
4811     \exp_not:N \group_begin:
4812     \exp_not:V \l__zrefclever_namefont_tl
4813     \exp_not:V \l__zrefclever_type_name_tl
4814     \exp_not:N \group_end:
4815     \exp_not:V \l__zrefclever_namesep_tl
4816     \__zrefclever_ref_default:
4817   }
4818 }
4819 {
4820   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4821   {
4822     \__zrefclever_name_default:
4823     \exp_not:V \l__zrefclever_namesep_tl
4824   }
4825   {
4826     \exp_not:N \group_begin:
4827     \exp_not:V \l__zrefclever_namefont_tl

```

```

4828         \exp_not:V \l__zrefclever_type_name_tl
4829         \exp_not:N \group_end:
4830         \tl_if_empty:NF \l__zrefclever_type_name_tl
4831         { \exp_not:V \l__zrefclever_namesep_tl }
4832     }
4833 \zref@ifrefcontainsprop
4834 { \l__zrefclever_type_first_label_tl }
4835 { \l__zrefclever_ref_property_tl }
4836 {
4837     \bool_if:nTF
4838     {
4839         \l__zrefclever_hyperlink_bool &&
4840         ! \l__zrefclever_link_star_bool
4841     }
4842     {
4843         \exp_not:N \group_begin:
4844         \exp_not:V \l__zrefclever_reffont_tl
4845         \seq_item:Nn
4846         \l__zrefclever_type_first_refbounds_seq { 1 }
4847         % It's two '@s', but escaped for DocStrip.
4848         \exp_not:N \hyper@@link
4849         {
4850             \__zrefclever_extract_url_unexp:V
4851             \l__zrefclever_type_first_label_tl
4852         }
4853         {
4854             \__zrefclever_extract_unexp:Vnn
4855             \l__zrefclever_type_first_label_tl { anchor } { }
4856         }
4857         {
4858             \seq_item:Nn
4859             \l__zrefclever_type_first_refbounds_seq { 2 }
4860             \__zrefclever_extract_unexp:Vnn
4861             \l__zrefclever_type_first_label_tl
4862             { \l__zrefclever_ref_property_tl } { }
4863             \seq_item:Nn
4864             \l__zrefclever_type_first_refbounds_seq { 3 }
4865         }
4866         \seq_item:Nn
4867         \l__zrefclever_type_first_refbounds_seq { 4 }
4868         \exp_not:N \group_end:
4869     }
4870     {
4871         \exp_not:N \group_begin:
4872         \exp_not:V \l__zrefclever_reffont_tl
4873         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4874         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4875         \__zrefclever_extract_unexp:Vnn
4876         \l__zrefclever_type_first_label_tl
4877         { \l__zrefclever_ref_property_tl } { }
4878         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4879         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4880         \exp_not:N \group_end:
4881     }

```

```

4882     }
4883     { \_zrefclever_ref_default: }
4884   }
4885 }
4886 }

```

(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.

```

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

```

```

4921         ! \l__zrefclever_noabbrev_first_bool
4922     }
4923     {
4924         \tl_set:NV \l__zrefclever_name_format_fallback_tl
4925             \l__zrefclever_name_format_tl
4926         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4927     }
4928     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4929
4930 % Handle number and gender nudges.
4931 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4932 {
4933     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4934     {
4935         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4936         {
4937             \msg_warning:nnx { zref-clever }
4938                 { nudge-plural-when-sg }
4939                 { \l__zrefclever_type_first_label_type_tl }
4940         }
4941     }
4942     {
4943         \bool_lazy_all:nT
4944         {
4945             { \l__zrefclever_nudge_comptosing_bool }
4946             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4947             {
4948                 \int_compare_p:nNn
4949                     { \l__zrefclever_label_count_int } > { 0 }
4950             }
4951         }
4952         {
4953             \msg_warning:nnx { zref-clever }
4954                 { nudge-comptosing }
4955                 { \l__zrefclever_type_first_label_type_tl }
4956         }
4957     }
4958     \bool_lazy_and:nnT
4959     { \l__zrefclever_nudge_gender_bool }
4960     { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4961     {
4962         \__zrefclever_get_rf_opt_seq:nxxN { gender }
4963         { \l__zrefclever_type_first_label_type_tl }
4964         { \l__zrefclever_ref_language_tl }
4965         \l__zrefclever_type_name_gender_seq
4966         \seq_if_in:NVF
4967             \l__zrefclever_type_name_gender_seq
4968             \l__zrefclever_ref_gender_tl
4969         {
4970             \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4971             {
4972                 \msg_warning:nnxxx { zref-clever }
4973                     { nudge-gender-not-declared-for-type }
4974                     { \l__zrefclever_ref_gender_tl }

```

```

4975         { \l__zrefclever_type_first_label_type_tl }
4976         { \l__zrefclever_ref_language_tl }
4977     }
4978     {
4979         \msg_warning:nxxxxx { zref-clever }
4980         { nudge-gender-mismatch }
4981         { \l__zrefclever_type_first_label_type_tl }
4982         { \l__zrefclever_ref_gender_tl }
4983         {
4984             \seq_use:Nn
4985                 \l__zrefclever_type_name_gender_seq { ,~ }
4986         }
4987         { \l__zrefclever_ref_language_tl }
4988     }
4989 }
4990 }
4991 }
4992
4993 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4994 {
4995     \__zrefclever_opt_tl_get:cNF
4996     {
4997         \__zrefclever_opt_varname_type:een
4998         { \l__zrefclever_type_first_label_type_tl }
4999         { \l__zrefclever_name_format_tl }
5000         { tl }
5001     }
5002     \l__zrefclever_type_name_tl
5003     {
5004         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5005         {
5006             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5007             \tl_put_left:NV \l__zrefclever_name_format_tl
5008                 \l__zrefclever_ref_decl_case_tl
5009         }
5010         \__zrefclever_opt_tl_get:cNF
5011         {
5012             \__zrefclever_opt_varname_lang_type:eeen
5013             { \l__zrefclever_ref_language_tl }
5014             { \l__zrefclever_type_first_label_type_tl }
5015             { \l__zrefclever_name_format_tl }
5016             { tl }
5017         }
5018         \l__zrefclever_type_name_tl
5019         {
5020             \tl_clear:N \l__zrefclever_type_name_tl
5021             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5022             \msg_warning:nxxx { zref-clever } { missing-name }
5023             { \l__zrefclever_name_format_tl }
5024             { \l__zrefclever_type_first_label_type_tl }
5025         }
5026     }
5027 }
5028 {

```

```

5029 \__zrefclever_opt_tl_get:cnf
5030 {
5031   \__zrefclever_opt_varname_type:een
5032   { \l__zrefclever_type_first_label_type_tl }
5033   { \l__zrefclever_name_format_tl }
5034   { tl }
5035 }
5036 \l__zrefclever_type_name_tl
5037 {
5038   \__zrefclever_opt_tl_get:cnf
5039   {
5040     \__zrefclever_opt_varname_type:een
5041     { \l__zrefclever_type_first_label_type_tl }
5042     { \l__zrefclever_name_format_fallback_tl }
5043     { tl }
5044   }
5045   \l__zrefclever_type_name_tl
5046   {
5047     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5048     {
5049       \tl_put_left:Nn
5050       \l__zrefclever_name_format_tl { - }
5051       \tl_put_left:NV \l__zrefclever_name_format_tl
5052       \l__zrefclever_ref_decl_case_tl
5053       \tl_put_left:Nn
5054       \l__zrefclever_name_format_fallback_tl { - }
5055       \tl_put_left:NV
5056       \l__zrefclever_name_format_fallback_tl
5057       \l__zrefclever_ref_decl_case_tl
5058     }
5059     \__zrefclever_opt_tl_get:cnf
5060     {
5061       \__zrefclever_opt_varname_lang_type:een
5062       { \l__zrefclever_ref_language_tl }
5063       { \l__zrefclever_type_first_label_type_tl }
5064       { \l__zrefclever_name_format_tl }
5065       { tl }
5066     }
5067     \l__zrefclever_type_name_tl
5068     {
5069       \__zrefclever_opt_tl_get:cnf
5070       {
5071         \__zrefclever_opt_varname_lang_type:een
5072         { \l__zrefclever_ref_language_tl }
5073         { \l__zrefclever_type_first_label_type_tl }
5074         { \l__zrefclever_name_format_fallback_tl }
5075         { tl }
5076       }
5077       \l__zrefclever_type_name_tl
5078       {
5079         \tl_clear:N \l__zrefclever_type_name_tl
5080         \bool_set_true:N
5081         \l__zrefclever_type_name_missing_bool
5082         \msg_warning:nxxx { zref-clever }

```



```

5083                                     { missing-name }
5084                                     { \l__zrefclever_name_format_tl }
5085                                     { \l__zrefclever_type_first_label_type_tl }
5086                                 }
5087                            }
5088                       }
5089                  }
5090             }
5091        }
5092    }
5093
5094    % Signal whether the type name is to be included in the hyperlink or not.
5095    \bool_lazy_any:nTF
5096    {
5097        { ! \l__zrefclever_hyperlink_bool }
5098        { \l__zrefclever_link_star_bool }
5099        { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5100        { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5101    }
5102    { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5103    {
5104        \bool_lazy_any:nTF
5105        {
5106            { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5107            {
5108                \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5109                \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5110            }
5111            {
5112                \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5113                \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5114                \l__zrefclever_typeset_last_bool &&
5115                \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5116            }
5117        }
5118        { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5119        { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5120    }
5121 }

```

(End definition for \\_\_zrefclever\_type\_name\_setup:.)

\\_\_zrefclever\_extract\_url\_unexp:n A convenience auxiliary function for extraction of the url / urluse property, provided by the zref-xr module. Ensure that, in the context of an x expansion, \zref@extractdefault is expanded exactly twice, but no further to retrieve the proper value. See documentation for \\_\_zrefclever\_extract\_unexp:nnn.

```

5122 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5123 {
5124     \zref@ifpropundefined { urluse }
5125     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5126     {
5127         \zref@ifrefcontainsprop {#1} { urluse }
5128         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5129         { \__zrefclever_extract_unexp:nnn {#1} { url } { } }

```

```

5130     }
5131   }
5132   \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for \\_\_zrefclever\_extract\_url\_unexp:n.)

\\_\_zrefclever\_labels\_in\_sequence:nn Auxiliary function to \\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type:. Sets \l\_\_zrefclever\_next\_maybe\_range\_bool to true if  $\langle label\ b \rangle$  comes in immediate sequence from  $\langle label\ a \rangle$ . And sets both \l\_\_zrefclever\_next\_maybe\_range\_bool and \l\_\_zrefclever\_next\_is\_same\_bool to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside \\_\_zrefclever\_typeset\_refs\_not\_last\_of\_type:, so this function is expected to be called at its beginning, if compression is enabled.

```

\__zrefclever_labels_in_sequence:nn {\langle label\ a \rangle} {\langle label\ b \rangle}

5133 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5134 {
5135   \exp_args:Nxx \tl_if_eq:nnT
5136     { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5137     { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5138   {
5139     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5140     {
5141       \exp_args:Nxx \tl_if_eq:nnT
5142         { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5143         { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5144       {
5145         \int_compare:nNnTF
5146           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5147           =
5148           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5149           { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5150         {
5151           \int_compare:nNnT
5152             { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5153             =
5154             { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5155           {
5156             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5157             \bool_set_true:N \l__zrefclever_next_is_same_bool
5158           }
5159         }
5160       }
5161     }
5162   {
5163     \exp_args:Nxx \tl_if_eq:nnT
5164       { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5165       { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5166     {
5167       \exp_args:Nxx \tl_if_eq:nnT
5168         { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5169         { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5170       {
5171         \int_compare:nNnTF

```

```

5172 { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5173 =
5174 { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5175 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5176 {
5177   \int_compare:nNtT
5178     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5179     =
5180     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5181   {

```

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.

```

5182 \exp_args:Nxx \tl_if_eq:nnT
5183 {
5184   \_zrefclever_extract_unexp:nvn {#1}
5185     { l__zrefclever_ref_property_tl } { }
5186 }
5187 {
5188   \_zrefclever_extract_unexp:nvn {#2}
5189     { l__zrefclever_ref_property_tl } { }
5190 }
5191 {
5192   \bool_set_true:N
5193     \l__zrefclever_next_maybe_range_bool
5194   \bool_set_true:N
5195     \l__zrefclever_next_is_same_bool
5196 }
5197 }
5198 }
5199 }
5200 }
5201 }
5202 }
5203 }

```

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

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

```

\_zrefclever_get_rf_opt_tl:nnnN \_zrefclever_get_rf_opt_tl:nnnN {<option>}
  {<ref type>} {<language>} {<tl variable>}
5204 \cs_new_protected:Npn \_zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5205 {
5206   % First attempt: general options.
5207   \_zrefclever_opt_tl_get:cNF
5208     { \_zrefclever_opt_varname_general:nn {#1} { tl } }
5209     #4
5210   {

```

```

5211 % If not found, try type specific options.
5212 \_zrefclever_opt_tl_get:cnf
5213 { \_zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5214 #4
5215 {
5216 % If not found, try type- and language-specific.
5217 \_zrefclever_opt_tl_get:cnf
5218 { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5219 #4
5220 {
5221 % If not found, try language-specific default.
5222 \_zrefclever_opt_tl_get:cnf
5223 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5224 #4
5225 {
5226 % If not found, try fallback.
5227 \_zrefclever_opt_tl_get:cnf
5228 { \_zrefclever_opt_varname_fallback:nn {#1} { tl } }
5229 #4
5230 { \tl_clear:N #4 }
5231 }
5232 }
5233 }
5234 }
5235 }
5236 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_tl:nnnN { nxxN }

```

(End definition for \\_zrefclever\_get\_rf\_opt\_tl:nnnN.)

```

\_zrefclever_get_rf_opt_seq:nnnN \_zrefclever_get_rf_opt_seq:nnnN {<option>}
{<ref type>} {<language>} {<seq variable>}
5237 \cs_new_protected:Npn \_zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5238 {
5239 % First attempt: general options.
5240 \_zrefclever_opt_seq_get:cnf
5241 { \_zrefclever_opt_varname_general:nn {#1} { seq } }
5242 #4
5243 {
5244 % If not found, try type specific options.
5245 \_zrefclever_opt_seq_get:cnf
5246 { \_zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5247 #4
5248 {
5249 % If not found, try type- and language-specific.
5250 \_zrefclever_opt_seq_get:cnf
5251 { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5252 #4
5253 {
5254 % If not found, try language-specific default.
5255 \_zrefclever_opt_seq_get:cnf
5256 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5257 #4
5258 {
5259 % If not found, try fallback.

```

```

5260         \_zrefclever_opt_seq_get:cNF
5261         { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5262         #4
5263         { \seq_clear:N #4 }
5264     }
5265 }
5266 }
5267 }
5268 }
5269 \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>}
5270 \cs_new_protected:Npn \_zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5271 {
5272     % First attempt: general options.
5273     \_zrefclever_opt_bool_get:cNF
5274     { \_zrefclever_opt_varname_general:nn {#1} { bool } }
5275     #5
5276     {
5277         % If not found, try type specific options.
5278         \_zrefclever_opt_bool_get:cNF
5279         { \_zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5280         #5
5281         {
5282             % If not found, try type- and language-specific.
5283             \_zrefclever_opt_bool_get:cNF
5284             { \_zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5285             #5
5286             {
5287                 % If not found, try language-specific default.
5288                 \_zrefclever_opt_bool_get:cNF
5289                 { \_zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5290                 #5
5291                 {
5292                     % If not found, try fallback.
5293                     \_zrefclever_opt_bool_get:cNF
5294                     { \_zrefclever_opt_varname_fallback:nn {#1} { bool } }
5295                     #5
5296                     { \use:c { bool_set_ #2 :N } #5 }
5297                 }
5298             }
5299         }
5300     }
5301 }
5302 \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.

```
5303 \__zrefclever_compat_module:nn { appendix }
5304 {
5305   \AddToHook { cmd / appendix / before }
5306   {
5307     \__zrefclever_zcsetup:n
5308     {
5309       countertype =
5310       {
5311         chapter      = appendix ,
5312         section      = appendix ,
5313         subsection   = appendix ,
5314         subsubsection = appendix ,
5315         paragraph    = appendix ,
5316         subparagraph = appendix ,
5317       }
5318     }
5319   }
5320 }
```

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.

```

5321 \__zrefclever_compat_module:nn { appendices }
5322 {
5323   \__zrefclever_if_package_loaded:nT { appendix }
5324   {
5325     \newcounter { zc@appendix }
5326     \newcounter { zc@save@appendix }
5327     \setcounter { zc@appendix } { 0 }
5328     \setcounter { zc@save@appendix } { 0 }
5329     \cs_if_exist:cTF { chapter }
5330     {
5331       \__zrefclever_zcsetup:n
5332       { counterresetby = { chapter = zc@appendix } }
5333     }
5334     {
5335       \cs_if_exist:cT { section }
5336       {
5337         \__zrefclever_zcsetup:n
5338         { counterresetby = { section = zc@appendix } }
5339       }
5340     }
5341   \AddToHook { env / appendices / begin }
5342   {
5343     \stepcounter { zc@save@appendix }
5344     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5345     \__zrefclever_zcsetup:n
5346     {
5347       countertype =
5348       {
5349         chapter      = appendix ,
5350         section      = appendix ,
5351         subsection   = appendix ,
5352         subsubsection = appendix ,
5353         paragraph    = appendix ,
5354         subparagraph = appendix ,
5355       }
5356     }
5357   }
5358   \AddToHook { env / appendices / end }
5359   { \setcounter { zc@appendix } { 0 } }
5360   \AddToHook { cmd / appendix / before }
5361   {
5362     \stepcounter { zc@save@appendix }
5363     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5364   }
5365   \AddToHook { env / subappendices / begin }
5366   {
5367     \__zrefclever_zcsetup:n
5368     {
5369       countertype =
5370       {

```

```

5371             section      = appendix ,
5372             subsection   = appendix ,
5373             subsubsection = appendix ,
5374             paragraph     = appendix ,
5375             subparagraph  = appendix ,
5376         } ,
5377     }
5378 }
5379 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5380 }
5381 }

```

### 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.

```

5382 \__zrefclever_compat_module:nn { memoir }
5383 {
5384   \__zrefclever_if_class_loaded:nT { memoir }
5385   {

```

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`.

