

# The zref-clever package

## Code documentation

gusbrs

<https://github.com/gusbrs/zref-clever>  
<https://www.ctan.org/pkg/zref-clever>

Version v0.4.3 – 2023-11-14

**EXPERIMENTAL**

### Contents

<b>1</b>	<b>Initial setup</b>	<b>2</b>
<b>2</b>	<b>Dependencies</b>	<b>3</b>
<b>3</b>	<b>zref setup</b>	<b>3</b>
<b>4</b>	<b>Plumbing</b>	<b>8</b>
	4.1 Auxiliary . . . . .	8
	4.2 Messages . . . . .	8
	4.3 Data extraction . . . . .	11
	4.4 Option infra . . . . .	12
	4.5 Reference format . . . . .	20
	4.6 Languages . . . . .	24
	4.7 Language files . . . . .	30
	4.8 Options . . . . .	43
<b>5</b>	<b>Configuration</b>	<b>68</b>
	5.1 \zcsetup . . . . .	68
	5.2 \zcRefTypeSetup . . . . .	68
	5.3 \zcLanguageSetup . . . . .	73
<b>6</b>	<b>User interface</b>	<b>84</b>
	6.1 \zceref . . . . .	84
	6.2 \zcpageref . . . . .	85
<b>7</b>	<b>Sorting</b>	<b>86</b>
<b>8</b>	<b>Typesetting</b>	<b>92</b>

<b>9</b>	<b>Compatibility</b>	<b>127</b>
9.1	appendix	127
9.2	appendices	128
9.3	memoir	129
9.4	amsmath	130
9.5	mathtools	132
9.6	breqn	133
9.7	listings	134
9.8	enumitem	134
9.9	subcaption	135
9.10	subfig	135
<b>10</b>	<b>Language files</b>	<b>136</b>
10.1	Localization guidelines	136
10.2	English	139
10.3	German	143
10.4	French	151
10.5	Portuguese	155
10.6	Spanish	160
10.7	Dutch	164
10.8	Italian	168
	<b>Index</b>	<b>173</b>

## 1 Initial setup

Start the DocStrip guards.

<sup>1</sup> `\*package`

Identify the internal prefix (L<sup>A</sup>T<sub>E</sub>X3 DocStrip convention).

<sup>2</sup> `\@@=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). 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). Also, 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 (ltxcmds), 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>). Critically, the new `label` hook introduced in the 2023-06-01 release, alongside the corresponding new hooks with arguments, just simplifies and improves label setting so much, by allowing `\zlabel` to be set with `\label`, that it is definitely a must for `zrefclever`, so we require that too. Finally, since we followed the move to e-type expansion, to play safe we require the 2023-11-01 kernel or newer.

<sup>3</sup> `\def\zrefclever@required@kernel{2023-11-01}`

```

4 \NeedsTeXFormat{LaTeX2e}[\zrefclever@required@kernel]
5 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
6 \IfFormatAtLeastTF{\zrefclever@required@kernel}
7   {}
8   {%
9     \PackageError{zref-clever}{LaTeX kernel too old}
10    {%
11      'zref-clever' requires a LaTeX kernel \zrefclever@required@kernel\space or newer.%
12    }%
13  }%

  Identify the package.
14 \ProvidesExplPackage {zref-clever} {2023-11-14} {0.4.3}
15 {Clever LaTeX cross-references based on zref}

```

## 2 Dependencies

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

```

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

```

## 3 zref setup

For the purposes of the package, we need to store some information with the labels, some of it standard, some of it not so much. So, we have to setup `zref` to do so.

Some basic properties are handled by `zref` itself, or some of its modules. The `default` and `page` properties are provided by `zref-base`, while `zref-abspage` provides the `abspage` property which gives us a safe and easy way to sort labels for page references.