```

5386   \__zrefclever_zcsetup:n
5387   {
5388     countertype =
5389     {
5390       subfigure = figure ,
5391       subtable  = table  ,
5392       poemline  = line   ,
5393     } ,
5394     counterresetby =
5395     {
5396       subfigure = figure ,
5397       subtable  = table  ,
5398     } ,
5399   }

```

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

```

5400   \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5401   {
5402     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5403     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5404     {
5405       \__zrefclever_memoir_orig_label:n {##1}
5406       \zlabel{##1}
5407     }
5408     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
5409   }
5410   \AddToHook { env / sidecaption / begin }
5411   { \__zrefclever_memoir_both_labels: }
5412   \AddToHook { env / sidecontcaption / begin }
5413   { \__zrefclever_memoir_both_labels: }

```

```

5414 \AddToHook{ cmd / bitwonumcaption / before }
5415 { \group_begin: \__zrefclever_memoir_both_labels: }
5416 \AddToHook{ cmd / bitwonumcaption / after }
5417 { \group_end: }
5418 \AddToHook{ cmd / bionenumcaption / before }
5419 { \group_begin: \__zrefclever_memoir_both_labels: }
5420 \AddToHook{ cmd / bionenumcaption / after }
5421 { \group_end: }
5422 \AddToHook{ cmd / bicaption / before }
5423 { \group_begin: \__zrefclever_memoir_both_labels: }
5424 \AddToHook{ cmd / bicaption / after }
5425 { \group_end: }

```

Support for subcaption reference.

```

5426 \zref@newprop { subcaption }
5427 { \cs_if_exist_use:c { @@thesub \@captype } }
5428 \AddToHook{ cmd / @memsubcaption / before }
5429 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5430 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5431 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5432 \AddToHook { env / minipage / begin }
5433 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5434 \AddToHook { cmd / @makefntext / before }
5435 {
5436   \__zrefclever_zcsetup:x
5437   { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5438 }
5439 \AddToHook { cmd / @makesidefntext / before }
5440 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5441 \__zrefclever_zcsetup:n
5442 {
5443   countertype =
5444   {
5445     sidefootnote = footnote ,
5446     pagenote = endnote ,
5447   } ,
5448 }
5449 \AddToHook { file / \jobname.ent / before }
5450 { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5451 \msg_info:nnn { zref-clever } { compat-class } { memoir }
5452 }
5453 }

```

## 9.4 KOMA

Support for KOMA-Script document classes.

```

5454 \__zrefclever_compat_module:nn { KOMA }
5455 {
5456   \cs_if_exist:NT \KOMAClassName
5457   {

```

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.

```

5458     \AddToHook { env / captionbeside / after }
5459     {
5460         \tl_if_exist:NT \@capttype
5461         { \tl_set_eq:NN \@currentcounter \@capttype }
5462     }
5463     \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
5464     \AddToHook { env / captionofbeside / end }
5465     { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
5466     \AddToHook { env / captionofbeside / after }
5467     {
5468         \tl_if_eq:NnF \@currenenvir { document }
5469         {
5470             \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
5471             {
5472                 \tl_set_eq:NN
5473                 \@currentcounter \g__zrefclever_koma_capttype_tl
5474             }
5475         }
5476         \tl_gclear:N \g__zrefclever_koma_capttype_tl
5477     }
5478     \msg_info:nmx { zref-clever } { compat-class } { \KOMAClassName }
5479 }
5480 }
```

## 9.5 amsmath

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

```

5481 \__zrefclever_compat_module:n { amsmath }
5482 {
5483     \__zrefclever_if_package_loaded:nT { amsmath }
5484     {
```

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...).

```

5485         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5486         {
```

```

5487         \_zrefclever_orig_ltxlabel:n {#1}
5488         \zref@wrapper@babel \zref@label {#1}
5489     }

```

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

```

5490     \_zrefclever_if_package_loaded:nTF { hyperref }
5491     {
5492         \AddToHook { package / nameref / after }
5493         {
5494             \cs_new_eq:NN \_zrefclever_orig_ltxlabel:n \ltx@label
5495             \cs_set_eq:NN \ltx@label \_zrefclever_ltxlabel:n
5496         }
5497     }
5498     {
5499         \cs_new_eq:NN \_zrefclever_orig_ltxlabel:n \ltx@label
5500         \cs_set_eq:NN \ltx@label \_zrefclever_ltxlabel:n
5501     }

```

The `subequations` environment uses `parentequation` and `equation` as counters, but only the later is subject to `\refstepcounter`. What happens is: at the start, `equation` is refstepped, it is then stored in `parentequation` and set to '0' and, at the end of the environment it is restored to the value of `parentequation`. We cannot even set `\@currentcounter` at `env/.../begin`, since the call to `\refstepcounter{equation}` done by `subequations` will override that in sequence. Unfortunately, the suggestion to set `\@currentcounter` to `parentequation` here was not accepted, see <https://github.com/latex3/latex2e/issues/687#issuecomment-951451024> and subsequent discussion. So, for `subequations`, we really must specify manually `currentcounter` and the resetting. Note that, for `subequations`, `\zlabel` works just fine (that is, if given immediately after `\begin{subequations}`}, to refer to the parent equation).

```

5502     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5503     \AddToHook { env / subequations / begin }

```

```

5504 {
5505   \_zrefclever_zcsetup:x
5506   {
5507     counterresetby =
5508     {
5509       parentequation =
5510       \_zrefclever_counter_reset_by:n { equation } ,
5511       equation = parentequation ,
5512     } ,
5513     currentcounter = parentequation ,
5514     countertype = { parentequation = equation } ,
5515   }
5516   \bool_set_true:N \l_zrefclever_amsmath_subequations_bool
5517 }

```

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.

```

5518 \zref@newprop { subeq } { \alph { equation } }
5519 \clist_map_inline:nn
5520 {
5521   equation ,
5522   equation* ,
5523   align ,
5524   align* ,
5525   alignat ,
5526   alignat* ,
5527   flalign ,
5528   flalign* ,
5529   xalignat ,
5530   xalignat* ,
5531   gather ,
5532   gather* ,
5533   multline ,
5534   multline* ,
5535 }
5536 {
5537   \AddToHook { env / #1 / begin }
5538   {
5539     \_zrefclever_zcsetup:n { currentcounter = equation }
5540     \bool_if:NT \l_zrefclever_amsmath_subequations_bool
5541       { \zref@localaddprop \ZREF@mainlist { subeq } }
5542   }
5543 }

```

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

```

5544     \zcRefTypeSetup { equation }
5545     { reffont = \upshape }
5546     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5547   }
5548 }

```

## 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.

```

5549 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5550 \__zrefclever_compat_module:nn { mathtools }
5551 {
5552   \__zrefclever_if_package_loaded:nT { mathtools }
5553   {
5554     \MH_if_boolean:nT { show_only_refs }
5555     {
5556       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5557       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5558       {
5559         \@bsphack
5560         \seq_map_inline:Nn #1
5561         {
5562           \exp_args:Nx \tl_if_eq:nnTF
5563           { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5564           { equation }
5565           {
5566             \protected@write \@auxout { }
5567             { \string \MT@newlabel {##1} }
5568           }
5569         }
5570         \exp_args:Nx \tl_if_eq:nnT
5571         { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5572         { parentequation }
5573         {
5574           \protected@write \@auxout { }
5575           { \string \MT@newlabel {##1} }
5576         }
5577       }
5578     }
5579     \@esphack
5580   }
5581   \msg_info:nnn { zref-clever } { compat-package } { mathtools }

```

```

5582     }
5583   }
5584 }

```

## 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”.

```

5585 \__zrefclever_compat_module:nn { breqn }
5586 {
5587   \__zrefclever_if_package_loaded:NT { breqn }
5588   {

```

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>).

```

5589   \bool_new:N \l__zrefclever_breqn_dgroup_bool
5590   \AddToHook { env / dgroup / begin }
5591   {
5592     \__zrefclever_zcsetup:x
5593     {
5594       counterresetby =
5595       {
5596         parentequation =
5597           \__zrefclever_counter_reset_by:n { equation } ,
5598         equation = parentequation ,
5599       } ,
5600       currentcounter = parentequation ,
5601       countertype = { parentequation = equation } ,
5602     }
5603     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5604   }
5605   \zref@ifpropundefined { subeq }
5606   { \zref@newprop { subeq } { \alph { equation } } }
5607   { }
5608   \clist_map_inline:nn
5609   {
5610     dmath ,
5611     dseries ,
5612     darray ,
5613   }
5614   {
5615     \AddToHook { env / #1 / begin }
5616     {
5617       \__zrefclever_zcsetup:n { currentcounter = equation }
5618       \bool_if:NT \l__zrefclever_breqn_dgroup_bool

```

```

5619             { \zref@localaddprop \ZREF@mainlist { subeq } }
5620         }
5621     }
5622     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5623 }
5624 }

```

## 9.8 listings

```

5625 \__zrefclever_compat_module:nn { listings }
5626 {
5627     \__zrefclever_if_package_loaded:nT { listings }
5628     {
5629         \__zrefclever_zcsetup:n
5630         {
5631             countertype =
5632             {
5633                 lstlisting = listing ,
5634                 lstnumber = line ,
5635             } ,
5636             counterresetby = { lstnumber = lstlisting } ,
5637         }

```

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.

```

5638     \lst@AddToHook { PreInit }
5639     { \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.

```

5640     \lst@AddToHook { Init }
5641     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5642     \msg_info:nnn { zref-clever } { compat-package } { listings }
5643 }
5644 }

```

## 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  $\{max-depth\}$ . `\renewlist` *hard-codes* `max-depth` in the environment’s definition (well, just as the kernel does), so we cannot retrieve this information from any sort of variable. But `\renewlist` also creates any needed missing counters, so we can use their existence to make the appropriate settings. In the end, the existence of the counters is indeed what matters from `zref-clever`’s perspective. Since the first four are defined by the kernel and already setup for `zref-clever` by default, we start from 5, and stop at the first non-existent `\c@enumN` counter.

```

5645 \_zrefclever_compat_module:nn { enumitem }
5646 {
5647   \_zrefclever_if_package_loaded:nT { enumitem }
5648   {
5649     \int_set:Nn \l_tmpa_int { 5 }
5650     \bool_while_do:nn
5651     {
5652       \cs_if_exist_p:c
5653       { c@ enum \int_to_roman:n { \l_tmpa_int } }
5654     }
5655     {
5656       \_zrefclever_zcsetup:x
5657       {
5658         counterresetby =
5659         {
5660           enum \int_to_roman:n { \l_tmpa_int } =
5661           enum \int_to_roman:n { \l_tmpa_int - 1 }
5662         } ,
5663         countertype =
5664         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5665       }
5666       \int_incr:N \l_tmpa_int
5667     }
5668     \int_compare:nNnT { \l_tmpa_int } > { 5 }
5669     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5670   }
5671 }

```

## 9.10 subcaption

```

5672 \_zrefclever_compat_module:nn { subcaption }
5673 {
5674   \_zrefclever_if_package_loaded:nT { subcaption }
5675   {
5676     \_zrefclever_zcsetup:n
5677     {
5678       countertype =
5679       {
5680         subfigure = figure ,
5681         subtable = table ,
5682       } ,
5683       counterresetby =
5684       {
5685         subfigure = figure ,
5686         subtable = table ,

```

```

5687         } ,
5688     }

```

Support for `subref` reference.

```

5689     \zref@newprop { subref }
5690     { \cs_if_exist_use:c { thesub \@captype } }
5691     \tl_put_right:Nn \caption@subtyp hook
5692     { \zref@localaddprop \ZREF@mainlist { subref } }
5693 }
5694 }

```

## 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.

```

5695 \__zrefclever_compat_module:nn { subfig }
5696 {
5697     \__zrefclever_if_package_loaded:nT { subfig }
5698     {
5699         \__zrefclever_zcsetup:n
5700         {
5701             countertype =
5702             {
5703                 subfigure = figure ,
5704                 subtable = table ,
5705             } ,
5706             counterresetby =
5707             {
5708                 subfigure = figure ,
5709                 subtable = table ,
5710             } ,
5711         }
5712     }
5713 }
5714 </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 English

English language file has been initially provided by the author.

```

5715 <*package>
5716 \zcDeclareLanguage { english }
5717 \zcDeclareLanguageAlias { american } { english }
5718 \zcDeclareLanguageAlias { australian } { english }
5719 \zcDeclareLanguageAlias { british } { english }
5720 \zcDeclareLanguageAlias { canadian } { english }

```

```

5721 \zcDeclareLanguageAlias { newzealand } { english }
5722 \zcDeclareLanguageAlias { UKenglish } { english }
5723 \zcDeclareLanguageAlias { USenglish } { english }
5724 \end{package}

5725 \begin{lang-english}

5726 namesep = {\nobreakspace} ,
5727 pairsep = {\and\nobreakspace} ,
5728 listsep = {,~} ,
5729 lastsep = {\and\nobreakspace} ,
5730 tpairsep = {\and\nobreakspace} ,
5731 tlistsep = {,~} ,
5732 tlastsep = {\and\nobreakspace} ,
5733 notesep = {~} ,
5734 rangesep = {\to\nobreakspace} ,
5735
5736 type = book ,
5737   Name-sg = Book ,
5738   name-sg = book ,
5739   Name-pl = Books ,
5740   name-pl = books ,
5741
5742 type = part ,
5743   Name-sg = Part ,
5744   name-sg = part ,
5745   Name-pl = Parts ,
5746   name-pl = parts ,
5747
5748 type = chapter ,
5749   Name-sg = Chapter ,
5750   name-sg = chapter ,
5751   Name-pl = Chapters ,
5752   name-pl = chapters ,
5753
5754 type = section ,
5755   Name-sg = Section ,
5756   name-sg = section ,
5757   Name-pl = Sections ,
5758   name-pl = sections ,
5759
5760 type = paragraph ,
5761   Name-sg = Paragraph ,
5762   name-sg = paragraph ,
5763   Name-pl = Paragraphs ,
5764   name-pl = paragraphs ,
5765   Name-sg-ab = Par. ,
5766   name-sg-ab = par. ,
5767   Name-pl-ab = Par. ,
5768   name-pl-ab = par. ,
5769
5770 type = appendix ,
5771   Name-sg = Appendix ,
5772   name-sg = appendix ,
5773   Name-pl = Appendices ,

```

```

5774     name-pl = appendices ,
5775
5776 type = page ,
5777     Name-sg = Page ,
5778     name-sg = page ,
5779     Name-pl = Pages ,
5780     name-pl = pages ,
5781     rangesep = {\textendash} ,
5782     rangetopair = false ,
5783
5784 type = line ,
5785     Name-sg = Line ,
5786     name-sg = line ,
5787     Name-pl = Lines ,
5788     name-pl = lines ,
5789
5790 type = figure ,
5791     Name-sg = Figure ,
5792     name-sg = figure ,
5793     Name-pl = Figures ,
5794     name-pl = figures ,
5795     Name-sg-ab = Fig. ,
5796     name-sg-ab = fig. ,
5797     Name-pl-ab = Figs. ,
5798     name-pl-ab = figs. ,
5799
5800 type = table ,
5801     Name-sg = Table ,
5802     name-sg = table ,
5803     Name-pl = Tables ,
5804     name-pl = tables ,
5805
5806 type = item ,
5807     Name-sg = Item ,
5808     name-sg = item ,
5809     Name-pl = Items ,
5810     name-pl = items ,
5811
5812 type = footnote ,
5813     Name-sg = Footnote ,
5814     name-sg = footnote ,
5815     Name-pl = Footnotes ,
5816     name-pl = footnotes ,
5817
5818 type = endnote ,
5819     Name-sg = Note ,
5820     name-sg = note ,
5821     Name-pl = Notes ,
5822     name-pl = notes ,
5823
5824 type = note ,
5825     Name-sg = Note ,
5826     name-sg = note ,
5827     Name-pl = Notes ,

```

```

5828     name-pl = notes ,
5829
5830 type = equation ,
5831     Name-sg = Equation ,
5832     name-sg = equation ,
5833     Name-pl = Equations ,
5834     name-pl = equations ,
5835     Name-sg-ab = Eq. ,
5836     name-sg-ab = eq. ,
5837     Name-pl-ab = Eqs. ,
5838     name-pl-ab = eqs. ,
5839     refbounds-first-sg = {,(,)}, ,
5840     refbounds = {(,,)} ,
5841
5842 type = theorem ,
5843     Name-sg = Theorem ,
5844     name-sg = theorem ,
5845     Name-pl = Theorems ,
5846     name-pl = theorems ,
5847
5848 type = lemma ,
5849     Name-sg = Lemma ,
5850     name-sg = lemma ,
5851     Name-pl = Lemmas ,
5852     name-pl = lemmas ,
5853
5854 type = corollary ,
5855     Name-sg = Corollary ,
5856     name-sg = corollary ,
5857     Name-pl = Corollaries ,
5858     name-pl = corollaries ,
5859
5860 type = proposition ,
5861     Name-sg = Proposition ,
5862     name-sg = proposition ,
5863     Name-pl = Propositions ,
5864     name-pl = propositions ,
5865
5866 type = definition ,
5867     Name-sg = Definition ,
5868     name-sg = definition ,
5869     Name-pl = Definitions ,
5870     name-pl = definitions ,
5871
5872 type = proof ,
5873     Name-sg = Proof ,
5874     name-sg = proof ,
5875     Name-pl = Proofs ,
5876     name-pl = proofs ,
5877
5878 type = result ,
5879     Name-sg = Result ,
5880     name-sg = result ,
5881     Name-pl = Results ,

```

```

5882   name-pl = results ,
5883
5884   type = remark ,
5885     Name-sg = Remark ,
5886     name-sg = remark ,
5887     Name-pl = Remarks ,
5888     name-pl = remarks ,
5889
5890   type = example ,
5891     Name-sg = Example ,
5892     name-sg = example ,
5893     Name-pl = Examples ,
5894     name-pl = examples ,
5895
5896   type = algorithm ,
5897     Name-sg = Algorithm ,
5898     name-sg = algorithm ,
5899     Name-pl = Algorithms ,
5900     name-pl = algorithms ,
5901
5902   type = listing ,
5903     Name-sg = Listing ,
5904     name-sg = listing ,
5905     Name-pl = Listings ,
5906     name-pl = listings ,
5907
5908   type = exercise ,
5909     Name-sg = Exercise ,
5910     name-sg = exercise ,
5911     Name-pl = Exercises ,
5912     name-pl = exercises ,
5913
5914   type = solution ,
5915     Name-sg = Solution ,
5916     name-sg = solution ,
5917     Name-pl = Solutions ,
5918     name-pl = solutions ,
5919 </lang-english>

```

## 10.2 German

German language file has been initially provided by the author.

```

5920 <*package>
5921 \zcDeclareLanguage
5922 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5923 { german }
5924 \zcDeclareLanguageAlias { austrian } { german }
5925 \zcDeclareLanguageAlias { germanb } { german }
5926 \zcDeclareLanguageAlias { ngerman } { german }
5927 \zcDeclareLanguageAlias { naustrian } { german }
5928 \zcDeclareLanguageAlias { nswissgerman } { german }
5929 \zcDeclareLanguageAlias { swissgerman } { german }
5930 </package>

```

```

5931 ⟨*lang-german⟩
5932 namesep = {\nobreakspace} ,
5933 pairsep = {\~und\nobreakspace} ,
5934 listsep = {,~} ,
5935 lastsep = {\~und\nobreakspace} ,
5936 tpairsep = {\~und\nobreakspace} ,
5937 tlistsep = {,~} ,
5938 tlastsep = {\~und\nobreakspace} ,
5939 notesep = {~} ,
5940 rangesep = {\~bis\nobreakspace} ,
5941
5942 type = book ,
5943   gender = n ,
5944   case = N ,
5945     Name-sg = Buch ,
5946     Name-pl = Bücher ,
5947   case = A ,
5948     Name-sg = Buch ,
5949     Name-pl = Bücher ,
5950   case = D ,
5951     Name-sg = Buch ,
5952     Name-pl = Büchern ,
5953   case = G ,
5954     Name-sg = Buches ,
5955     Name-pl = Bücher ,
5956
5957 type = part ,
5958   gender = m ,
5959   case = N ,
5960     Name-sg = Teil ,
5961     Name-pl = Teile ,
5962   case = A ,
5963     Name-sg = Teil ,
5964     Name-pl = Teile ,
5965   case = D ,
5966     Name-sg = Teil ,
5967     Name-pl = Teilen ,
5968   case = G ,
5969     Name-sg = Teiles ,
5970     Name-pl = Teile ,
5971
5972 type = chapter ,
5973   gender = n ,
5974   case = N ,
5975     Name-sg = Kapitel ,
5976     Name-pl = Kapitel ,
5977   case = A ,
5978     Name-sg = Kapitel ,
5979     Name-pl = Kapitel ,
5980   case = D ,
5981     Name-sg = Kapitel ,
5982     Name-pl = Kapiteln ,
5983   case = G ,
5984     Name-sg = Kapitels ,

```

```

5985     Name-pl = Kapitel ,
5986
5987 type = section ,
5988     gender = m ,
5989     case = N ,
5990     Name-sg = Abschnitt ,
5991     Name-pl = Abschnitte ,
5992     case = A ,
5993     Name-sg = Abschnitt ,
5994     Name-pl = Abschnitte ,
5995     case = D ,
5996     Name-sg = Abschnitt ,
5997     Name-pl = Abschnitten ,
5998     case = G ,
5999     Name-sg = Abschnitts ,
6000     Name-pl = Abschnitte ,
6001
6002 type = paragraph ,
6003     gender = m ,
6004     case = N ,
6005     Name-sg = Absatz ,
6006     Name-pl = Absätze ,
6007     case = A ,
6008     Name-sg = Absatz ,
6009     Name-pl = Absätze ,
6010     case = D ,
6011     Name-sg = Absatz ,
6012     Name-pl = Absätzen ,
6013     case = G ,
6014     Name-sg = Absatzes ,
6015     Name-pl = Absätze ,
6016
6017 type = appendix ,
6018     gender = m ,
6019     case = N ,
6020     Name-sg = Anhang ,
6021     Name-pl = Anhänge ,
6022     case = A ,
6023     Name-sg = Anhang ,
6024     Name-pl = Anhänge ,
6025     case = D ,
6026     Name-sg = Anhang ,
6027     Name-pl = Anhängen ,
6028     case = G ,
6029     Name-sg = Anhangs ,
6030     Name-pl = Anhänge ,
6031
6032 type = page ,
6033     gender = f ,
6034     case = N ,
6035     Name-sg = Seite ,
6036     Name-pl = Seiten ,
6037     case = A ,
6038     Name-sg = Seite ,

```



```

6039     Name-pl = Seiten ,
6040     case = D ,
6041     Name-sg = Seite ,
6042     Name-pl = Seiten ,
6043     case = G ,
6044     Name-sg = Seite ,
6045     Name-pl = Seiten ,
6046     rangesep = {\textendash} ,
6047     rangetopair = false ,
6048
6049     type = line ,
6050     gender = f ,
6051     case = N ,
6052     Name-sg = Zeile ,
6053     Name-pl = Zeilen ,
6054     case = A ,
6055     Name-sg = Zeile ,
6056     Name-pl = Zeilen ,
6057     case = D ,
6058     Name-sg = Zeile ,
6059     Name-pl = Zeilen ,
6060     case = G ,
6061     Name-sg = Zeile ,
6062     Name-pl = Zeilen ,
6063
6064     type = figure ,
6065     gender = f ,
6066     case = N ,
6067     Name-sg = Abbildung ,
6068     Name-pl = Abbildungen ,
6069     Name-sg-ab = Abb. ,
6070     Name-pl-ab = Abb. ,
6071     case = A ,
6072     Name-sg = Abbildung ,
6073     Name-pl = Abbildungen ,
6074     Name-sg-ab = Abb. ,
6075     Name-pl-ab = Abb. ,
6076     case = D ,
6077     Name-sg = Abbildung ,
6078     Name-pl = Abbildungen ,
6079     Name-sg-ab = Abb. ,
6080     Name-pl-ab = Abb. ,
6081     case = G ,
6082     Name-sg = Abbildung ,
6083     Name-pl = Abbildungen ,
6084     Name-sg-ab = Abb. ,
6085     Name-pl-ab = Abb. ,
6086
6087     type = table ,
6088     gender = f ,
6089     case = N ,
6090     Name-sg = Tabelle ,
6091     Name-pl = Tabellen ,
6092     case = A ,

```

```

6093     Name-sg = Tabelle ,
6094     Name-pl = Tabellen ,
6095     case = D ,
6096     Name-sg = Tabelle ,
6097     Name-pl = Tabellen ,
6098     case = G ,
6099     Name-sg = Tabelle ,
6100     Name-pl = Tabellen ,
6101
6102 type = item ,
6103     gender = m ,
6104     case = N ,
6105     Name-sg = Punkt ,
6106     Name-pl = Punkte ,
6107     case = A ,
6108     Name-sg = Punkt ,
6109     Name-pl = Punkte ,
6110     case = D ,
6111     Name-sg = Punkt ,
6112     Name-pl = Punkten ,
6113     case = G ,
6114     Name-sg = Punktes ,
6115     Name-pl = Punkte ,
6116
6117 type = footnote ,
6118     gender = f ,
6119     case = N ,
6120     Name-sg = Fußnote ,
6121     Name-pl = Fußnoten ,
6122     case = A ,
6123     Name-sg = Fußnote ,
6124     Name-pl = Fußnoten ,
6125     case = D ,
6126     Name-sg = Fußnote ,
6127     Name-pl = Fußnoten ,
6128     case = G ,
6129     Name-sg = Fußnote ,
6130     Name-pl = Fußnoten ,
6131
6132 type = endnote ,
6133     gender = f ,
6134     case = N ,
6135     Name-sg = Endnote ,
6136     Name-pl = Endnoten ,
6137     case = A ,
6138     Name-sg = Endnote ,
6139     Name-pl = Endnoten ,
6140     case = D ,
6141     Name-sg = Endnote ,
6142     Name-pl = Endnoten ,
6143     case = G ,
6144     Name-sg = Endnote ,
6145     Name-pl = Endnoten ,
6146

```

```

6147 type = note ,
6148   gender = f ,
6149   case = N ,
6150     Name-sg = Anmerkung ,
6151     Name-pl = Anmerkungen ,
6152   case = A ,
6153     Name-sg = Anmerkung ,
6154     Name-pl = Anmerkungen ,
6155   case = D ,
6156     Name-sg = Anmerkung ,
6157     Name-pl = Anmerkungen ,
6158   case = G ,
6159     Name-sg = Anmerkung ,
6160     Name-pl = Anmerkungen ,
6161
6162 type = equation ,
6163   gender = f ,
6164   case = N ,
6165     Name-sg = Gleichung ,
6166     Name-pl = Gleichungen ,
6167   case = A ,
6168     Name-sg = Gleichung ,
6169     Name-pl = Gleichungen ,
6170   case = D ,
6171     Name-sg = Gleichung ,
6172     Name-pl = Gleichungen ,
6173   case = G ,
6174     Name-sg = Gleichung ,
6175     Name-pl = Gleichungen ,
6176   refbounds-first-sg = {,(,)}, ,
6177   refbounds = {(,,)} ,
6178
6179 type = theorem ,
6180   gender = n ,
6181   case = N ,
6182     Name-sg = Theorem ,
6183     Name-pl = Theoreme ,
6184   case = A ,
6185     Name-sg = Theorem ,
6186     Name-pl = Theoreme ,
6187   case = D ,
6188     Name-sg = Theorem ,
6189     Name-pl = Theoremen ,
6190   case = G ,
6191     Name-sg = Theorems ,
6192     Name-pl = Theoreme ,
6193
6194 type = lemma ,
6195   gender = n ,
6196   case = N ,
6197     Name-sg = Lemma ,
6198     Name-pl = Lemmata ,
6199   case = A ,
6200     Name-sg = Lemma ,

```

```

6201     Name-pl = Lemmata ,
6202     case = D ,
6203     Name-sg = Lemma ,
6204     Name-pl = Lemmata ,
6205     case = G ,
6206     Name-sg = Lemmas ,
6207     Name-pl = Lemmata ,
6208
6209 type = corollary ,
6210     gender = n ,
6211     case = N ,
6212     Name-sg = Korollar ,
6213     Name-pl = Korollare ,
6214     case = A ,
6215     Name-sg = Korollar ,
6216     Name-pl = Korollare ,
6217     case = D ,
6218     Name-sg = Korollar ,
6219     Name-pl = Korollaren ,
6220     case = G ,
6221     Name-sg = Korollars ,
6222     Name-pl = Korollare ,
6223
6224 type = proposition ,
6225     gender = m ,
6226     case = N ,
6227     Name-sg = Satz ,
6228     Name-pl = Sätze ,
6229     case = A ,
6230     Name-sg = Satz ,
6231     Name-pl = Sätze ,
6232     case = D ,
6233     Name-sg = Satz ,
6234     Name-pl = Sätzen ,
6235     case = G ,
6236     Name-sg = Satzes ,
6237     Name-pl = Sätze ,
6238
6239 type = definition ,
6240     gender = f ,
6241     case = N ,
6242     Name-sg = Definition ,
6243     Name-pl = Definitionen ,
6244     case = A ,
6245     Name-sg = Definition ,
6246     Name-pl = Definitionen ,
6247     case = D ,
6248     Name-sg = Definition ,
6249     Name-pl = Definitionen ,
6250     case = G ,
6251     Name-sg = Definition ,
6252     Name-pl = Definitionen ,
6253
6254 type = proof ,

```

```

6255 gender = m ,
6256 case = N ,
6257     Name-sg = Beweis ,
6258     Name-pl = Beweise ,
6259 case = A ,
6260     Name-sg = Beweis ,
6261     Name-pl = Beweise ,
6262 case = D ,
6263     Name-sg = Beweis ,
6264     Name-pl = Beweisen ,
6265 case = G ,
6266     Name-sg = Beweises ,
6267     Name-pl = Beweise ,
6268
6269 type = result ,
6270 gender = n ,
6271 case = N ,
6272     Name-sg = Ergebnis ,
6273     Name-pl = Ergebnisse ,
6274 case = A ,
6275     Name-sg = Ergebnis ,
6276     Name-pl = Ergebnisse ,
6277 case = D ,
6278     Name-sg = Ergebnis ,
6279     Name-pl = Ergebnissen ,
6280 case = G ,
6281     Name-sg = Ergebnisses ,
6282     Name-pl = Ergebnisse ,
6283
6284 type = remark ,
6285 gender = f ,
6286 case = N ,
6287     Name-sg = Bemerkung ,
6288     Name-pl = Bemerkungen ,
6289 case = A ,
6290     Name-sg = Bemerkung ,
6291     Name-pl = Bemerkungen ,
6292 case = D ,
6293     Name-sg = Bemerkung ,
6294     Name-pl = Bemerkungen ,
6295 case = G ,
6296     Name-sg = Bemerkung ,
6297     Name-pl = Bemerkungen ,
6298
6299 type = example ,
6300 gender = n ,
6301 case = N ,
6302     Name-sg = Beispiel ,
6303     Name-pl = Beispiele ,
6304 case = A ,
6305     Name-sg = Beispiel ,
6306     Name-pl = Beispiele ,
6307 case = D ,
6308     Name-sg = Beispiel ,

```

```

6309     Name-pl = Beispielen ,
6310     case = G ,
6311     Name-sg = Beispiels ,
6312     Name-pl = Beispiele ,
6313
6314 type = algorithm ,
6315     gender = m ,
6316     case = N ,
6317     Name-sg = Algorithmus ,
6318     Name-pl = Algorithmen ,
6319     case = A ,
6320     Name-sg = Algorithmus ,
6321     Name-pl = Algorithmen ,
6322     case = D ,
6323     Name-sg = Algorithmus ,
6324     Name-pl = Algorithmen ,
6325     case = G ,
6326     Name-sg = Algorithmus ,
6327     Name-pl = Algorithmen ,
6328
6329 type = listing ,
6330     gender = n ,
6331     case = N ,
6332     Name-sg = Listing ,
6333     Name-pl = Listings ,
6334     case = A ,
6335     Name-sg = Listing ,
6336     Name-pl = Listings ,
6337     case = D ,
6338     Name-sg = Listing ,
6339     Name-pl = Listings ,
6340     case = G ,
6341     Name-sg = Listings ,
6342     Name-pl = Listings ,
6343
6344 type = exercise ,
6345     gender = f ,
6346     case = N ,
6347     Name-sg = Übungsaufgabe ,
6348     Name-pl = Übungsaufgaben ,
6349     case = A ,
6350     Name-sg = Übungsaufgabe ,
6351     Name-pl = Übungsaufgaben ,
6352     case = D ,
6353     Name-sg = Übungsaufgabe ,
6354     Name-pl = Übungsaufgaben ,
6355     case = G ,
6356     Name-sg = Übungsaufgabe ,
6357     Name-pl = Übungsaufgaben ,
6358
6359 type = solution ,
6360     gender = f ,
6361     case = N ,
6362     Name-sg = Lösung ,

```

```

6363     Name-pl = Lösungen ,
6364     case = A ,
6365     Name-sg = Lösung ,
6366     Name-pl = Lösungen ,
6367     case = D ,
6368     Name-sg = Lösung ,
6369     Name-pl = Lösungen ,
6370     case = G ,
6371     Name-sg = Lösung ,
6372     Name-pl = Lösungen ,
6373 </lang-german>

```

### 10.3 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.

```

6374 <*package>
6375 \zcDeclareLanguage [ gender = { f , m } ] { french }
6376 \zcDeclareLanguageAlias { acadian } { french }
6377 \zcDeclareLanguageAlias { canadien } { french }
6378 \zcDeclareLanguageAlias { francais } { french }
6379 \zcDeclareLanguageAlias { frenchb } { french }
6380 </package>
6381 <*lang-french>
6382 namesep = {\nobreakspace} ,
6383 pairsep = {\~et\nobreakspace} ,
6384 listsep = { ,~ } ,
6385 lastsep = {\~et\nobreakspace} ,
6386 tpairsep = {\~et\nobreakspace} ,
6387 tlistsep = { ,~ } ,
6388 tlastsep = {\~et\nobreakspace} ,
6389 notesep = {~} ,
6390 rangesep = {\~à\nobreakspace} ,
6391
6392 type = book ,
6393     gender = m ,
6394     Name-sg = Livre ,
6395     name-sg = livre ,
6396     Name-pl = Livres ,
6397     name-pl = livres ,
6398
6399 type = part ,
6400     gender = f ,
6401     Name-sg = Partie ,
6402     name-sg = partie ,
6403     Name-pl = Parties ,
6404     name-pl = parties ,
6405
6406 type = chapter ,

```

```

6407     gender = m ,
6408     Name-sg = Chapitre ,
6409     name-sg = chapitre ,
6410     Name-pl = Chapitres ,
6411     name-pl = chapitres ,
6412
6413     type = section ,
6414     gender = f ,
6415     Name-sg = Section ,
6416     name-sg = section ,
6417     Name-pl = Sections ,
6418     name-pl = sections ,
6419
6420     type = paragraph ,
6421     gender = m ,
6422     Name-sg = Paragraphe ,
6423     name-sg = paragraphe ,
6424     Name-pl = Paragraphes ,
6425     name-pl = paragraphes ,
6426
6427     type = appendix ,
6428     gender = f ,
6429     Name-sg = Annexe ,
6430     name-sg = annexe ,
6431     Name-pl = Annexes ,
6432     name-pl = annexes ,
6433
6434     type = page ,
6435     gender = f ,
6436     Name-sg = Page ,
6437     name-sg = page ,
6438     Name-pl = Pages ,
6439     name-pl = pages ,
6440     rangesep = {-} ,
6441     rangetopair = false ,
6442
6443     type = line ,
6444     gender = f ,
6445     Name-sg = Ligne ,
6446     name-sg = ligne ,
6447     Name-pl = Lignes ,
6448     name-pl = lignes ,
6449
6450     type = figure ,
6451     gender = f ,
6452     Name-sg = Figure ,
6453     name-sg = figure ,
6454     Name-pl = Figures ,
6455     name-pl = figures ,
6456
6457     type = table ,
6458     gender = f ,
6459     Name-sg = Table ,
6460     name-sg = table ,

```



```

6461     Name-pl = Tables ,
6462     name-pl = tables ,
6463
6464     type = item ,
6465     gender = m ,
6466     Name-sg = Point ,
6467     name-sg = point ,
6468     Name-pl = Points ,
6469     name-pl = points ,
6470
6471     type = footnote ,
6472     gender = f ,
6473     Name-sg = Note ,
6474     name-sg = note ,
6475     Name-pl = Notes ,
6476     name-pl = notes ,
6477
6478     type = endnote ,
6479     gender = f ,
6480     Name-sg = Note ,
6481     name-sg = note ,
6482     Name-pl = Notes ,
6483     name-pl = notes ,
6484
6485     type = note ,
6486     gender = f ,
6487     Name-sg = Note ,
6488     name-sg = note ,
6489     Name-pl = Notes ,
6490     name-pl = notes ,
6491
6492     type = equation ,
6493     gender = f ,
6494     Name-sg = Équation ,
6495     name-sg = équation ,
6496     Name-pl = Équations ,
6497     name-pl = équations ,
6498     refbounds-first-sg = {,(,),} ,
6499     refbounds = {(,,)} ,
6500
6501     type = theorem ,
6502     gender = m ,
6503     Name-sg = Théorème ,
6504     name-sg = théorème ,
6505     Name-pl = Théorèmes ,
6506     name-pl = théorèmes ,
6507
6508     type = lemma ,
6509     gender = m ,
6510     Name-sg = Lemme ,
6511     name-sg = lemme ,
6512     Name-pl = Lemmes ,
6513     name-pl = lemmes ,
6514

```

```

6515 type = corollary ,
6516     gender = m ,
6517     Name-sg = Corollaire ,
6518     name-sg = corollaire ,
6519     Name-pl = Corollaires ,
6520     name-pl = corollaires ,
6521
6522 type = proposition ,
6523     gender = f ,
6524     Name-sg = Proposition ,
6525     name-sg = proposition ,
6526     Name-pl = Propositions ,
6527     name-pl = propositions ,
6528
6529 type = definition ,
6530     gender = f ,
6531     Name-sg = Définition ,
6532     name-sg = définition ,
6533     Name-pl = Définitions ,
6534     name-pl = définitions ,
6535
6536 type = proof ,
6537     gender = f ,
6538     Name-sg = Démonstration ,
6539     name-sg = démonstration ,
6540     Name-pl = Démonstrations ,
6541     name-pl = démonstrations ,
6542
6543 type = result ,
6544     gender = m ,
6545     Name-sg = Résultat ,
6546     name-sg = résultat ,
6547     Name-pl = Résultats ,
6548     name-pl = résultats ,
6549
6550 type = remark ,
6551     gender = f ,
6552     Name-sg = Remarque ,
6553     name-sg = remarque ,
6554     Name-pl = Remarques ,
6555     name-pl = remarques ,
6556
6557 type = example ,
6558     gender = m ,
6559     Name-sg = Exemple ,
6560     name-sg = exemple ,
6561     Name-pl = Exemples ,
6562     name-pl = exemples ,
6563
6564 type = algorithm ,
6565     gender = m ,
6566     Name-sg = Algorithme ,
6567     name-sg = algorithme ,
6568     Name-pl = Algorithmes ,

```

```

6569   name-pl = algorithmes ,
6570
6571   type = listing ,
6572   gender = m ,
6573   Name-sg = Listing ,
6574   name-sg = listing ,
6575   Name-pl = Listings ,
6576   name-pl = listings ,
6577
6578   type = exercise ,
6579   gender = m ,
6580   Name-sg = Exercice ,
6581   name-sg = exercice ,
6582   Name-pl = Exercices ,
6583   name-pl = exercices ,
6584
6585   type = solution ,
6586   gender = f ,
6587   Name-sg = Solution ,
6588   name-sg = solution ,
6589   Name-pl = Solutions ,
6590   name-pl = solutions ,
6591 </lang-french>

```

## 10.4 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.