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

```

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

```

The reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the<counter>` and store it “clean” in `thecounter` for reserved use. Since `\@currentlabel`, which populates the `default` property, is *more reliable* than `\@currentcounter`, `thecounter` is meant to be kept as an *option* (`ref` option), in case there’s need to use `zref-clever` together with `\labelformat`. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in `texdoc source2e`, section `ltxref.dtx`. We just drop the `\p@...` prefix.

```

22 \zref@newprop { thecounter }

```

```

23 {
24   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
25     { \use:c { the \l__zrefclever_current_counter_tl } }
26     {
27       \cs_if_exist:cT { c@ \@currentcounter }
28         { \use:c { the \@currentcounter } }
29     }
30 }
31 \zref@addprop \ZREF@mainlist { thecounter }

```

Much of the work of `zref-clever` relies on the association between a label’s “counter” and its “type” (see the User manual section on “Reference types”). Superficially examined, one might think this relation could just be stored in a global property list, rather than in the label itself. However, there are cases in which we want to distinguish different types for the same counter, depending on the document context. Hence, we need to store the “type” of the “counter” for each “label”. In setting this, the presumption is that the label’s type has the same name as its counter, unless it is specified otherwise by the `countertype` option, as stored in `\l__zrefclever_counter_type_prop`.

```

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

```

Since the default/`thecounter` and `page` properties store the “*printed* representation” of their respective counters, for sorting and compressing purposes, we are also interested in their numeric values. So we store them in `zc@cntval` and `zc@pgval`. For this, we use `\c@⟨counter⟩`, which contains the counter’s numerical value (see ‘`texdoc source2e`’, section ‘`ltxcounts.dtx`’). Also, even if we can’t find a valid `\@currentcounter`, we set the value of 0 to the property, so that it is never empty (the property’s default is not sufficient to avoid that), because we rely on this value being a number and an empty value there will result in “Missing number, treated as zero.” error. A typical situation where this might occur is the user setting a label before `\refstepcounter` is called for the first time in the document. A user error, no doubt, but we should avoid a hard crash.

```

47 \zref@newprop { zc@cntval } [0]
48 {
49   \bool_lazy_and:nnTF
50     { ! \tl_if_empty_p:N \l__zrefclever_current_counter_tl }
51     { \cs_if_exist_p:c { c@ \l__zrefclever_current_counter_tl } }
52     { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
53     {
54       \bool_lazy_and:nnTF
55         { ! \tl_if_empty_p:N \@currentcounter }

```

```

56         { \cs_if_exist_p:c { c@ \@currentcounter } }
57         { \int_use:c { c@ \@currentcounter } }
58         { 0 }
59     }
60 }
61 \zref@addprop \ZREF@mainlist { zc@cntval }
62 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
63 \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 `\c1@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see `ltxcounts.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 `\c1@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresettters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\c1@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other

“general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

Recursively generate a *sequence* of “enclosing counters” values, for a given  $\langle counter \rangle$  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>}
64 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
65 {
66   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
67   {
68     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
69     \__zrefclever_get_enclosing_counters_value:e
70     { \__zrefclever_counter_reset_by:n {#1} }
71   }
72 }

```

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)).

```
73 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }
```

(End of 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 {<counter>}
74 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
75 {
76   \bool_if:nTF
77   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
78   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
79   {
80     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
81     { \__zrefclever_counter_reset_by_aux:nn {#1} }
82   }
83 }
84 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
85 {

```

```

86   \cs_if_exist:cT { c@ #2 }
87   {
88     \tl_if_empty:cF { c1@ #2 }
89     {
90       \tl_map_tokens:cn { c1@ #2 }
91       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
92     }
93   }
94 }
95 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
96 {
97   \str_if_eq:nnT {#2} {#3}
98   { \tl_map_break:n { \seq_map_break:n {#1} } }
99 }

```

(End of definition for `\__zrefclever_counter_reset_by:n`.)

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