```

6592 <*package>
6593 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6594 \zcDeclareLanguageAlias { brazilian } { portuguese }
6595 \zcDeclareLanguageAlias { brazil   } { portuguese }
6596 \zcDeclareLanguageAlias { portuges  } { portuguese }
6597 </package>
6598 <*lang-portuguese>
6599 namesep = {\nobreakspace} ,
6600 pairsep  = {\sim\nobreakspace} ,
6601 listsep  = { ,\sim } ,
6602 lastsep  = {\sim\nobreakspace} ,
6603 tpairsep = {\sim\nobreakspace} ,
6604 tlistsep = { ,\sim } ,
6605 tlastsep = {\sim\nobreakspace} ,
6606 notesep  = { \sim } ,
6607 rangesep = {\sim a\nobreakspace} ,
6608
6609 type = book ,
6610 gender = m ,
6611 Name-sg = Livro ,
6612 name-sg = livro ,
6613 Name-pl = Livros ,
6614 name-pl = livros ,

```

```

6615
6616 type = part ,
6617     gender = f ,
6618     Name-sg = Parte ,
6619     name-sg = parte ,
6620     Name-pl = Partes ,
6621     name-pl = partes ,
6622
6623 type = chapter ,
6624     gender = m ,
6625     Name-sg = Capítulo ,
6626     name-sg = capítulo ,
6627     Name-pl = Capítulos ,
6628     name-pl = capítulos ,
6629
6630 type = section ,
6631     gender = f ,
6632     Name-sg = Seção ,
6633     name-sg = seção ,
6634     Name-pl = Seções ,
6635     name-pl = seções ,
6636
6637 type = paragraph ,
6638     gender = m ,
6639     Name-sg = Parágrafo ,
6640     name-sg = parágrafo ,
6641     Name-pl = Parágrafos ,
6642     name-pl = parágrafos ,
6643     Name-sg-ab = Par. ,
6644     name-sg-ab = par. ,
6645     Name-pl-ab = Par. ,
6646     name-pl-ab = par. ,
6647
6648 type = appendix ,
6649     gender = m ,
6650     Name-sg = Apêndice ,
6651     name-sg = apêndice ,
6652     Name-pl = Apêndices ,
6653     name-pl = apêndices ,
6654
6655 type = page ,
6656     gender = f ,
6657     Name-sg = Página ,
6658     name-sg = página ,
6659     Name-pl = Páginas ,
6660     name-pl = páginas ,
6661     rangesep = {\textendash} ,
6662     rangetopair = false ,
6663
6664 type = line ,
6665     gender = f ,
6666     Name-sg = Linha ,
6667     name-sg = linha ,
6668     Name-pl = Linhas ,

```

```

6669     name-pl = linhas ,
6670
6671 type = figure ,
6672     gender = f ,
6673     Name-sg = Figura ,
6674     name-sg = figura ,
6675     Name-pl = Figuras ,
6676     name-pl = figuras ,
6677     Name-sg-ab = Fig. ,
6678     name-sg-ab = fig. ,
6679     Name-pl-ab = Figs. ,
6680     name-pl-ab = figs. ,
6681
6682 type = table ,
6683     gender = f ,
6684     Name-sg = Tabela ,
6685     name-sg = tabela ,
6686     Name-pl = Tabelas ,
6687     name-pl = tabelas ,
6688
6689 type = item ,
6690     gender = m ,
6691     Name-sg = Item ,
6692     name-sg = item ,
6693     Name-pl = Itens ,
6694     name-pl = itens ,
6695
6696 type = footnote ,
6697     gender = f ,
6698     Name-sg = Nota ,
6699     name-sg = nota ,
6700     Name-pl = Notas ,
6701     name-pl = notas ,
6702
6703 type = endnote ,
6704     gender = f ,
6705     Name-sg = Nota ,
6706     name-sg = nota ,
6707     Name-pl = Notas ,
6708     name-pl = notas ,
6709
6710 type = note ,
6711     gender = f ,
6712     Name-sg = Nota ,
6713     name-sg = nota ,
6714     Name-pl = Notas ,
6715     name-pl = notas ,
6716
6717 type = equation ,
6718     gender = f ,
6719     Name-sg = Equação ,
6720     name-sg = equação ,
6721     Name-pl = Equações ,
6722     name-pl = equações ,

```

```

6723 Name-sg-ab = Eq. ,
6724 name-sg-ab = eq. ,
6725 Name-pl-ab = Eqs. ,
6726 name-pl-ab = eqs. ,
6727 refbounds-first-sg = {(,),(,)},
6728 refbounds = {(,,)},
6729
6730 type = theorem ,
6731   gender = m ,
6732   Name-sg = Teorema ,
6733   name-sg = teorema ,
6734   Name-pl = Teoremas ,
6735   name-pl = teoremas ,
6736
6737 type = lemma ,
6738   gender = m ,
6739   Name-sg = Lema ,
6740   name-sg = lema ,
6741   Name-pl = Lemas ,
6742   name-pl = lemas ,
6743
6744 type = corollary ,
6745   gender = m ,
6746   Name-sg = Corolário ,
6747   name-sg = corolário ,
6748   Name-pl = Corolários ,
6749   name-pl = corolários ,
6750
6751 type = proposition ,
6752   gender = f ,
6753   Name-sg = Proposição ,
6754   name-sg = proposição ,
6755   Name-pl = Proposições ,
6756   name-pl = proposições ,
6757
6758 type = definition ,
6759   gender = f ,
6760   Name-sg = Definição ,
6761   name-sg = definição ,
6762   Name-pl = Definições ,
6763   name-pl = definições ,
6764
6765 type = proof ,
6766   gender = f ,
6767   Name-sg = Demonstração ,
6768   name-sg = demonstração ,
6769   Name-pl = Demonstrações ,
6770   name-pl = demonstrações ,
6771
6772 type = result ,
6773   gender = m ,
6774   Name-sg = Resultado ,
6775   name-sg = resultado ,
6776   Name-pl = Resultados ,

```

```

6777   name-pl = resultados ,
6778
6779   type = remark ,
6780   gender = f ,
6781   Name-sg = Observação ,
6782   name-sg = observação ,
6783   Name-pl = Observações ,
6784   name-pl = observações ,
6785
6786   type = example ,
6787   gender = m ,
6788   Name-sg = Exemplo ,
6789   name-sg = exemplo ,
6790   Name-pl = Exemplos ,
6791   name-pl = exemplos ,
6792
6793   type = algorithm ,
6794   gender = m ,
6795   Name-sg = Algoritmo ,
6796   name-sg = algoritmo ,
6797   Name-pl = Algoritmos ,
6798   name-pl = algoritmos ,
6799
6800   type = listing ,
6801   gender = f ,
6802   Name-sg = Listagem ,
6803   name-sg = listagem ,
6804   Name-pl = Listagens ,
6805   name-pl = listagens ,
6806
6807   type = exercise ,
6808   gender = m ,
6809   Name-sg = Exercício ,
6810   name-sg = exercício ,
6811   Name-pl = Exercícios ,
6812   name-pl = exercícios ,
6813
6814   type = solution ,
6815   gender = f ,
6816   Name-sg = Solução ,
6817   name-sg = solução ,
6818   Name-pl = Soluções ,
6819   name-pl = soluções ,
6820 </lang-portuguese>

```

## 10.5 Spanish

Spanish language file has been initially provided by the author.

```

6821 <*package>
6822 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6823 </package>
6824 <*lang-spanish>

```

```

6825 namesep = {\nobreakspace} ,
6826 pairsep = {\~y\nobreakspace} ,
6827 listsep = {,~} ,
6828 lastsep = {\~y\nobreakspace} ,
6829 tpairsep = {\~y\nobreakspace} ,
6830 tlistsep = {,~} ,
6831 tlastsep = {\~y\nobreakspace} ,
6832 notesep = {\~} ,
6833 rangesep = {\~a\nobreakspace} ,
6834
6835 type = book ,
6836 gender = m ,
6837 Name-sg = Libro ,
6838 name-sg = libro ,
6839 Name-pl = Libros ,
6840 name-pl = libros ,
6841
6842 type = part ,
6843 gender = f ,
6844 Name-sg = Parte ,
6845 name-sg = parte ,
6846 Name-pl = Partes ,
6847 name-pl = partes ,
6848
6849 type = chapter ,
6850 gender = m ,
6851 Name-sg = Capítulo ,
6852 name-sg = capítulo ,
6853 Name-pl = Capítulos ,
6854 name-pl = capítulos ,
6855
6856 type = section ,
6857 gender = f ,
6858 Name-sg = Sección ,
6859 name-sg = sección ,
6860 Name-pl = Secciones ,
6861 name-pl = secciones ,
6862
6863 type = paragraph ,
6864 gender = m ,
6865 Name-sg = Párrafo ,
6866 name-sg = párrafo ,
6867 Name-pl = Párrafos ,
6868 name-pl = párrafos ,
6869
6870 type = appendix ,
6871 gender = m ,
6872 Name-sg = Apéndice ,
6873 name-sg = apéndice ,
6874 Name-pl = Apéndices ,
6875 name-pl = apéndices ,
6876
6877 type = page ,
6878 gender = f ,

```



```

6879 Name-sg = Página ,
6880 name-sg = página ,
6881 Name-pl = Páginas ,
6882 name-pl = páginas ,
6883 rangesep = {\textendash} ,
6884 rangetopair = false ,
6885
6886 type = line ,
6887 gender = f ,
6888 Name-sg = Línea ,
6889 name-sg = línea ,
6890 Name-pl = Líneas ,
6891 name-pl = líneas ,
6892
6893 type = figure ,
6894 gender = f ,
6895 Name-sg = Figura ,
6896 name-sg = figura ,
6897 Name-pl = Figuras ,
6898 name-pl = figuras ,
6899
6900 type = table ,
6901 gender = m ,
6902 Name-sg = Cuadro ,
6903 name-sg = cuadro ,
6904 Name-pl = Cuadros ,
6905 name-pl = cuadros ,
6906
6907 type = item ,
6908 gender = m ,
6909 Name-sg = Punto ,
6910 name-sg = punto ,
6911 Name-pl = Puntos ,
6912 name-pl = puntos ,
6913
6914 type = footnote ,
6915 gender = f ,
6916 Name-sg = Nota ,
6917 name-sg = nota ,
6918 Name-pl = Notas ,
6919 name-pl = notas ,
6920
6921 type = endnote ,
6922 gender = f ,
6923 Name-sg = Nota ,
6924 name-sg = nota ,
6925 Name-pl = Notas ,
6926 name-pl = notas ,
6927
6928 type = note ,
6929 gender = f ,
6930 Name-sg = Nota ,
6931 name-sg = nota ,
6932 Name-pl = Notas ,

```

```

6933     name-pl = notas ,
6934
6935 type = equation ,
6936     gender = f ,
6937     Name-sg = Ecuación ,
6938     name-sg = ecuación ,
6939     Name-pl = Ecuaciones ,
6940     name-pl = ecuaciones ,
6941     refbounds-first-sg = {(,),(,)},
6942     refbounds = {(,,)},
6943
6944 type = theorem ,
6945     gender = m ,
6946     Name-sg = Teorema ,
6947     name-sg = teorema ,
6948     Name-pl = Teoremas ,
6949     name-pl = teoremas ,
6950
6951 type = lemma ,
6952     gender = m ,
6953     Name-sg = Lema ,
6954     name-sg = lema ,
6955     Name-pl = Lemas ,
6956     name-pl = lemas ,
6957
6958 type = corollary ,
6959     gender = m ,
6960     Name-sg = Corolario ,
6961     name-sg = corolario ,
6962     Name-pl = Corolarios ,
6963     name-pl = corolarios ,
6964
6965 type = proposition ,
6966     gender = f ,
6967     Name-sg = Proposición ,
6968     name-sg = proposición ,
6969     Name-pl = Propositiones ,
6970     name-pl = proposiciones ,
6971
6972 type = definition ,
6973     gender = f ,
6974     Name-sg = Definición ,
6975     name-sg = definición ,
6976     Name-pl = Definiciones ,
6977     name-pl = definiciones ,
6978
6979 type = proof ,
6980     gender = f ,
6981     Name-sg = Demostración ,
6982     name-sg = demostración ,
6983     Name-pl = Demostraciones ,
6984     name-pl = demostraciones ,
6985
6986 type = result ,

```

```

6987     gender = m ,
6988     Name-sg = Resultado ,
6989     name-sg = resultado ,
6990     Name-pl = Resultados ,
6991     name-pl = resultados ,
6992
6993     type = remark ,
6994     gender = f ,
6995     Name-sg = Observación ,
6996     name-sg = observación ,
6997     Name-pl = Observaciones ,
6998     name-pl = observaciones ,
6999
7000     type = example ,
7001     gender = m ,
7002     Name-sg = Ejemplo ,
7003     name-sg = ejemplo ,
7004     Name-pl = Ejemplos ,
7005     name-pl = ejemplos ,
7006
7007     type = algorithm ,
7008     gender = m ,
7009     Name-sg = Algoritmo ,
7010     name-sg = algoritmo ,
7011     Name-pl = Algoritmos ,
7012     name-pl = algoritmos ,
7013
7014     type = listing ,
7015     gender = m ,
7016     Name-sg = Listado ,
7017     name-sg = listado ,
7018     Name-pl = Listados ,
7019     name-pl = listados ,
7020
7021     type = exercise ,
7022     gender = m ,
7023     Name-sg = Ejercicio ,
7024     name-sg = ejercicio ,
7025     Name-pl = Ejercicios ,
7026     name-pl = ejercicios ,
7027
7028     type = solution ,
7029     gender = f ,
7030     Name-sg = Solución ,
7031     name-sg = solución ,
7032     Name-pl = Soluciones ,
7033     name-pl = soluciones ,
7034 </lang-spanish>

```

## 10.6 Dutch

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

```

7035 <*package>
7036 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7037 </package>
7038 <*lang-dutch>

7039 namesep   = {\nobreakspace} ,
7040 pairsep    = {\~en\nobreakspace} ,
7041 listsep    = {,~} ,
7042 lastsep    = {\~en\nobreakspace} ,
7043 tpairsep   = {\~en\nobreakspace} ,
7044 tlistsep   = {,~} ,
7045 tlastsep   = {\~en\nobreakspace} ,
7046 notesep    = {~} ,
7047 rangesep   = {\~t/m\nobreakspace} ,
7048
7049 type = book ,
7050   gender = n ,
7051   Name-sg = Boek ,
7052   name-sg = boek ,
7053   Name-pl = Boeken ,
7054   name-pl = boeken ,
7055
7056 type = part ,
7057   gender = n ,
7058   Name-sg = Deel ,
7059   name-sg = deel ,
7060   Name-pl = Delen ,
7061   name-pl = delen ,
7062
7063 type = chapter ,
7064   gender = n ,
7065   Name-sg = Hoofdstuk ,
7066   name-sg = hoofdstuk ,
7067   Name-pl = Hoofdstukken ,
7068   name-pl = hoofdstukken ,
7069
7070 type = section ,
7071   gender = m ,
7072   Name-sg = Paragraaf ,
7073   name-sg = paragraaf ,
7074   Name-pl = Paragrafen ,
7075   name-pl = paragrafen ,
7076
7077 type = paragraph ,
7078   gender = f ,
7079   Name-sg = Alinea ,
7080   name-sg = alinea ,
7081   Name-pl = Alinea's ,
7082   name-pl = alinea's ,
7083
7084 type = appendix ,
7085   gender = { m , n } ,
7086   Name-sg = Appendix ,
7087   name-sg = appendix ,

```

```

7088     Name-pl = Appendices ,
7089     name-pl = appendices ,
7090
7091     type = page ,
7092     gender = { f , m } ,
7093     Name-sg = Pagina ,
7094     name-sg = pagina ,
7095     Name-pl = Pagina's ,
7096     name-pl = pagina's ,
7097     rangesep = {\textendash} ,
7098     rangetopair = false ,
7099
7100     type = line ,
7101     gender = m ,
7102     Name-sg = Regel ,
7103     name-sg = regel ,
7104     Name-pl = Regels ,
7105     name-pl = regels ,
7106
7107     type = figure ,
7108     gender = { n , f , m } ,
7109     Name-sg = Figuur ,
7110     name-sg = figuur ,
7111     Name-pl = Figuren ,
7112     name-pl = figuren ,
7113
7114     type = table ,
7115     gender = { f , m } ,
7116     Name-sg = Tabel ,
7117     name-sg = tabel ,
7118     Name-pl = Tabellen ,
7119     name-pl = tabellen ,
7120
7121     type = item ,
7122     gender = n ,
7123     Name-sg = Punt ,
7124     name-sg = punt ,
7125     Name-pl = Punten ,
7126     name-pl = punten ,
7127
7128     type = footnote ,
7129     gender = { f , m } ,
7130     Name-sg = Voetnoot ,
7131     name-sg = voetnoot ,
7132     Name-pl = Voetnoten ,
7133     name-pl = voetnoten ,
7134
7135     type = endnote ,
7136     gender = { f , m } ,
7137     Name-sg = Eindnoot ,
7138     name-sg = eindnoot ,
7139     Name-pl = Eindnoten ,
7140     name-pl = eindnoten ,
7141

```

```

7142 type = note ,
7143     gender = f ,
7144     Name-sg = Opmerking ,
7145     name-sg = opmerking ,
7146     Name-pl = Opmerkingen ,
7147     name-pl = opmerkingen ,
7148
7149 type = equation ,
7150     gender = f ,
7151     Name-sg = Vergelijking ,
7152     name-sg = vergelijking ,
7153     Name-pl = Vergelijkingen ,
7154     name-pl = vergelijkingen ,
7155     Name-sg-ab = Vgl. ,
7156     name-sg-ab = vgl. ,
7157     Name-pl-ab = Vgl.'s ,
7158     name-pl-ab = vgl.'s ,
7159     refbounds-first-sg = {,(,),} ,
7160     refbounds = {(,,)} ,
7161
7162 type = theorem ,
7163     gender = f ,
7164     Name-sg = Stelling ,
7165     name-sg = stelling ,
7166     Name-pl = Stellingen ,
7167     name-pl = stellingen ,
7168

```

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”.