```

100 \zref@newprop { zc@enclval }
101 {
102   \__zrefclever_get_enclosing_counters_value:e
103   \l__zrefclever_current_counter_tl
104 }
105 \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`, is simple and smart: store with the label what `\thepage` would return, if the counter `\c@page` was “1”. That would 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. However, `x` expanding `\thepage` can lead to errors for some `babel` packages which redefine `\roman` containing non-expandable material (see <https://chat.stackexchange.com/transcript/message/63810027#63810027>, <https://chat.stackexchange.com/transcript/message/63810318#63810318>, <https://chat.stackexchange.com/transcript/message/63810720#63810720> and discussion). So I went for something a little different. As mentioned, we want to know if `\thepage` is the same for different labels, or if it has changed. We can thus test this directly, by comparing `\thepage` with a stored value of it, `\g__zrefclever_prev_page_format_tl`, and stepping a counter every time they differ. Of course, this cannot be done at label setting time, since it is not expandable. But we can do that comparison before shipout and then define the label property as starred (`\zref@newprop*{zc@pgfmt}`), so that the label comes after the counter, and we can get the correct value of the counter.

```

106 \int_new:N \g__zrefclever_page_format_int
107 \tl_new:N \g__zrefclever_prev_page_format_tl

```

```

108 \AddToHook { shipout / before }
109 {
110   \tl_if_eq:NMF \g__zrefclever_prev_page_format_tl \thepage
111   {
112     \int_gincr:N \g__zrefclever_page_format_int
113     \tl_gset_eq:NN \g__zrefclever_prev_page_format_tl \thepage
114   }
115 }
116 \zref@newprop* { zc@pgfmt } { \int_use:N \g__zrefclever_page_format_int }
117 \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`  
`\__zrefclever_if_class_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.

```

118 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
119 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
120 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
121 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

*(End of definition for \\_\_zrefclever\_if\_package\_loaded:n and \\_\_zrefclever\_if\_class\_loaded:n.)*

`\l__zrefclever_tmpa_tl`  
`\l__zrefclever_tmpb_tl`  
`\l__zrefclever_tmpa_seq`  
`\g__zrefclever_tmpa_seq`  
`\l__zrefclever_tmpa_bool`  
`\l__zrefclever_tmpa_int` Temporary scratch variables.

```

122 \tl_new:N \l__zrefclever_tmpa_tl
123 \tl_new:N \l__zrefclever_tmpb_tl
124 \seq_new:N \l__zrefclever_tmpa_seq
125 \seq_new:N \g__zrefclever_tmpa_seq
126 \bool_new:N \l__zrefclever_tmpa_bool
127 \int_new:N \l__zrefclever_tmpa_int

```

*(End of definition for \l\_\_zrefclever\_tmpa\_tl and others.)*

### 4.2 Messages

```

128 \msg_new:nnn { zref-clever } { option-not-type-specific }
129 {
130   Option~'#1'~is-not-type-specific~\msg_line_context:..~
131   Set~it~in~'\iow_char:N\zcLanguageSetup'~before~first~'type'~
132   switch~or~as~package~option.
133 }
134 \msg_new:nnn { zref-clever } { option-only-type-specific }
135 {
136   No~type~specified~for~option~'#1'~\msg_line_context:..~
137   Set~it~after~'type'~switch.
138 }

```



```

139 \msg_new:nnn { zref-clever } { key-requires-value }
140   { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:.. }
141 \msg_new:nnn { zref-clever } { language-declared }
142   { Language~'#1'~is~already~declared~\msg_line_context:..Nothing~to~do. }
143 \msg_new:nnn { zref-clever } { unknown-language-alias }
144   {
145     Language~'#1'~is~unknown~\msg_line_context:..Can't~alias~to~it.~
146     See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
147     '\iow_char:N\zcDeclareLanguageAlias'.
148   }
149 \msg_new:nnn { zref-clever } { unknown-language-setup }
150   {
151     Language~'#1'~is~unknown~\msg_line_context:..Can't~set~it~up.~
152     See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
153     '\iow_char:N\zcDeclareLanguageAlias'.
154   }
155 \msg_new:nnn { zref-clever } { unknown-language-opt }
156   {
157     Language~'#1'~is~unknown~\msg_line_context:..
158     See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
159     '\iow_char:N\zcDeclareLanguageAlias'.
160   }
161 \msg_new:nnn { zref-clever } { unknown-language-decl }
162   {
163     Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:..
164     See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
165     '\iow_char:N\zcDeclareLanguageAlias'.
166   }
167 \msg_new:nnn { zref-clever } { language-no-decl-ref }
168   {
169     Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..
170     Nothing~to~do~with~option~'d=#2'.
171   }
172 \msg_new:nnn { zref-clever } { language-no-gender }
173   {
174     Language~'#1'~has~no~declared~gender~\msg_line_context:..
175     Nothing~to~do~with~option~'#2=#3'.
176   }
177 \msg_new:nnn { zref-clever } { language-no-decl-setup }
178   {
179     Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..
180     Nothing~to~do~with~option~'case=#2'.
181   }
182 \msg_new:nnn { zref-clever } { unknown-decl-case }
183   {
184     Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:..
185     Using~default~declension~case.
186   }
187 \msg_new:nnn { zref-clever } { nudge-multitype }
188   {
189     Reference~with~multiple~types~\msg_line_context:..
190     You~may~wish~to~separate~them~or~review~language~around~it.
191   }
192 \msg_new:nnn { zref-clever } { nudge-comptosing }

```

```

193 {
194   Multiple-labels-have-been-compressed-into-singular-type-name-
195   for-type-#1-\msg_line_context:.
196 }
197 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
198 {
199   Option-’sg’-signals-that-a-singular-type-name-was-expected-
200   \msg_line_context:.~But-type-#1’-has-plural-type-name.
201 }
202 \msg_new:nnn { zref-clever } { gender-not-declared }
203 { Language-#1’-has-no-#2’-gender-declared-\msg_line_context:. }
204 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
205 {
206   Gender-mismatch-for-type-#1’-\msg_line_context:.~
207   You’ve-specified-’g=#2’-but-type-name-is-#3’-for-language-#4’.
208 }
209 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
210 {
211   You’ve-specified-’g=#1’-\msg_line_context:.~
212   But-gender-for-type-#2’-is-not-declared-for-language-#3’.
213 }
214 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
215 { Unknown-value-#1’-for-’nudgeif’-option-\msg_line_context:. }
216 \msg_new:nnn { zref-clever } { option-document-only }
217 { Option-#1’-is-only-available-after-\iow_char:N\begin\{document\}. }
218 \msg_new:nnn { zref-clever } { langfile-loaded }
219 { Loaded-#1’-language-file. }
220 \msg_new:nnn { zref-clever } { zref-property-undefined }
221 {
222   Option-’ref=#1’-requested-\msg_line_context:.~
223   But-the-property-#1’-is-not-declared,-falling-back-to-’default’.
224 }
225 \msg_new:nnn { zref-clever } { endrange-property-undefined }
226 {
227   Option-’endrange=#1’-requested-\msg_line_context:.~
228   But-the-property-#1’-is-not-declared,-’endrange’-not-set.
229 }
230 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
231 {
232   Option-’hyperref’-only-available-in-the-preamble-\msg_line_context:.~
233   To-inhibit-hyperlinking-locally,-you-can-use-the-starred-version-of-
234   ’\iow_char:N\zcref’.
235 }
236 \msg_new:nnn { zref-clever } { missing-hyperref }
237 { Missing-’hyperref’-package.-Setting-’hyperref=false’. }
238 \msg_new:nnn { zref-clever } { option-preamble-only }
239 { Option-#1’-only-available-in-the-preamble-\msg_line_context:. }
240 \msg_new:nnn { zref-clever } { unknown-compat-module }
241 {
242   Unknown-compatibility-module-#1’-given-to-option-’nocompat’.~
243   Nothing-to-do.
244 }
245 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
246 {