```

7169 type = lemma ,
7170     gender = n ,
7171     Name-sg = Lemma ,
7172     name-sg = lemma ,
7173     Name-pl = Lemma's ,
7174     name-pl = lemma's ,
7175
7176 type = corollary ,
7177     gender = n ,
7178     Name-sg = Gevolg ,
7179     name-sg = gevolg ,
7180     Name-pl = Gevolgen ,
7181     name-pl = gevolgen ,
7182
7183 type = proposition ,
7184     gender = f ,
7185     Name-sg = Propositie ,
7186     name-sg = propositie ,
7187     Name-pl = Proposities ,
7188     name-pl = proposities ,
7189
7190 type = definition ,
7191     gender = f ,

```

```

7192   Name-sg = Definitie ,
7193   name-sg = definitie ,
7194   Name-pl = Definities ,
7195   name-pl = definities ,
7196
7197   type = proof ,
7198   gender = n ,
7199   Name-sg = Bewijs ,
7200   name-sg = bewijs ,
7201   Name-pl = Bewijzen ,
7202   name-pl = bewijzen ,
7203
7204   type = result ,
7205   gender = n ,
7206   Name-sg = Resultaat ,
7207   name-sg = resultaat ,
7208   Name-pl = Resultaten ,
7209   name-pl = resultaten ,
7210
7211   type = remark ,
7212   gender = f ,
7213   Name-sg = Opmerking ,
7214   name-sg = opmerking ,
7215   Name-pl = Opmerkingen ,
7216   name-pl = opmerkingen ,
7217
7218   type = example ,
7219   gender = n ,
7220   Name-sg = Voorbeeld ,
7221   name-sg = voorbeeld ,
7222   Name-pl = Voorbeelden ,
7223   name-pl = voorbeelden ,
7224
7225   type = algorithm ,
7226   gender = { n , f , m } ,
7227   Name-sg = Algoritme ,
7228   name-sg = algoritme ,
7229   Name-pl = Algoritmes ,
7230   name-pl = algoritmes ,
7231

```

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

```

7232   type = listing ,
7233   gender = m ,
7234   Name-sg = Listing ,
7235   name-sg = listing ,
7236   Name-pl = Listings ,
7237   name-pl = listings ,
7238
7239   type = exercise ,
7240   gender = { f , m } ,
7241   Name-sg = Opgave ,
7242   name-sg = opgave ,

```

```

7243 Name-pl = Opgaven ,
7244 name-pl = opgaven ,
7245
7246 type = solution ,
7247 gender = f ,
7248 Name-sg = Oplossing ,
7249 name-sg = oplossing ,
7250 Name-pl = Oplossingen ,
7251 name-pl = oplossingen ,
7252 </lang-dutch>

```

## 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                  | 1876   |
| \AddToHook          | 100, 2022, 2066, 2089, 2120, 2122, 2160, 2233, 2249, 2262, 2286, 2430, 2443, 2451, 5305, 5341, 5358, 5360, 5365, 5410, 5412, 5414, 5416, 5418, 5420, 5422, 5424, 5428, 5432, 5434, 5439, 5449, 5458, 5464, 5466, 5492, 5503, 5537, 5590, 5615  |
| \alph               | 5518, 5606   |
| \appendix           | 2, 126, 129  |
| \appendixname       | 126  |
| \AtEndOfPackage     | 2441   |
| B                   |  |
| \babelname          | 2076   |
| \babelprovide       | 29, 54   |
| \bicaption          | 128  |
| \bionenumcaption    | 128  |
| \bitwonumcaption    | 128  |
| bool commands:      |  |
| \bool_case_true:    | 2  |
| \bool_gset_false:N  | 532  |
| \bool_gset_true:N   | 525, 2417  |
| \bool_if:NTF        | 376, 453, 491, 549, 1752, 1803, 2026, 2030, 2453, 3424, 3827, 3968, 4095, 4131, 4165, 4232, 4245, 4257, 4308, 4325, 4335, 4340, 4387, 4391, 4447, 4472, 4479, 4485, 4496, 4502, 4530, 4575, 4597, 4626, 4773, 4931, 4933, 5540, 5618   |
| \bool_if:nTF        | 68, 3538, 3548, 3572, 3589, 3604, 3669, 3677, 4318, 4692, 4734, 4820, 4837   |
| \bool_if_exist:N    | 330, 340, 364, 374, 424, 441, 451, 476, 479, 487, 489, 503, 506, 513, 516, 523, 530  |
| \bool_lazy_all:nTF  | 4415, 4943   |
| \bool_lazy_and:nnTF | 3395, 3416, 4199, 4270, 4649, 4916, 4958   |
| \bool_lazy_any:nTF  | 5095, 5104   |
| \bool_lazy_or:nnTF  | 1369, 1402, 1952, 2523, 2784, 3283, 3308, 3399, 4904   |
| \bool_new:N         | 331, 341, 366, 425, 443, 481, 504, 507, 514, 517, 524, 531, 791, 1566, 1567, 1594, 1618, 1970, 1977, 1984, 1997, 1998, 2168, 2169, 2170, 2171, 2172, 2279, 2280, 2410, 3432, 3447, 3709, 3710, 3721, 3722, 3729, 3731, 3732, 3745, 3746, 3747, 3759, 3760, 5502, 5549, 5589  |
| \bool_set:Nn        | 3392   |
| \bool_set_eq:NN     | 539  |
| \bool_set_false:N   | 365, 442, 478, 480, 515, 1579, 1583, 1755, 1853, 1898, 1940, 2005, 2014, 2015, 2032, 2039, 2180, 2184, 2191, 2199, 2200, 2201, 2304, 2316, 3437, 3530, 3776, 3777, 3794, 3833, 3844, 4244, 4437, 4438, 4445, 4446, 5102, 5119  |
| \bool_set_true:N    | 332, 342, 426, 505, 508, 518, 1573, 1574, 1578, 1584, 1777, 1799, 1862, 1864, 1902, 1904, 1913, 1915, 1944, 1946, 1959, 1961, 2004, 2009, 2010, 2178, 2185, 2190, 2207, 2209, 2211, 2214, 2215, 2216, 2292, 2297, 3544, 3554, 3558, 3580, 3595, 3610, 3634, 3802, 3828, 3834, 3838, 3845, 3991, 4002, 4013, 4063, 4099, 4135, 4169, 4186, 4267, 4301, 4474, 4534, 4580, 4602, 4631, 4892, 4899 |



|   |   |
|---|---|
| 5021, 5080, 5118, 5149, 5156, 5157,<br>5175, 5192, 5194, 5516, 5556, 5603 | \endsidecaption . . . . . 128               |
| \bool_until_do:Nn . . . . .   | exp commands:                               |
| . . . . . 1854, 1905, 1947, 3570, 3795                                    | \exp_args:NNe . . . . . 35, 38              |
| \bool_while_do:nn . . . . . 5650  | \exp_args:NNNo . . . . . 270                |
| \l_tmpa_bool . . . . . 1755, 1777,  | \exp_args:NNNV 1817, 1866, 1917, 1963       |
| 1799, 1803, 1853, 1854, 1862, 1864,                                       | \exp_args:NNNo . . . . . 270, 276           |
| 1898, 1902, 1904, 1905, 1913, 1915,                                       | \exp_args:No . . . . . 276                  |
| 1940, 1944, 1946, 1947, 1959, 1961  | \exp_args:Nx . . . . . 916, 5562, 5570      |
| <b>C</b>  | \exp_args:Nxx . . . . .                     |
| \caption . . . . . 131  | 1756, 1768, 1780, 1790, 1856, 1907,         |
| clist commands:   | 1949, 5135, 5141, 5163, 5167, 5182          |
| \clist_map_inline:nn . . . . .  | \exp_not:N . . . . . 112, 4342,             |
| . . . . . 624, 672, 689, 1008,  | 4345, 4356, 4359, 4362, 4698, 4702,         |
| 2202, 2358, 2419, 2920, 5519, 5608  | 4712, 4715, 4723, 4740, 4744, 4753,         |
| \contsubbottom . . . . . 128  | 4756, 4763, 4779, 4781, 4791, 4794,         |
| \contsubcaption . . . . . 128   | 4796, 4804, 4808, 4811, 4814, 4826,         |
| \counterwithin . . . . . 5  | 4829, 4843, 4848, 4868, 4871, 4880          |
| \crefstripprefix . . . . . 45   | \exp_not:n . . . . . 277, 4007, 4030, 4038, |
| cs commands:  | 4056, 4069, 4073, 4108, 4117, 4139,         |
| \cs_generate_variant:Nn . . . . . 65,                                     | 4147, 4154, 4178, 4192, 4209, 4219,         |
| 273, 279, 286, 297, 308, 319, 334,  | 4261, 4284, 4294, 4343, 4355, 4360,         |
| 344, 351, 358, 369, 393, 403, 428,  | 4361, 4518, 4541, 4553, 4587, 4609,         |
| 435, 446, 484, 510, 520, 527, 534,  | 4618, 4638, 4659, 4669, 4699, 4716,         |
| 962, 1836, 1872, 1923, 1969, 2579,  | 4741, 4749, 4757, 4760, 4792, 4793,         |
| 4728, 4767, 5132, 5236, 5269, 5302  | 4795, 4797, 4806, 4812, 4813, 4815,         |
| \cs_if_exist:NTF . . . . . 25,  | 4823, 4827, 4828, 4831, 4844, 4872          |
| 28, 46, 49, 58, 78, 5329, 5335, 5456                                      | \ExplSyntaxOn . . . . . 30, 922             |
| \cs_if_exist_p:N 4201, 4272, 4651, 5652                                   | <b>F</b>                                    |
| \cs_if_exist_use:N . . . . . 5427, 5690                                   | file commands:                              |
| \cs_new:Npn . . . . . 56, 66, 76, 87,                                     | \file_get:nnNTF . . . . . 916               |
| 274, 280, 282, 284, 287, 298, 309,  | \fmtversion . . . . . 3                     |
| 321, 323, 711, 4688, 4729, 4768, 5122                                     | \footnote . . . . . 2, 129, 130             |
| \cs_new_eq:NN . . . . . 5494, 5499  | <b>G</b>                                    |
| \cs_new_protected:Npn 268, 325, 335,                                      | group commands:                             |
| 345, 352, 359, 384, 394, 415, 417,  | \group_begin: . . . . . 102, 727, 908,      |
| 419, 429, 436, 474, 501, 511, 521,  | 1754, 1841, 1886, 1928, 2844, 3389,         |
| 528, 804, 906, 1515, 1534, 1738,  | 3403, 3435, 4342, 4359, 4698, 4715,         |
| 1837, 1882, 1924, 2449, 2577, 3387,                                       | 4740, 4756, 4779, 4791, 4796, 4811,         |
| 3451, 3493, 3504, 3517, 3647, 3699,                                       | 4826, 4843, 4871, 5415, 5419, 5423          |
| 3761, 3975, 4441, 4684, 4686, 4887,                                       | \group_end: . . . . . 105, 739, 960,        |
| 5133, 5204, 5237, 5270, 5400, 5557  | 1818, 1867, 1918, 1964, 2872, 3406,         |
| \cs_new_protected:Npx . . . . . 99  | 3429, 3439, 4356, 4362, 4712, 4723,         |
| \cs_set:Npn . . . . . 5403  | 4753, 4763, 4794, 4804, 4808, 4814,         |
| \cs_set_eq:NN . . . . .   | 4829, 4868, 4880, 5417, 5421, 5425          |
| . . . . . 103, 713, 5402, 5408, 5495, 5500                                | <b>I</b>                                    |
| \cs_set_nopar:Npn . . . . . 5485  | \IfBooleanT . . . . . 3436                  |
| \cs_to_str:N . . . . . 324  | \IfClassLoadedTF . . . . . 112              |
| <b>D</b>  | \ifdraft . . . . . 2183                     |
| \d . . . . . 1876   | \IfFormatAtLeastTF . . . . . 3, 4           |
| <b>E</b>  | \ifoptionfinal . . . . . 2189               |
| \endinput . . . . . 12  | \IfPackageAtLeastTF . . . . . 2290          |

|                    |   |                     |  |
|--------------------|---|---------------------|--|
| \IfPackageLoadedTF | 110   | keyval commands:    |  |
| \input             | 29, 30  | \keyval_parse:nnn   | 1520, 2329, 2384   |
| int commands:      |   | \KOMAClassName      | 5456, 5478   |
| \int_case:nnTF     | 3978, 4020, 4082, 4394, 4509, 4566  | L                   |  |
| \int_compare:nNnTF | 3581, 3596, 3611, 3623, 3635, 3655, 3657, 3701, 3875, 3998, 4026, 4048, 4103, 4173, 4379, 4381, 4458, 4488, 4547, 5145, 5151, 5171, 5177, 5668  | \label              | 128, 131, 132, 135, 5402, 5408   |
| \int_compare_p:nNn | 1372, 1405, 1953, 1954, 2525, 2786, 3286, 3311, 3671, 3679, 4419, 4908, 4919, 4948, 5115  | \labelformat        | 3  |
| \int_eval:n        | 99  | \language           | 24, 2070   |
| \int_incr:N        | 4434, 4466, 4478, 4480, 4495, 4497, 4501, 4503, 4515, 4538, 4550, 4584, 4606, 4615, 4635, 4682, 5666  | M                   |  |
| \int_new:N         | 3448, 3449, 3711, 3712, 3713, 3726, 3727  | \mainbabelname      | 24, 2077   |
| \int_rand:n        | 295, 306, 317   | \MessageBreak       | 10   |
| \int_set:Nn        | 3656, 3658, 3662, 3665, 5649  | MH commands:        |  |
| \int_to_roman:n    | 5653, 5660, 5661, 5664  | \MH_if_boolean:nTF  | 5554   |
| \int_use:N         | 47, 50, 54, 60  | msg commands:       |  |
| \int_zero:N        | 3649, 3650, 3771, 3772, 3773, 3774, 3775, 4432, 4433, 4435, 4436, 4677, 4678  | \msg_info:nnn       | 950, 1003, 1068, 1261, 1267, 1321, 5379, 5451, 5478, 5546, 5581, 5622, 5642, 5669  |
| \l_tmpa_int        | 5649, 5653, 5660, 5661, 5664, 5666, 5668  | \msg_info:nnnn      | 976, 983, 1013, 1385, 1419   |
| iow commands:      |   | \msg_info:nnnnn     | 997  |
| \iow_char:N        | 116, 131, 132, 137, 138, 143, 144, 149, 150, 202, 219   | \msg_line_context:  | 115, 121, 125, 127, 130, 136, 142, 148, 154, 159, 164, 169, 174, 180, 185, 188, 191, 196, 200, 207, 212, 217, 225, 229, 238, 243, 248, 252, 254, 256, 258, 265                               |
| \iow_newline:      | 265   | \msg_new:nnn        | 113, 119, 124, 126, 128, 134, 140, 146, 152, 157, 162, 167, 172, 177, 182, 187, 189, 194, 199, 201, 203, 205, 210, 215, 221, 223, 228, 230, 235, 241, 246, 251, 253, 255, 257, 259, 261, 263 |
| J                  |   | \msg_warning:nn     | 2031, 2037, 2254, 2320, 4421   |
| \jobname           | 5449  | \msg_warning:nnn    | 731, 752, 1547, 1554, 1710, 1716, 2109, 2146, 2158, 2220, 2231, 2268, 2309, 2386, 2436, 2446, 2593, 2707, 2713, 2871, 2915, 2961, 3153, 3159, 3240, 3859, 4239, 4937, 4953                   |
| K                  |   | \msg_warning:nnnn   | 820, 837, 871, 889, 2062, 2331, 2484, 2490, 2496, 2502, 2532, 2742, 2748, 2754, 2760, 2796, 2888, 2895, 2925, 3208, 3214, 3220, 3226, 3299, 3324, 3867, 5022, 5082                           |
| keys commands:     |   | \msg_warning:nnnnn  | 857, 896, 2909, 4972   |
| \l_keys_choice_tl  | 796   | \msg_warning:nnnnnn | 4979   |
| \keys_define:nn    | 20, 632, 678, 695, 756, 963, 1052, 1077, 1105, 1314, 1353, 1431, 1541, 1568, 1595, 1604, 1619, 1628, 1971, 1978, 1985, 1991, 1999, 2034, 2043, 2057, 2085, 2124, 2155, 2162, 2174, 2235, 2242, 2244, 2251, 2257, 2264, 2274, 2281, 2293, 2305, 2317, 2325, 2354, 2380, 2404, 2412, 2432, 2461, 2479, 2509, 2543, 2566, 2589, 2601, 2625, 2737, 2767, 2807, 2875, 2946, 2970, 2997, 3203, 3233, 3273, 3335 | N                   |  |
| \keys_set:nn       | 27, 30, 57, 58, 83, 736, 884, 947, 2298, 2578, 2583, 2869, 3390   | \newcounter         | 5, 5325, 5326  |
|                    |   | \NewDocumentCommand | 725, 742, 2575, 2580, 2842, 3385, 3433   |
|                    |   | \newfloat           | 129  |
|                    |   | \NewHook            | 1627   |

|                                     |   |                            |  |
|-------------------------------------|---|----------------------------|--|
| \newsfloat                          | 129                                     | seq commands:              |  |
| \nobreakspace                       | 1528,                                   | \seq_clear:N               | 440, 815,                              |
|                                     | 5726, 5727, 5729, 5730, 5732, 5734,     |                            | 852, 933, 946, 1007, 1615, 2520,       |
|                                     | 5932, 5933, 5935, 5936, 5938, 5940,     |                            | 2781, 2855, 2868, 2919, 3453, 5263     |
|                                     | 6382, 6383, 6385, 6386, 6388, 6390,     | \seq_const_from_clist:Nn   | 21                                     |
|                                     | 6599, 6600, 6602, 6603, 6605, 6607,     | \seq_count:N               |  |
|                                     | 6825, 6826, 6828, 6829, 6831, 6833,     |                            | 1373, 1387, 1406, 1421, 2525, 2534,    |
|                                     | 7039, 7040, 7042, 7043, 7045, 7047      |                            | 2786, 2798, 3287, 3301, 3312, 3326     |
| \NumCheckSetup                      | 45                                      | \seq_gclear:N              | 1366, 1399, 3280, 3305                 |
| \NumsCheckSetup                     | 45                                      | \seq_gconcat:NNN           | 617, 621                               |
| <b>P</b>                            |   |                            |  |
| \PackageError                       | 7                                       | \seq_get_left:NN           |  |
| \pagenote                           | 129, 130                                |                            | 830, 841, 937, 985, 2859, 2897, 3805   |
| \pagenumbering                      | 7                                       | \seq_gput_right:Nn         | 948, 954, 2423                         |
| \pageref                            | 84                                      | \seq_gremove_all:Nn        | 2455                                   |
| \PagesCheckSetup                    | 45                                      | \seq_gset_eq:NN            | 433, 1035                              |
| \paragraph                          | 61                                      | \seq_gset_from_clist:Nn    |  |
| prg commands:                       |   |                            | 560, 569, 581, 596, 604, 765, 780      |
| \prg_generate_conditional_-         |   | \seq_gset_split:Nnn        | 418                                    |
| variant:Nnn                         | 413,                                    | \seq_if_empty:NTF          | 816, 853, 934,                         |
|                                     | 461, 472, 499, 544, 552, 721, 1880      |                            | 974, 995, 2856, 2886, 2907, 3799, 4970 |
| \prg_new_conditional:Npnn           |   | \seq_if_exist:NTF          | 421, 431, 438, 449                     |
|                                     | 109, 111, 370, 447, 485, 546, 715       | \seq_if_in:NnTF            |  |
| \prg_new_protected_conditional:Npnn |   |                            | 834, 868, 912, 980, 1010, 2360,        |
|                                     | 404, 463, 535, 1873                     |                            | 2421, 2454, 2892, 2922, 3497, 4966     |
| \prg_return_false:                  |   | \seq_item:Nn               | 4700, 4706, 4709, 4711,                |
|                                     | 110, 112, 378, 382, 411, 455, 459,      |                            | 4717, 4718, 4721, 4722, 4742, 4748,    |
|                                     | 470, 493, 497, 542, 549, 550, 719, 1878 |                            | 4750, 4752, 4758, 4759, 4761, 4762,    |
| \prg_return_true:                   |   |                            | 4798, 4799, 4803, 4807, 4845, 4858,    |
|                                     | 110, 112, 377, 380, 409, 454,           |                            | 4863, 4866, 4873, 4874, 4878, 4879     |
|                                     | 457, 468, 492, 495, 540, 549, 718, 1877 | \seq_map_break:n           | 90, 3690, 3693                         |
| \prg_set_eq_conditional:NNn         | 723                                     | \seq_map_function:NN       | 3456                                   |
| \ProcessKeysOptions                 | 2574                                    | \seq_map_indexed_inline:Nn | 44, 3651                               |
| prop commands:                      |   | \seq_map_inline:Nn         | 1049, 1074,                            |
| \prop_if_in:NnTF                    | 35                                      |                            | 1311, 1350, 1428, 2445, 2458, 2506,    |
| \prop_if_in_p:Nn                    | 69                                      |                            | 2540, 2586, 2598, 2764, 2804, 2943,    |
| \prop_item:Nn                       | 38, 70                                  |                            | 2967, 3230, 3270, 3332, 3687, 5560     |
| \prop_new:N                         | 2324, 2379                              | \seq_map_tokens:Nn         | 72                                     |
| \prop_put:Nnn                       | 1538                                    | \seq_new:N                 | 422,                                   |
| \prop_remove:Nn                     | 1537                                    |                            | 432, 557, 558, 559, 568, 580, 595,     |
| \providecommand                     | 3                                       |                            | 603, 616, 620, 760, 775, 905, 1027,    |
| \ProvidesExplPackage                | 14                                      |                            | 1603, 2353, 2411, 3431, 3450, 3708,    |
| \ProvidesFile                       | 29                                      |                            | 3725, 3748, 3749, 3750, 3751, 3752,    |
|                                     |   |                            | 3753, 3754, 3755, 3756, 3757, 3758     |
| <b>R</b>                            |   |                            |  |
| \refstepcounter                     | 3, 128, 131–133, 135, 136               | \seq_pop_left:NN           | 3797                                   |
| regex commands:                     |   | \seq_put_right:Nn          |  |
| \regex_match:nnTF                   | 1876                                    |                            | 1011, 2362, 2923, 3500                 |
| \renewlist                          | 136, 137                                | \seq_reverse:N             | 1609                                   |
| \RequirePackage                     | 16,                                     | \seq_set_eq:NN             |  |
|                                     | 17, 18, 19, 20, 2027, 2246, 2259, 2283  |                            | 423, 467, 3763, 3989, 4000, 4011,      |
|                                     |   |                            | 4061, 4097, 4133, 4167, 4183, 4265,    |
|                                     |   |                            | 4299, 4310, 4532, 4577, 4599, 4628     |
| <b>S</b>                            |   |                            |  |
| \scantokens                         | 126                                     | \seq_set_from_clist:Nn     | 1608, 3391                             |
|                                     |   | \seq_set_split:Nnn         | 416                                    |
|                                     |   | \seq_sort:Nn               | 86, 3459                               |

|  |   |  |  |
|--|---|--|--|
| <code>\seq_use:Nn</code> .....                                   | 4984  | <code>\ifl@t@r</code> .....                | 3  |
| <code>\g_tmpa_seq</code> .....                                   | 1366, 1368,<br>1373, 1382, 1387, 1399, 1401, 1406,<br>1416, 1421, 3280, 3282, 3287, 3296,<br>3301, 3305, 3307, 3312, 3321, 3326 | <code>\@mem@scap@afterhook</code> .....    | 128  |
| <code>\l_tmpa_seq</code> .....                                   | 1007, 1011, 1043, 2520,<br>2522, 2525, 2529, 2534, 2781, 2783,<br>2786, 2793, 2798, 2919, 2923, 2938                            | <code>\@memsubcaption</code> .....         | 129  |
| <code>\setcounter</code> ..                                      | 5327, 5328, 5344, 5359, 5363  | <code>\@onlypreamble</code> .....          | 741, 755, 2874   |
| <code>\sidefootnote</code> .....                                 | 129, 130  | <code>\@raw@opt@{package}.sty</code> ..... | 58   |
| sort commands:   |   | <code>\bbl@loaded</code> .....             | 54   |
| <code>\sort_return_same:</code> .....                            | 87,<br>91, 3466, 3471, 3545, 3565, 3586,<br>3601, 3615, 3640, 3675, 3690, 3706  | <code>\bbl@main@language</code> .....      | 24, 2071   |
| <code>\sort_return_swapped:</code> .....                         | 87, 91, 3479, 3555, 3564, 3585,<br>3600, 3616, 3639, 3683, 3693, 3705   | <code>\c@lstnumber</code> .....            | 136  |
| <code>\stepcounter</code> .....                                  | 135, 5343, 5362   | <code>\c@page</code> .....                 | 7, 103   |
| str commands:  |   | <code>\caption@subtypehook</code> .....    | 5691   |
| <code>\str_case:nn</code> .....                                  | 45  | <code>\hyper@link</code> .....             | 112, 4345, 4702, 4744, 4781, 4848  |
| <code>\str_case:nnTF</code> .....                                | 1110,<br>1632, 2091, 2128, 2204, 2629, 3002   | <code>\lst@AddToHook</code> .....          | 5638, 5640   |
| <code>\str_compare:nNnTF</code> .....                            | 3561  | <code>\lst@Init</code> .....               | 136  |
| <code>\str_if_eq:nnTF</code> .....                               | 89, 4731  | <code>\lst@label</code> .....              | 5639   |
| <code>\str_if_eq_p:nn</code> .....                               | 5100, 5106, 5108, 5112  | <code>\lst@MakeCaption</code> .....        | 136  |
| <code>\str_new:N</code> .....                                    | 2042  | <code>\ltx@gobble</code> .....             | 131  |
| <code>\str_set:Nn</code> ...                                     | 2047, 2049, 2051, 2053  | <code>\ltx@label</code> ..                 | 132, 5494, 5495, 5499, 5500  |
| <code>\string</code> .....                                       | 5567, 5575  | <code>\m@mscaplabel</code> .....           | 128  |
| <code>\subbottom</code> .....                                    | 128   | <code>\MT@newlabel</code> .....            | 5567, 5575   |
| <code>\subcaption</code> .....                                   | 128   | <code>\protected@write</code> .....        | 5566, 5574   |
| <code>\subcaptionref</code> .....                                | 128   | <code>\zref@addprop</code> ..              | 22, 32, 43, 53, 55, 97, 108  |
| <code>\subref</code> .....                                       | 138   | <code>\zref@default</code> .....           | 112, 4685, 4687  |
| <code>\subsubsection</code> .....                                | 61  | <code>\zref@extractdefault</code> .....    | 10, 11, 121, 271, 277, 281   |
| <code>\subsubsection</code> .....                                | 61  | <code>\zref@ifpropundefined</code> ..      | 42, 1265,<br>1552, 1714, 2711, 3157, 5124, 5605                                      |
| <code>\subtop</code> .....                                       | 128   | <code>\zref@ifrefcontainsprop</code> ..    | 42, 45,<br>1742, 1750, 1809, 1827, 1839, 1884,<br>1926, 3862, 4690, 4775, 4833, 5127 |
|  |   | <code>\zref@ifrefundefined</code> .....    | 3461, 3463, 3475, 3830, 3832, 3837,<br>3854, 4236, 4247, 4449, 4770, 4889            |
|  |   | <code>\zref@label</code> .....             | 131, 5488  |
|  |   | <code>\zref@localaddprop</code> .....      | 5429, 5541, 5619, 5692   |
|  |   | <code>\ZREF@mainlist</code> .....          | 22, 32, 43,<br>53, 55, 97, 108, 5429, 5541, 5619, 5692                               |
| <code>\tag</code> .....  | 123, 133, 135   | <code>\zref@newprop</code> ....            | 5, 7, 21, 23, 33,<br>44, 54, 92, 107, 5426, 5518, 5606, 5689                         |
| TEX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands: |   | <code>\zref@refused</code> .....           | 3852   |
| <code>\@sidecaption</code> .....                                 | 128   | <code>\zref@wrapper@babel</code> ..        | 83, 131, 3386, 5488  |
| <code>\@Alph</code> .....  | 126   | <code>\textendash</code> .....             | 1532, 5781, 6046, 6661, 6883, 7097   |
| <code>\@addtoreset</code> .....                                  | 5   | <code>\textup</code> .....                 | 134  |
| <code>\@auxout</code> .....                                      | 5566, 5574  | <code>\thechapter</code> .....             | 126  |
| <code>\@bsphack</code> .....                                     | 909, 5559   | <code>\thelstnumber</code> .....           | 136  |
| <code>\@capttype</code> .....                                    | 131, 5427, 5460, 5461, 5465, 5690   | <code>\thepage</code> .....                | 7, 104   |
| <code>\@chapapp</code> .....                                     | 126   | <code>\thesection</code> .....             | 126  |
| <code>\@currentcounter</code> .....                              | 2, 3, 5, 62, 129, 131–133,<br>136, 28, 29, 49, 50, 2408, 5461, 5473   | tl commands:                               |  |
| <code>\@currentlabel</code> ..                                   | 3, 123, 129, 131, 136   | <code>\c_novalue_tl</code> ..              | 680, 681, 682, 683,<br>684, 685, 686, 2463, 2511, 2603, 2769                         |
| <code>\@currenvir</code> .....                                   | 5468  | <code>\tl_clear:N</code> .....             | 339, 363, 824,<br>862, 875, 892, 901, 926, 935, 968,                                 |
| <code>\@elt</code> .....   | 5   |  |  |
| <code>\@esphack</code> .....                                     | 959, 5579   |  |  |

2584, 2847, 2857, 2880, 3765, 3766,  
 3767, 3768, 3769, 3770, 3801, 4427,  
 4428, 4429, 4430, 4431, 4477, 4494,  
 4891, 4898, 4928, 5020, 5079, 5230  
 \tl\_const:Nn ..... 1517  
 \tl\_gclear:N ..... 356, 400, 5476  
 \tl\_gset:Nn ... 104, 349, 390, 734, 749  
 \tl\_gset\_eq:NN ..... 5465  
 \tl\_head:N .....  
 .. 3599, 3612, 3624, 3626, 3636, 3638  
 \tl\_head:n .... 1857, 1908, 1950, 1954  
 \tl\_if\_empty:NTF 80, 818, 828, 855,  
 866, 887, 894, 1001, 1057, 1082,  
 1114, 1149, 1186, 1223, 1271, 1319,  
 1325, 1358, 1436, 1474, 1740, 1861,  
 1912, 2913, 2951, 2975, 3006, 3041,  
 3078, 3115, 3163, 3238, 3244, 3278,  
 3340, 3361, 3407, 4234, 4830, 4913,  
 4935, 4993, 5004, 5047, 5470, 5639  
 \tl\_if\_empty:nTF .....  
 ..... 728, 744, 967, 1259, 1536,  
 1545, 1708, 2416, 2705, 2879, 3151  
 \tl\_if\_empty\_p:N ..... 4200, 4271,  
 4650, 4946, 4960, 5099, 5109, 5113  
 \tl\_if\_empty\_p:n ..... 1370, 1403,  
 2524, 2785, 3284, 3309, 3540, 3541,  
 3550, 3551, 3576, 3577, 3592, 3607  
 \tl\_if\_eq:NNTF ..... 3511, 3534, 3841  
 \tl\_if\_eq:NnTF .....  
 1766, 3454, 3486, 3661, 3664, 3689,  
 3692, 3809, 3857, 4895, 5139, 5468  
 \tl\_if\_eq:nnTF .. 1756, 1768, 1780,  
 1790, 1856, 1907, 1949, 3653, 5135,  
 5141, 5163, 5167, 5182, 5562, 5570  
 \tl\_if\_exist:NTF ..... 327, 337,  
 347, 354, 361, 372, 388, 398, 717, 5460  
 \tl\_if\_novalue:nTF .....  
 ..... 2466, 2514, 2606, 2772  
 \tl\_map\_break:n ..... 90  
 \tl\_map\_tokens:Nn ..... 82  
 \tl\_new:N ..... 98, 328, 338,  
 348, 355, 389, 399, 554, 555, 556,  
 706, 707, 708, 709, 733, 748, 1540,  
 2154, 2173, 2241, 2273, 2403, 3441,  
 3442, 3443, 3444, 3445, 3446, 3714,  
 3715, 3716, 3717, 3718, 3719, 3720,  
 3723, 3724, 3728, 3730, 3733, 3734,  
 3735, 3736, 3737, 3738, 3739, 3740,  
 3741, 3742, 3743, 3744, 5430, 5463  
 \tl\_put\_left:Nn . 4321, 4328, 4372,  
 5006, 5007, 5049, 5051, 5053, 5055  
 \tl\_put\_right:Nn . 4005, 4028, 4036,  
 4054, 4067, 4106, 4115, 4137, 4145,  
 4152, 4176, 4190, 4207, 4217, 4516,  
 4539, 4551, 4585, 4607, 4616, 4636,  
 4657, 4667, 4914, 4915, 4926, 5691  
 \tl\_reverse:N ..... 3521, 3524  
 \tl\_set:Nn .....  
 . 270, 329, 710, 735, 925, 969, 981,  
 1549, 1556, 1558, 1747, 1815, 1819,  
 1832, 1843, 1848, 1859, 1860, 1868,  
 1870, 1888, 1893, 1910, 1911, 1919,  
 1921, 1930, 1935, 1956, 1957, 1965,  
 1967, 2070, 2071, 2076, 2077, 2080,  
 2081, 2095, 2101, 2106, 2132, 2138,  
 2143, 2582, 2848, 2881, 2893, 3628,  
 3630, 3811, 3812, 3985, 3987, 4259,  
 4282, 4292, 4338, 4462, 4464, 4475,  
 4492, 4910, 4911, 4924, 5431, 5433  
 \tl\_set\_eq:NN .. 408, 4425, 5461, 5472  
 \tl\_show:N ..... 4388  
 \tl\_tail:N ..... 3629, 3631  
 \tl\_tail:n .....  
 .. 1859, 1860, 1910, 1911, 1956, 1957  
 \tl\_use:N ..... 292,  
 303, 314, 750, 914, 919, 949, 951, 955  
 \l\_tmpa\_tl ..... 923, 947, 1843,  
 1857, 1859, 1888, 1899, 1908, 1910,  
 1930, 1941, 1950, 1956, 3412, 3413  
 \l\_tmpb\_tl . 1805, 1812, 1815, 1819,  
 1848, 1857, 1860, 1861, 1868, 1893,  
 1901, 1908, 1911, 1912, 1919, 1935,  
 1943, 1950, 1953, 1954, 1957, 1965  

## U

 \upshape ..... 5545  
 use commands:  
 \use:N 26, 29, 796, 4203, 4278, 4653, 5296  
 \UseHook ..... 1842, 1887, 1929  

## V

 \value ..... 5344, 5363  
 \verbfootnote ..... 129, 130  

## Z

 \Z ..... 1876  
 \zcDeclareLanguage ..... 12, 25,  
 725, 5716, 5921, 6375, 6593, 6822, 7036  
 \zcDeclareLanguageAlias .....  
 ..... 25, 742, 5717, 5718,  
 5719, 5720, 5721, 5722, 5723, 5924,  
 5925, 5926, 5927, 5928, 5929, 6376,  
 6377, 6378, 6379, 6594, 6595, 6596  
 \zcLanguageSetup 20, 29, 30, 67, 72, 73, 2842  
 \zcpageref ..... 84, 3433  
 \zceref ..... 20, 58,  
 64, 66, 82–86, 92, 94, 134, 3385, 3438  
 \zcRefTypeSetup ..... 20, 67, 2580, 5544

`\zcsetup` . . . . . [20](#), [54](#), [58](#), [64](#), [66](#), [67](#), [2575](#)  
`\zlabel` [128](#), [131](#), [132](#), [135](#), [136](#), [5406](#), [5639](#)  
zrefcheck commands:  
  `\zrefcheck_zcref_beg_label:` . . . . . [3398](#)  
  `\zrefcheck_zcref_end_label_-`  
    maybe: . . . . . [3420](#)  
  `\zrefcheck_zcref_run_checks_on_-`  
    labels:n . . . . . [3421](#)  
zrefclever commands:  
  `\zrefclever_language_if_declared:nTF`  
    . . . . . [723](#)  
  `\zrefclever_language_varname:n` . . . . . [713](#)  
  `\l_zrefclever_ref_language_tl` . . . . . [709](#)  
zrefclever internal commands:  
  `\l__zrefclever_abbrev_bool` . . . . .  
    . . . . . [3733](#), [3920](#), [4917](#)  
  `\l_zrefclever_amsmath_subequations_-`  
    bool . . . . . [5502](#), [5516](#), [5540](#)  
  `\l__zrefclever_breqn_dgroup_bool`  
    . . . . . [5589](#), [5603](#), [5618](#)  
  `\l__zrefclever_cap_bool` . . . . .  
    . . . . . [3733](#), [3916](#), [4905](#)  
  `\l__zrefclever_capfirst_bool` . . . . .  
    . . . . . [1977](#), [1980](#), [4907](#)  
  `\__zrefclever_compat_module:nn` . . . . .  
    . . . . . [64](#),  
    [2449](#), [5303](#), [5321](#), [5382](#), [5454](#), [5481](#),  
    [5550](#), [5585](#), [5625](#), [5645](#), [5672](#), [5695](#)  
  `\__zrefclever_counter_reset_by:n`  
    . . . . . [6](#), [61](#), [62](#), [58](#), [60](#), [62](#), [66](#), [5510](#), [5597](#)  
  `\__zrefclever_counter_reset_by_-`  
    aux:nn . . . . . [73](#), [76](#)  
  `\__zrefclever_counter_reset_by_-`  
    aux:nnn . . . . . [83](#), [87](#)  
  `\l_zrefclever_counter_resetby_-`  
    prop . . . . . [5](#), [62](#), [69](#), [70](#), [2379](#), [2391](#)  
  `\l__zrefclever_counter_resettters_-`  
    seq . . . . . [5](#), [61](#), [62](#), [72](#), [2353](#), [2360](#), [2363](#)  
  `\l__zrefclever_counter_type_prop`  
    . . . . . [4](#), [60](#), [35](#), [38](#), [2324](#), [2336](#)  
  `\l_zrefclever_current_counter_-`  
    tl . . . . . [3](#), [5](#), [62](#), [21](#), [25](#),  
    [26](#), [36](#), [39](#), [41](#), [46](#), [47](#), [95](#), [2403](#), [2406](#)  
  `\l__zrefclever_current_language_-`  
    tl . . . . . [24](#),  
    [54](#), [707](#), [2070](#), [2076](#), [2080](#), [2096](#), [2133](#)  
  `\l_zrefclever_endrangefunc_tl` . . . . .  
    . . . . . [3733](#), [3908](#), [4200](#), [4201](#), [4203](#),  
    [4271](#), [4272](#), [4278](#), [4650](#), [4651](#), [4653](#)  
  `\l__zrefclever_endrangeprop_tl` . . . . .  
    . . . . . [47](#), [1740](#), [1750](#), [3733](#), [3912](#)  
  `\__zrefclever_extract:nnn` . . . . .  
    . . . . . [11](#), [280](#), [1845](#), [1850](#),  
    [1890](#), [1895](#), [1932](#), [1937](#), [3582](#), [3584](#),  
    [3597](#), [3614](#), [3702](#), [3704](#), [5146](#), [5148](#),  
    [5152](#), [5154](#), [5172](#), [5174](#), [5178](#), [5180](#)  
  `\__zrefclever_extract_default:Nnnn`  
    . . . . . [11](#), [268](#), [1744](#), [1805](#),  
    [1812](#), [1822](#), [1829](#), [3495](#), [3506](#), [3508](#),  
    [3519](#), [3522](#), [3525](#), [3527](#), [3815](#), [3818](#)  
  `\__zrefclever_extract_unexp:nnn` . . . . .  
    . . . . . [11](#), [121](#), [274](#), [1758](#), [1762](#), [1770](#),  
    [1774](#), [1782](#), [1786](#), [1792](#), [1796](#), [4351](#),  
    [4704](#), [4707](#), [4719](#), [4746](#), [4787](#), [4800](#),  
    [4854](#), [4860](#), [4875](#), [5125](#), [5128](#), [5129](#),  
    [5136](#), [5137](#), [5142](#), [5143](#), [5164](#), [5165](#),  
    [5168](#), [5169](#), [5184](#), [5188](#), [5563](#), [5571](#)  
  `\__zrefclever_extract_url_-`  
    unexp:n . . . . .  
    . . . . . [4347](#), [4703](#), [4745](#), [4783](#), [4850](#), [5122](#)  
  `\__zrefclever_get_enclosing_-`  
    counters\_value:n . . . . . [5](#), [6](#), [56](#), [61](#), [94](#)  
  `\__zrefclever_get_endrange_-`  
    pagecomp:nnN . . . . . [1882](#), [1923](#)  
  `\__zrefclever_get_endrange_-`  
    pagecomptwo:nnN . . . . . [1924](#), [1969](#)  
  `\__zrefclever_get_endrange_-`  
    property:nnN . . . . . [45](#), [1738](#), [1836](#)  
  `\__zrefclever_get_endrange_-`  
    stripprefix:nnN . . . . . [1837](#), [1872](#)  
  `\__zrefclever_get_ref:nnN` . . . . .  
    . . . . . [112](#), [113](#), [4008](#), [4031](#),  
    [4039](#), [4057](#), [4070](#), [4074](#), [4109](#), [4118](#),  
    [4140](#), [4148](#), [4155](#), [4179](#), [4193](#), [4220](#),  
    [4262](#), [4295](#), [4330](#), [4519](#), [4542](#), [4554](#),  
    [4588](#), [4610](#), [4619](#), [4639](#), [4670](#), [4688](#)  
  `\__zrefclever_get_ref_endrange:nnN`  
    . . . . . [45](#), [113](#), [4210](#), [4285](#), [4660](#), [4729](#)  
  `\__zrefclever_get_ref_first:` . . . . .  
    . . . . . [112](#), [113](#), [117](#), [4322](#), [4373](#), [4768](#)  
  `\__zrefclever_get_rf_opt_bool:nnN` [125](#)  
  `\__zrefclever_get_rf_opt_-`  
    bool:nnnnN [20](#), [3913](#), [3917](#), [3921](#), [5270](#)  
  `\__zrefclever_get_rf_opt_-`  
    seq:nnnN . . . . . [20](#), [124](#),  
    [3925](#), [3929](#), [3933](#), [3937](#), [3941](#), [3945](#),  
    [3949](#), [3953](#), [3957](#), [3961](#), [4962](#), [5237](#)  
  `\__zrefclever_get_rf_opt_tl:nnnN`  
    . . . . . [20](#), [22](#), [45](#), [123](#), [3409](#), [3780](#),  
    [3784](#), [3788](#), [3877](#), [3881](#), [3885](#), [3889](#),  
    [3893](#), [3897](#), [3901](#), [3905](#), [3909](#), [5204](#)  
  `\l__zrefclever_hyperlink_bool` . . . . .  
    . . . . . [1997](#), [2004](#), [2009](#), [2014](#), [2026](#), [2032](#),  
    [2039](#), [3437](#), [4694](#), [4736](#), [4839](#), [5097](#)  
  `\l_zrefclever_hyperref_warn_-`  
    bool . . . . . [1998](#), [2005](#), [2010](#), [2015](#), [2030](#)  
  `\__zrefclever_if_class_loaded:n` [109](#)