```

```

247     The~value~of~option~'#1'~must~be~a~comma~sepatared~list~
248     of~four~items.~We~received~'#2'~items~\msg_line_context:..~
249     Option~not~set.
250   }
251   \msg_new:nnn { zref-clever } { missing-zref-check }
252   {
253     Option~'check'~requested~\msg_line_context:..~
254     But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
255   }
256   \msg_new:nnn { zref-clever } { zref-check-too-old }
257   {
258     Option~'check'~requested~\msg_line_context:..~
259     But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
260   }
261   \msg_new:nnn { zref-clever } { missing-type }
262   { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
263   \msg_new:nnn { zref-clever } { missing-property }
264   { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
265   \msg_new:nnn { zref-clever } { missing-name }
266   { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
267   \msg_new:nnn { zref-clever } { single-element-range }
268   { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
269   \msg_new:nnn { zref-clever } { compat-package }
270   { Loaded~support~for~'#1'~package. }
271   \msg_new:nnn { zref-clever } { compat-class }
272   { Loaded~support~for~'#1'~documentclass. }
273   \msg_new:nnn { zref-clever } { option-deprecated }
274   {
275     Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
276     Use~'#2'~instead.
277   }
278   \msg_new:nnn { zref-clever } { load-time-options }
279   {
280     'zref-clever'~does~not~accept~load~time~options.~
281     To~configure~package~options,~use~'\iow_char:N\zcsetup'.
282   }

```

### 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}
283 \cs_new_protected:Npn \_zrefclever_extract_default:Nnnn #1#2#3#4
284 {
285   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
286   { \zref@extractdefault {#2} {#3} {#4} }
287 }
288 \cs_generate_variant:Nn \_zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

(End of 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}

289 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
290 {
291   \exp_args:NNo \exp_args:No
292   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
293 }
294 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

(End of definition for \__zrefclever_extract_unexp:nnn.)

```

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

```

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

295 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
296 { \zref@extractdefault {#1} {#2} {#3} }

(End of 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 (`t1`, `seq`, `bool`, etc.).

```

\__zrefclever_opt_varname_general:nn {\option} {\data type}

297 \cs_new:Npn \__zrefclever_opt_varname_general:nn #1#2
298 { l__zrefclever_opt_general_ #1 _ #2 }

```

(End of 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 *option* for *ref type*.

```
\_zrefclever_opt_varname_type:nnn {<ref type>} {<option>} {<data type>}
```

```
299 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
```

```
300 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
```

```
301 \cs_generate_variant:Nn \_zrefclever_opt_varname_type:nnn { enn , een }
```

(End of 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 *option* for *lang* (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 {<lang>} {<option>} {<data type>}
```

```
302 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
```

```
303 {
```

```
304   \_zrefclever_language_if_declared:nTF {#1}
```

```
305   {
```

```
306     g\_zrefclever_opt_language_
```

```
307     \tl_use:c { \_zrefclever_language_varname:n {#1} }
```

```
308     _ #2 _ #3
```

```
309   }
```

```
310   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
```

```
311 }
```

```
312 \cs_generate_variant:Nn \_zrefclever_opt_varname_language:nnn { enn }
```

(End of 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 *option* for *lang*.

```
\_zrefclever_opt_varname_lang_default:nnn {<lang>} {<option>} {<data type>}
```

```
313 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
```

```
314 {
```

```
315   \_zrefclever_language_if_declared:nTF {#1}
```

```
316   {
```

```
317     g\_zrefclever_opt_lang_
```

```
318     \tl_use:c { \_zrefclever_language_varname:n {#1} }
```

```
319     _default_ #2 _ #3
```

```
320   }
```

```
321   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
```

```
322 }
```

```
323 \cs_generate_variant:Nn \_zrefclever_opt_varname_lang_default:nnn { enn }
```

(End of 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:nmmm {<lang>} {<ref type>}
    {<option>} {<data type>}
324 \cs_new:Npn \_zrefclever_opt_varname_lang_type:nmmm #1#2#3#4
325 {
326   \_zrefclever_language_if_declared:nTF {#1}
327   {
328     g__zrefclever_opt_lang_
329     \tl_use:c { \_zrefclever_language_varname:n {#1} }
330     _type_ #2 _ #3 _ #4
331   }
332   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
333 }
334 \cs_generate_variant:Nn
335   \_zrefclever_opt_varname_lang_type:nmmm { eenn , eeen }