```

\__zrefclever_if_class_loaded:nTF
..... 5384
\__zrefclever_if_package_-
loaded:n ..... 109
\__zrefclever_if_package_-
loaded:nTF ..... 2024,
2068, 2074, 2288, 5323, 5483, 5490,
5552, 5587, 5627, 5647, 5674, 5697
\__zrefclever_is_integer_rgx:n 1873
\__zrefclever_is_integer_rgx:nTF
..... 1899, 1901, 1941, 1943
\g__zrefclever_koma_captionofbeside_-
capttype_tl ..... 5463
\g__zrefclever_koma_capttype_tl ..
..... 5465, 5470, 5473, 5476
\l__zrefclever_label_a_tl .....
. 91, 3714, 3798, 3817, 3830, 3852,
3854, 3860, 3863, 3869, 3986, 4008,
4031, 4039, 4074, 4140, 4155, 4205,
4211, 4220, 4252, 4262, 4280, 4286,
4295, 4449, 4453, 4463, 4476, 4493,
4519, 4555, 4619, 4655, 4661, 4670
\l__zrefclever_label_b_tl .....
..... 91, 3714,
3801, 3806, 3820, 3832, 3837, 4453
\l__zrefclever_label_count_int ..
..... 92, 3711, 3771,
3875, 3978, 4432, 4458, 4682, 4949
\l__zrefclever_label_enclval_a_-
tl ..... 3441, 3519, 3521, 3576,
3592, 3612, 3624, 3628, 3629, 3636
\l__zrefclever_label_enclval_b_-
tl ..... 3441, 3522, 3524, 3577,
3599, 3607, 3626, 3630, 3631, 3638
\l__zrefclever_label_extdoc_a_tl
.. 3441, 3525, 3535, 3540, 3550, 3563
\l__zrefclever_label_extdoc_b_tl
.. 3441, 3527, 3536, 3541, 3551, 3562
\l__zrefclever_label_type_a_tl ..
..... 3410, 3441, 3496, 3498,
3501, 3507, 3512, 3661, 3689, 3781,
3785, 3789, 3811, 3816, 3842, 3857,
3878, 3882, 3886, 3890, 3894, 3898,
3902, 3906, 3910, 3914, 3918, 3922,
3926, 3930, 3934, 3938, 3942, 3946,
3950, 3954, 3958, 3962, 3988, 4465
\l__zrefclever_label_type_b_tl ..
..... 3441, 3509,
3513, 3664, 3692, 3812, 3819, 3843
\__zrefclever_label_type_put_-
new_right:n .... 85, 86, 3457, 3493
\l__zrefclever_label_types_seq ..
.... 86, 3450, 3453, 3497, 3500, 3687

\__zrefclever_labels_in_sequence:nn
..... 47, 92, 122, 4250, 4452, 5133
\l__zrefclever_lang_decl_case_tl
554, 935, 938, 981, 986, 1325, 1342,
2857, 2860, 2893, 2898, 3244, 3261
\l__zrefclever_lang_declension_-
seq ..... 554,
814, 815, 816, 830, 834, 841, 932,
933, 934, 937, 974, 980, 985, 2854,
2855, 2856, 2859, 2886, 2892, 2897
\l__zrefclever_lang_gender_seq ..
.... 554, 851, 852, 853, 868, 945,
946, 995, 1010, 2867, 2868, 2907, 2922
\__zrefclever_language_if_-
declared:n ..... 25, 724
\__zrefclever_language_if_-
declared:n(TF) ..... 24
\__zrefclever_language_if_-
declared:nTF 289, 300, 311, 715,
730, 746, 806, 910, 2107, 2144, 2845
\__zrefclever_language_varname:n
..... 24, 292,
303, 314, 711, 714, 717, 733, 734,
748, 749, 750, 914, 919, 949, 951, 955
\l__zrefclever_last_of_type_bool
..... 92, 3708, 3828,
3833, 3834, 3838, 3844, 3845, 3968
\l__zrefclever_lastsep_tl . 3733,
3892, 4038, 4073, 4117, 4154, 4192
\l__zrefclever_link_star_bool ...
.. 3392, 3431, 4695, 4737, 4840, 5098
\l__zrefclever_listsep_tl .....
... 3733, 3888, 4069, 4147, 4518,
4541, 4553, 4587, 4609, 4618, 4638
\g__zrefclever_loaded_langfiles_-
seq ..... 905, 913, 948, 954
\__zrefclever_ltxlabel:n .....
..... 132, 5485, 5495, 5500
\l__zrefclever_main_language_tl .
..... 24,
54, 708, 2071, 2077, 2081, 2102, 2139
\__zrefclever_mathtools_showonlyrefs:n
..... 3426, 5557
\l__zrefclever_mathtools_-
showonlyrefs_bool 3424, 5549, 5556
\__zrefclever_memoir_both_-
labels: .....
.. 5400, 5411, 5413, 5415, 5419, 5423
\l__zrefclever_memoir_footnote_-
type_tl .... 5430, 5431, 5433, 5437
\__zrefclever_memoir_label_and_-
zlabel:n ..... 5403, 5408
\__zrefclever_memoir_orig_-
label:n ..... 5402, 5405

```



```

\__zrefclever_name_default: ....
..... 4684, 4822
\l__zrefclever_name_format_-
  fallback_tl ..... 3720, 4924,
    4928, 4993, 5042, 5054, 5056, 5074
\l__zrefclever_name_format_tl ...
... 3720, 4910, 4911, 4914, 4915,
    4925, 4926, 4999, 5006, 5007, 5015,
    5023, 5033, 5050, 5051, 5064, 5084
\l__zrefclever_name_in_link_bool
..... 114,
    117, 3720, 4340, 4773, 5102, 5118, 5119
\l__zrefclever_namefont_tl 3733,
    3900, 4343, 4360, 4792, 4812, 4827
\l__zrefclever_nameinlink_str ...
..... 2042, 2047, 2049,
    2051, 2053, 5100, 5106, 5108, 5112
\l__zrefclever_namesep_tl .....
.. 3733, 3880, 4795, 4815, 4823, 4831
\l__zrefclever_next_is_same_bool
..... 92, 122, 3726,
    4446, 4479, 4496, 4502, 5157, 5195
\l__zrefclever_next_maybe_range_-
  bool .....
. 92, 122, 3726, 4244, 4257, 4445,
    4472, 4485, 5149, 5156, 5175, 5193
\l__zrefclever_noabbrev_first_-
  bool ..... 1984, 1987, 4921
\g__zrefclever_nocompat_bool ...
..... 2410, 2417, 2453
\l__zrefclever_nocompat_bool ... 63
\g__zrefclever_nocompat_modules_-
  seq 2411, 2421, 2424, 2445, 2454, 2455
\l__zrefclever_nocompat_modules_-
  seq ..... 63
\l__zrefclever_nudge_comptosing_-
  bool ... 2170, 2200, 2209, 2215, 4945
\l__zrefclever_nudge_enabled_-
  bool ..... 2168, 2178, 2180,
    2184, 2185, 2190, 2191, 4417, 4931
\l__zrefclever_nudge_gender_bool
..... 2172, 2201, 2211, 2216, 4959
\l__zrefclever_nudge_multitype_-
  bool ... 2169, 2199, 2207, 2214, 4418
\l__zrefclever_nudge_singular_-
  bool ..... 2171, 2227, 4933
\__zrefclever_opt_bool_get:NN(TF)
..... 19
\__zrefclever_opt_bool_get:NNTF .
... 535, 5273, 5278, 5283, 5288, 5293
\__zrefclever_opt_bool_gset_-
  false:N .....
..... 18, 501, 1483, 1500, 3363, 3371
\__zrefclever_opt_bool_gset_-
  true:N 18, 501, 1445, 1462, 3342, 3350
\__zrefclever_opt_bool_if:N(TF) . 19
\__zrefclever_opt_bool_if:NTF ...
..... 546, 879
\__zrefclever_opt_bool_if_-
  set:N(TF) ..... 18
\__zrefclever_opt_bool_if_-
  set:NTF ..... 485,
    537, 548, 1438, 1454, 1476, 1492
\__zrefclever_opt_bool_set_-
  false:N ..... 18, 501, 2553, 2821
\__zrefclever_opt_bool_set_-
  true:N ..... 18, 501, 2548, 2812
\__zrefclever_opt_bool_unset:N ..
..... 17, 474, 2558, 2830
\__zrefclever_opt_seq_get:NN(TF) 17
\__zrefclever_opt_seq_get:NNTF ..
... 463, 809, 846, 927, 940, 2849,
    2862, 5240, 5245, 5250, 5255, 5260
\__zrefclever_opt_seq_gset_-
  clist_split:Nn .....
..... 16, 415, 1367, 1400, 3281, 3306
\__zrefclever_opt_seq_gset_eq:NN
    16, 415, 1376, 1409, 2930, 3290, 3315
\__zrefclever_opt_seq_if_-
  set:N(TF) ..... 17
\__zrefclever_opt_seq_if_set:NTF
..... 447, 465, 1018, 1360, 1392
\__zrefclever_opt_seq_set_clist_-
  split:Nn ..... 16, 415, 2521, 2782
\__zrefclever_opt_seq_set_eq:NN .
..... 16, 415, 2527, 2788
\__zrefclever_opt_seq_unset:N ...
..... 16, 436, 2516, 2774
\__zrefclever_opt_tl_clear:N ...
.. 14, 325, 1636, 1641, 1656, 1671,
    1686, 2633, 2638, 2653, 2668, 2683
\__zrefclever_opt_tl_cset_-
  fallback:nn ..... 1515, 1522
\__zrefclever_opt_tl_gclear:N ...
.. 14, 325, 3008, 3014, 3022, 3029,
    3050, 3066, 3087, 3103, 3124, 3140
\__zrefclever_opt_tl_gclear_if_-
  new:N .....
.. 15, 384, 1116, 1122, 1130, 1137,
    1158, 1174, 1195, 1211, 1232, 1248
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF ...
    404, 4995, 5010, 5029, 5038, 5059,
    5069, 5207, 5212, 5217, 5222, 5227
\__zrefclever_opt_tl_gset:N .... 14
\__zrefclever_opt_tl_gset:Nn ...
..... 325, 2953, 2977, 2985,

```



3043, 3058, 3080, 3095, 3117, 3132,  
 3165, 3172, 3181, 3189, 3246, 3256  
 \\_zrefclever\_opt\_tl\_gset\_if\_-  
   new:Nn 15, 384, 1059, 1084, 1093,  
   1151, 1166, 1188, 1203, 1225, 1240,  
   1273, 1280, 1289, 1297, 1327, 1337  
 \\_zrefclever\_opt\_tl\_if\_set:N(TF)  
   ..... 15  
 \\_zrefclever\_opt\_tl\_if\_set:NTF .  
   ..... 370, 386, 396, 406  
 \\_zrefclever\_opt\_tl\_set:N ..... 14  
 \\_zrefclever\_opt\_tl\_set:Nn 325,  
   1650, 1665, 1680, 1720, 1726, 2472,  
   2615, 2647, 2662, 2677, 2717, 2724  
 \\_zrefclever\_opt\_tl\_unset:N 14,  
   359, 1695, 1700, 2468, 2608, 2692, 2697  
 \\_zrefclever\_opt\_var\_set\_bool:n  
   ..... 13, 14, 323, 330,  
   331, 332, 340, 341, 342, 364, 365,  
   366, 374, 376, 424, 425, 426, 441,  
   442, 443, 451, 453, 479, 480, 481,  
   489, 491, 506, 507, 508, 516, 517, 518  
 \\_zrefclever\_opt\_varname\_-  
   fallback:nn .....  
   ..... 13, 321, 1518, 5228, 5261, 5294  
 \\_zrefclever\_opt\_varname\_-  
   general:nn ..... 12,  
   282, 1638, 1643, 1652, 1658, 1667,  
   1673, 1682, 1688, 1697, 1702, 1722,  
   1728, 2469, 2473, 2517, 2528,  
   2549, 2554, 2559, 5208, 5241, 5274  
 \\_zrefclever\_opt\_varname\_lang\_-  
   default:nnn ..... 12, 298,  
   1061, 1086, 1118, 1124, 1153, 1160,  
   1190, 1197, 1227, 1234, 1275, 1282,  
   1362, 1378, 1440, 1447, 1478, 1485,  
   2955, 2979, 3010, 3016, 3045, 3052,  
   3082, 3089, 3119, 3126, 3167, 3174,  
   3292, 3344, 3365, 5223, 5256, 5289  
 \\_zrefclever\_opt\_varname\_lang\_-  
   type:nnnn .....  
   ..... 13, 309, 1020, 1029, 1037,  
   1095, 1132, 1139, 1168, 1176, 1205,  
   1213, 1242, 1250, 1291, 1299, 1329,  
   1339, 1394, 1411, 1456, 1464, 1494,  
   1502, 2932, 2987, 3024, 3031, 3060,  
   3068, 3097, 3105, 3134, 3142, 3183,  
   3191, 3248, 3258, 3317, 3352, 3373,  
   5012, 5061, 5071, 5218, 5251, 5284  
 \\_zrefclever\_opt\_varname\_-  
   language:nnn ..... 12,  
   287, 762, 767, 777, 782, 793, 798,  
   811, 848, 881, 929, 942, 2851, 2864  
 \\_zrefclever\_opt\_varname\_-  
   type:nnn ..... 12, 284, 2610,  
   2617, 2635, 2640, 2649, 2655, 2664,  
   2670, 2679, 2685, 2694, 2699, 2719,  
   2726, 2776, 2790, 2814, 2823, 2832,  
   4997, 5031, 5040, 5213, 5246, 5279  
 \\_zrefclever\_orig\_ltxlabel:n ...  
   ..... 5487, 5494, 5499  
 \\_zrefclever\_page\_format\_aux: ..  
   ..... 99, 103  
 \g\_zrefclever\_page\_format\_tl ...  
   ..... 7, 98, 104, 107  
 \l\_zrefclever\_pairsep\_tl .....  
   ..... 3733, 3884, 4007,  
   4030, 4056, 4108, 4139, 4178, 4261  
 \\_zrefclever\_process\_language\_-  
   settings: ..... 57, 58, 804, 3394  
 \\_zrefclever\_prop\_put\_non\_-  
   empty:Nnn .... 42, 1534, 2335, 2390  
 \\_zrefclever\_provide\_langfile:n  
   ..... 20, 30, 31, 83, 906, 2113, 3393  
 \l\_zrefclever\_range\_beg\_is\_-  
   first\_bool .....  
   ... 3726, 3776, 4095, 4131, 4165,  
   4437, 4474, 4530, 4575, 4597, 4626  
 \l\_zrefclever\_range\_beg\_label\_-  
   tl ..... 92,  
   3726, 3769, 4058, 4071, 4110, 4119,  
   4149, 4180, 4194, 4204, 4430, 4475,  
   4492, 4543, 4589, 4611, 4640, 4654  
 \l\_zrefclever\_range\_count\_int ..  
   ..... 92,  
   3726, 3774, 4020, 4084, 4435, 4478,  
   4489, 4495, 4501, 4509, 4568, 4677  
 \l\_zrefclever\_range\_end\_ref\_tl .  
   ... 3726, 3770, 4206, 4212, 4281,  
   4287, 4431, 4477, 4494, 4656, 4662  
 \l\_zrefclever\_range\_same\_count\_-  
   int ..... 92,  
   3726, 3775, 3998, 4049, 4085, 4436,  
   4480, 4497, 4503, 4548, 4569, 4678  
 \l\_zrefclever\_rangesep\_tl .....  
   ..... 3733, 3896,  
   4209, 4219, 4284, 4294, 4659, 4669  
 \l\_zrefclever\_rangetopair\_bool .  
   ..... 3733, 3924, 4245  
 \l\_zrefclever\_ref\_count\_int ...  
   ..... 3711, 3773,  
   4026, 4104, 4174, 4433, 4466, 4515,  
   4538, 4550, 4584, 4606, 4615, 4635  
 \l\_zrefclever\_ref\_decl\_case\_tl .  
   ..... 27, 818, 823, 824, 828, 831,  
   835, 839, 842, 887, 890, 892, 2154,  
   2164, 5004, 5008, 5047, 5052, 5057

```

\__zrefclever_ref_default: .....
.. 4684, 4726, 4732, 4771, 4816, 4883
\l__zrefclever_ref_gender_tl ...
..... 28, 855, 861,
862, 866, 869, 874, 875, 894, 900,
901, 2173, 2237, 4960, 4968, 4974, 4982
\l__zrefclever_ref_language_tl ..
..... 24, 27, 54, 706, 710, 807,
812, 822, 840, 849, 859, 873, 882,
891, 898, 2095, 2101, 2106, 2114,
2132, 2138, 2143, 3393, 3411, 3782,
3786, 3790, 3879, 3883, 3887, 3891,
3895, 3899, 3903, 3907, 3911, 3915,
3919, 3923, 3927, 3931, 3935, 3939,
3943, 3947, 3951, 3955, 3959, 3963,
4964, 4976, 4987, 5013, 5062, 5072
\l__zrefclever_ref_property_tl ..
..... 42, 47, 1540,
1549, 1556, 1558, 1742, 1766, 1810,
1827, 1839, 1846, 1851, 1884, 1891,
1896, 1926, 1933, 1938, 3486, 3809,
3864, 3868, 4690, 4777, 4835, 5139
\l__zrefclever_ref_propserity_tl 3454
\l__zrefclever_ref_typeset_font_-
tl ..... 2241, 2243, 3404
\l__zrefclever_refbounds_first_-
pb_seq ..... 3748,
3936, 4012, 4062, 4134, 4185, 4266
\l__zrefclever_refbounds_first_-
rb_seq . 3748, 3940, 4168, 4300, 4630
\l__zrefclever_refbounds_first_-
seq 3748, 3928, 4311, 4533, 4579, 4601
\l__zrefclever_refbounds_first_-
sg_seq . 3748, 3932, 3990, 4001, 4098
\l__zrefclever_refbounds_last_-
pe_seq ..... 3748, 3960,
4009, 4032, 4059, 4111, 4141, 4263
\l__zrefclever_refbounds_last_-
re_seq .....
.. 3748, 3964, 4213, 4221, 4288, 4296
\l__zrefclever_refbounds_last_-
seq 3748, 3956, 4040, 4075, 4120, 4156
\l__zrefclever_refbounds_mid_rb_-
seq ... 3748, 3948, 4181, 4195, 4641
\l__zrefclever_refbounds_mid_re_-
seq ..... 3748, 3952, 4663, 4671
\l__zrefclever_refbounds_mid_seq
..... 3748, 3944, 4072, 4150,
4520, 4544, 4556, 4590, 4612, 4620
\l__zrefclever_reffont_tl .....
..... 3733, 3904, 4699, 4716,
4741, 4757, 4797, 4806, 4844, 4872
\g__zrefclever_rf_opts_bool_-
maybe_type_specific_seq ....
..... 52, 559, 1429, 2541, 2805, 3333
\g__zrefclever_rf_opts_seq_-
refbounds_seq .....
..... 559, 1351, 2507, 2765, 3271
\g__zrefclever_rf_opts_tl_maybe_-
type_specific_seq 559, 1075, 2968
\g__zrefclever_rf_opts_tl_not_-
type_specific_seq .....
..... 559, 1050, 2587, 2944
\g__zrefclever_rf_opts_tl_-
reference_seq ..... 559, 2459
\g__zrefclever_rf_opts_tl_type_-
names_seq ..... 559, 1312, 3231
\g__zrefclever_rf_opts_tl_-
typesetup_seq ..... 559, 2599
\l__zrefclever_setup_language_tl
..... 554, 735, 763, 768, 778,
783, 794, 799, 925, 977, 984, 998,
1015, 1021, 1030, 1038, 1062, 1087,
1096, 1119, 1125, 1133, 1140, 1154,
1161, 1169, 1177, 1191, 1198, 1206,
1214, 1228, 1235, 1243, 1251, 1276,
1283, 1292, 1300, 1330, 1340, 1363,
1379, 1395, 1412, 1441, 1448, 1457,
1465, 1479, 1486, 1495, 1503, 2848,
2889, 2896, 2910, 2927, 2933, 2956,
2980, 2988, 3011, 3017, 3025, 3032,
3046, 3053, 3061, 3069, 3083, 3090,
3098, 3106, 3120, 3127, 3135, 3143,
3168, 3175, 3184, 3192, 3249, 3259,
3293, 3318, 3345, 3353, 3366, 3374
\l__zrefclever_setup_type_tl 554,
926, 968, 969, 1001, 1022, 1031,
1039, 1057, 1082, 1097, 1114, 1134,
1141, 1149, 1170, 1178, 1186, 1207,
1215, 1223, 1244, 1252, 1271, 1293,
1301, 1319, 1331, 1341, 1358, 1396,
1413, 1436, 1458, 1466, 1474, 1496,
1504, 2582, 2584, 2611, 2618, 2636,
2641, 2650, 2656, 2665, 2671, 2680,
2686, 2695, 2700, 2720, 2727, 2777,
2791, 2815, 2824, 2833, 2847, 2880,
2881, 2913, 2934, 2951, 2975, 2989,
3006, 3026, 3033, 3041, 3062, 3070,
3078, 3099, 3107, 3115, 3136, 3144,
3163, 3185, 3193, 3238, 3250, 3260,
3278, 3319, 3340, 3354, 3361, 3375
\l__zrefclever_sort_decided_bool
..... 3447, 3530, 3544, 3554,
3558, 3570, 3580, 3595, 3610, 3634
\__zrefclever_sort_default:nn ...
..... 87, 3488, 3504
\__zrefclever_sort_default_-
different_types:nn .....

```

..... [44](#), [85](#), [90](#), [3515](#), [3647](#)  
 \\_zrefclever\_sort\_default\_same\_  
   type:nn ..... [85](#), [87](#), [3514](#), [3517](#)  
 \\_zrefclever\_sort\_labels: .....  
   ..... [85–87](#), [91](#), [3402](#), [3451](#)  
 \\_zrefclever\_sort\_page:nn .....  
   ..... [91](#), [3487](#), [3699](#)  
 \l\_zrefclever\_sort\_prior\_a\_int .  
   ..... [3448](#),  
   [3649](#), [3655](#), [3656](#), [3662](#), [3672](#), [3680](#)  
 \l\_zrefclever\_sort\_prior\_b\_int .  
   ..... [3448](#),  
   [3650](#), [3657](#), [3658](#), [3665](#), [3673](#), [3681](#)  
 \l\_zrefclever\_tlastsep\_tl .....  
   ..... [3733](#), [3791](#), [4411](#)  
 \l\_zrefclever\_tlistsep\_tl .....  
   ..... [3733](#), [3787](#), [4382](#)  
 \l\_zrefclever\_tpairsep\_tl .....  
   ..... [3733](#), [3783](#), [4404](#)  
 \l\_zrefclever\_type\_count\_int ...  
   . [92](#), [117](#), [3711](#), [3772](#), [4379](#), [4381](#),  
   [4394](#), [4419](#), [4434](#), [4908](#), [4920](#), [5115](#)  
 \l\_zrefclever\_type\_first\_label\_  
   tl ..... [92](#),  
   [114](#), [3714](#), [3767](#), [3985](#), [4236](#), [4247](#),  
   [4251](#), [4279](#), [4330](#), [4348](#), [4352](#), [4428](#),  
   [4462](#), [4770](#), [4776](#), [4784](#), [4788](#), [4801](#),  
   [4834](#), [4851](#), [4855](#), [4861](#), [4876](#), [4889](#)  
 \l\_zrefclever\_type\_first\_label\_  
   type\_tl ... [92](#), [117](#), [3714](#), [3768](#),  
   [3987](#), [4240](#), [4429](#), [4464](#), [4896](#), [4939](#),  
   [4955](#), [4963](#), [4975](#), [4981](#), [4998](#), [5014](#),  
   [5024](#), [5032](#), [5041](#), [5063](#), [5073](#), [5085](#)  
 \l\_zrefclever\_type\_first\_  
   refbounds\_seq .....  
   ... [3748](#), [3989](#), [4000](#), [4011](#), [4061](#),  
   [4097](#), [4133](#), [4167](#), [4184](#), [4265](#), [4299](#),  
   [4310](#), [4331](#), [4532](#), [4578](#), [4600](#), [4629](#),  
   [4798](#), [4799](#), [4803](#), [4807](#), [4846](#), [4859](#),  
   [4864](#), [4867](#), [4873](#), [4874](#), [4878](#), [4879](#)  
 \l\_zrefclever\_type\_first\_  
   refbounds\_set\_bool ..... [3748](#),  
   [3777](#), [3991](#), [4002](#), [4013](#), [4064](#), [4100](#),  
   [4136](#), [4170](#), [4187](#), [4267](#), [4301](#),  
   [4308](#), [4438](#), [4535](#), [4581](#), [4603](#), [4632](#)  
 \l\_zrefclever\_type\_name\_gender\_  
   seq ... [3720](#), [4965](#), [4967](#), [4970](#), [4985](#)  
 \l\_zrefclever\_type\_name\_  
   missing\_bool .....  
   ... [3720](#), [4820](#), [4892](#), [4899](#), [5021](#), [5081](#)  
 \\_zrefclever\_type\_name\_setup: ..  
   ..... [20](#), [22](#), [114](#), [4317](#), [4887](#)  
 \l\_zrefclever\_type\_name\_tl .....  
   ..... [114](#), [117](#),  
   [3720](#), [4355](#), [4361](#), [4793](#), [4813](#), [4828](#),  
   [4830](#), [4891](#), [4898](#), [5002](#), [5018](#), [5020](#),  
   [5036](#), [5045](#), [5067](#), [5077](#), [5079](#), [5099](#)  
 \l\_zrefclever\_typeset\_compress\_  
   bool ..... [1618](#), [1621](#), [4447](#)  
 \l\_zrefclever\_typeset\_labels\_  
   seq [91](#), [3708](#), [3763](#), [3797](#), [3799](#), [3805](#)  
 \l\_zrefclever\_typeset\_last\_bool  
   ..... [92](#), [3708](#),  
   [3794](#), [3795](#), [3802](#), [3827](#), [4391](#), [5114](#)  
 \l\_zrefclever\_typeset\_name\_bool  
   .. [1567](#), [1574](#), [1579](#), [1584](#), [4319](#), [4335](#)  
 \l\_zrefclever\_typeset\_queue\_  
   curr\_tl ..... [92](#),  
   [94](#), [112](#), [117](#), [3714](#), [3766](#), [4005](#),  
   [4028](#), [4036](#), [4054](#), [4067](#), [4106](#),  
   [4115](#), [4137](#), [4145](#), [4152](#), [4176](#), [4190](#),  
   [4207](#), [4217](#), [4234](#), [4259](#), [4282](#), [4292](#),  
   [4321](#), [4328](#), [4338](#), [4372](#), [4388](#), [4399](#),  
   [4405](#), [4412](#), [4426](#), [4427](#), [4516](#), [4539](#),  
   [4551](#), [4585](#), [4607](#), [4616](#), [4636](#), [4657](#),  
   [4667](#), [4913](#), [4935](#), [4946](#), [5109](#), [5113](#)  
 \l\_zrefclever\_typeset\_queue\_  
   prev\_tl . [92](#), [3714](#), [3765](#), [4383](#), [4425](#)  
 \l\_zrefclever\_typeset\_range\_  
   bool ... [1752](#), [1970](#), [1973](#), [3401](#), [4232](#)  
 \l\_zrefclever\_typeset\_ref\_bool .  
   .. [1566](#), [1573](#), [1578](#), [1583](#), [4319](#), [4325](#)  
 \\_zrefclever\_typeset\_refs: .....  
   ..... [91](#), [93](#), [94](#), [3405](#), [3761](#)  
 \\_zrefclever\_typeset\_refs\_last\_  
   of\_type: [98](#), [112](#), [114](#), [117](#), [3970](#), [3975](#)  
 \\_zrefclever\_typeset\_refs\_not\_  
   last\_of\_type: .....  
   ..... [92](#), [99](#), [112](#), [122](#), [3972](#), [4441](#)  
 \l\_zrefclever\_typeset\_sort\_bool  
   ..... [1594](#), [1597](#), [3400](#)  
 \l\_zrefclever\_typesort\_seq .....  
   . [44](#), [90](#), [1603](#), [1608](#), [1609](#), [1615](#), [3651](#)  
 \l\_zrefclever\_verbose\_testing\_  
   bool ..... [3760](#), [4387](#)  
 \\_zrefclever\_zcref:nnn .....  
   ..... [27](#), [56](#), [3386](#), [3387](#)  
 \\_zrefclever\_zcref:nnnn [83](#), [85](#), [3387](#)  
 \l\_zrefclever\_zcref\_labels\_seq .  
   ..... [85](#), [86](#), [3391](#),  
   [3422](#), [3427](#), [3431](#), [3456](#), [3459](#), [3764](#)  
 \l\_zrefclever\_zcref\_note\_tl ...  
   ..... [2273](#), [2276](#), [3407](#), [3414](#)  
 \l\_zrefclever\_zcref\_with\_check\_  
   bool ..... [2280](#), [2297](#), [3397](#), [3418](#)  
 \\_zrefclever\_zcsetup:n .....  
   ..... [67](#), [2576](#), [2577](#), [5307](#),  
   [5331](#), [5337](#), [5345](#), [5367](#), [5386](#), [5436](#),

|                                     |                                       |
|-------------------------------------|---------------------------------------|
| 5440, 5441, 5450, 5505, 5539, 5592, | available_bool .....                  |
| 5617, 5629, 5641, 5656, 5676, 5699  | .. 2279, 2292, 2304, 2316, 3396, 3417 |
| \l_zrefclever_zrefcheck_-           |                                       |