```

(End of definition for \\_zrefclever\_opt\_varname\_lang\_type:nmmm.)

\\_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>}
336 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
337 { c__zrefclever_opt_fallback_ #1 _ #2 }

```

(End of 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>}
338 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
339 { \cs_to_str:N #1 _is_set_bool }

```

(End of 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>}
340 \cs_new_protected:Npn \_zrefclever_opt_tl_set:Nn #1#2
341 {
342   \tl_if_exist:NF #1
343   { \tl_new:N #1 }
344   \tl_set:Nn #1 {#2}

```

```

345     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
346     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
347     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
348   }
349 \cs_generate_variant:Nn \__zrefclever_opt_tl_set:Nn { cn }
350 \cs_new_protected:Npn \__zrefclever_opt_tl_clear:N #1
351   {
352     \tl_if_exist:NF #1
353     { \tl_new:N #1 }
354     \tl_clear:N #1
355     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
356     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
357     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
358   }
359 \cs_generate_variant:Nn \__zrefclever_opt_tl_clear:N { c }
360 \cs_new_protected:Npn \__zrefclever_opt_tl_gset:Nn #1#2
361   {
362     \tl_if_exist:NF #1
363     { \tl_new:N #1 }
364     \tl_gset:Nn #1 {#2}
365   }
366 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset:Nn { cn }
367 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear:N #1
368   {
369     \tl_if_exist:NF #1
370     { \tl_new:N #1 }
371     \tl_gclear:N #1
372   }
373 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear:N { c }

```

(End of definition for \\_\_zrefclever\_opt\_tl\_set:Nn and others.)

\\_\_zrefclever\_opt\_tl\_unset:N Unset  $\langle$ option  $tl$  $\rangle$ .

```

    \__zrefclever_opt_tl_unset:N { $\langle$ option  $tl$  $\rangle$ }
374 \cs_new_protected:Npn \__zrefclever_opt_tl_unset:N #1
375   {
376     \tl_if_exist:NT #1
377     {
378       \tl_clear:N #1 % ?
379       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
380       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
381       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
382     }
383   }
384 \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End of 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) { $\langle$ option  $tl$  $\rangle$ } { $\langle$ true $\rangle$ } { $\langle$ false $\rangle$ }

```

```

385 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
386 {
387   \tl_if_exist:NTF #1
388   {
389     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
390     {
391       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
392       { \prg_return_true: }
393       { \prg_return_false: }
394     }
395     { \prg_return_true: }
396   }
397   { \prg_return_false: }
398 }

```

(End of definition for \\_\_zrefclever\_opt\_tl\_if\_set:NTF.)

\\_\_zrefclever\_opt\_tl\_gset\_if\_new:Nn  
\\_\_zrefclever\_opt\_tl\_gclear\_if\_new:N

```

\__zrefclever_opt_tl_gset_if_new:Nn {(option tl)} {(value)}
\__zrefclever_opt_tl_gclear_if_new:N {(option tl)}
399 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
400 {
401   \__zrefclever_opt_tl_if_set:NF #1
402   {
403     \tl_if_exist:NF #1
404     { \tl_new:N #1 }
405     \tl_gset:Nn #1 {#2}
406   }
407 }
408 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
409 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
410 {
411   \__zrefclever_opt_tl_if_set:NF #1
412   {
413     \tl_if_exist:NF #1
414     { \tl_new:N #1 }
415     \tl_gclear:N #1
416   }
417 }
418 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }

```

(End of 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)}
419 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
420 {
421   \__zrefclever_opt_tl_if_set:NTF #1
422   {
423     \tl_set_eq:NN #2 #1
424     \prg_return_true:
425   }
426   { \prg_return_false: }
427 }

```



```

428 \prg_generate_conditional_variant:Nnn
429 \__zrefclever_opt_tl_get:NN { cN } { F }

```

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

```

\__zrefclever_opt_seq_set_clist_split:Nn
\__zrefclever_opt_seq_gset_clist_split:Nn
\__zrefclever_opt_seq_set_eq:NN
\__zrefclever_opt_seq_gset_eq:NN
430 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
431 { \seq_set_split:Nnn #1 { , } {#2} }
432 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
433 { \seq_gset_split:Nnn #1 { , } {#2} }
434 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
435 {
436   \seq_if_exist:NF #1
437   { \seq_new:N #1 }
438   \seq_set_eq:NN #1 #2
439   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
440   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
441   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
442 }
443 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
444 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
445 {
446   \seq_if_exist:NF #1
447   { \seq_new:N #1 }
448   \seq_gset_eq:NN #1 #2
449 }
450 \cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }

```

(End of definition for \\_\_zrefclever\_opt\_seq\_set\_clist\_split:Nn and others.)

\\_\_zrefclever\_opt\_seq\_unset:N Unset  $\langle option\ seq \rangle$ .

```

\__zrefclever_opt_seq_unset:N { $\langle option\ seq \rangle$ }
451 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
452 {
453   \seq_if_exist:NT #1
454   {
455     \seq_clear:N #1 % ?
456     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
457     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
458     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
459   }
460 }
461 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

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

\\_\_zrefclever\_opt\_seq\_if\_set:NTF This conditional *defines* what means to be unset for a sequence option.

```

\__zrefclever_opt_seq_if_set:N(TF) { $\langle option\ seq \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }

```

```

462 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
463 {
464   \seq_if_exist:NTF #1
465   {
466     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
467     {
468       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
469       { \prg_return_true: }
470       { \prg_return_false: }
471     }
472     { \prg_return_true: }
473   }
474   { \prg_return_false: }
475 }
476 \prg_generate_conditional_variant:Nnn
477 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End of definition for \\_\_zrefclever\_opt\_seq\_if\_set:NTF.)

```

\__zrefclever_opt_seq_get:NNTF \__zrefclever_opt_seq_get:NN(TF) {<option seq to get>} {<seq var to set>}
  {<true>} {<false>}
478 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
479 {
480   \__zrefclever_opt_seq_if_set:NTF #1
481   {
482     \seq_set_eq:NN #2 #1
483     \prg_return_true:
484   }
485   { \prg_return_false: }
486 }
487 \prg_generate_conditional_variant:Nnn
488 \__zrefclever_opt_seq_get:NN { cN } { F }

```

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

\\_\_zrefclever\_opt\_bool\_unset:N Unset <option bool>.

```

\__zrefclever_opt_bool_unset:N {<option bool>}
489 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
490 {
491   \bool_if_exist:NT #1
492   {
493     % \bool_set_false:N #1 % ?
494     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
495     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
496     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
497   }
498 }
499 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }

```

(End of 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>}

```

```

500 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
501 {
502   \bool_if_exist:NTF #1
503   {
504     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
505     {
506       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
507       { \prg_return_true: }
508       { \prg_return_false: }
509     }
510     { \prg_return_true: }
511   }
512   { \prg_return_false: }
513 }
514 \prg_generate_conditional_variant:Nnn
515 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

*(End of definition for \\_\_zrefclever\_opt\_bool\_if\_set:NTF.)*

```

\__zrefclever_opt_bool_set_true:N
\__zrefclever_opt_bool_set_false:N
\__zrefclever_opt_bool_gset_true:N
\__zrefclever_opt_bool_gset_false:N
516 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
517 {
518   \bool_if_exist:NF #1
519   { \bool_new:N #1 }
520   \bool_set_true:N #1
521   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
522   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
523   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
524 }
525 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
526 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
527 {
528   \bool_if_exist:NF #1
529   { \bool_new:N #1 }
530   \bool_set_false:N #1
531   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
532   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
533   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
534 }
535 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
536 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
537 {
538   \bool_if_exist:NF #1
539   { \bool_new:N #1 }
540   \bool_gset_true:N #1
541 }
542 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
543 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
544 {
545   \bool_if_exist:NF #1
546   { \bool_new:N #1 }

```

```

547     \bool_gset_false:N #1
548   }
549   \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }

```

*(End of definition for \\_\_zrefclever\_opt\_bool\_set\_true:N and others.)*

```

\__zrefclever_opt_bool_get:NTF
    \__zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
    {<true>} {<false>}
550 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
551 {
552   \__zrefclever_opt_bool_if_set:NTF #1
553   {
554     \bool_set_eq:NN #2 #1
555     \prg_return_true:
556   }
557   { \prg_return_false: }
558 }
559 \prg_generate_conditional_variant:Nnn
560 \__zrefclever_opt_bool_get:NN { cN } { F }

```

*(End of definition for \\_\_zrefclever\_opt\_bool\_get:NTF.)*

```

\__zrefclever_opt_bool_if:NTF
    \__zrefclever_opt_bool_if:N(TF) {<option bool>} {<true>} {<false>}
561 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
562 {
563   \__zrefclever_opt_bool_if_set:NTF #1
564   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
565   { \prg_return_false: }
566 }
567 \prg_generate_conditional_variant:Nnn
568 \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

*(End of 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.

`\l__zrefclever_setup_type_tl` Store “current” type, language, and declension cases in different places for type-specific and language-specific options handling, notably in `\__zrefclever_provide_langfile:n`, `\zcRefTypeSetup`, and `\zcLanguageSetup`, but also for language specific options retrieval.

```

569 \tl_new:N \l__zrefclever_setup_type_tl
570 \tl_new:N \l__zrefclever_setup_language_tl
571 \tl_new:N \l__zrefclever_lang_decl_case_tl
572 \seq_new:N \l__zrefclever_lang_declension_seq
573 \seq_new:N \l__zrefclever_lang_gender_seq

```

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

`zrefclever_rf_opts_tl_not_type_specific_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.

```

574 \seq_new:N \g__zrefclever_rf_opts_tl_not_type_specific_seq
575 \seq_gset_from_clist:Nn
576   \g__zrefclever_rf_opts_tl_not_type_specific_seq
577   {
578     tpairsep ,
579     tlistsep ,
580     tlastsep ,
581     notesep ,
582   }
583 \seq_new:N \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
584 \seq_gset_from_clist:Nn
585   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
586   {

```

```

587     namesep ,
588     pairsep ,
589     listsep ,
590     lastsep ,
591     rangeseq ,
592     namefont ,
593     reffont ,
594 }
595 \seq_new:N \g__zrefclever_rf_opts_seq_refbounds_seq
596 \seq_gset_from_clist:Nn
597   \g__zrefclever_rf_opts_seq_refbounds_seq
598   {
599     refbounds-first ,
600     refbounds-first-sg ,
601     refbounds-first-pb ,
602     refbounds-first-rb ,
603     refbounds-mid ,
604     refbounds-mid-rb ,
605     refbounds-mid-re ,
606     refbounds-last ,
607     refbounds-last-pe ,
608     refbounds-last-re ,
609   }
610 \seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
611 \seq_gset_from_clist:Nn
612   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
613   {
614     cap ,
615     abbrev ,
616     rangetopair ,
617   }

```

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:`.

```

618 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
619 \seq_gset_from_clist:Nn
620   \g__zrefclever_rf_opts_tl_type_names_seq
621   {
622     Name-sg ,
623     name-sg ,
624     Name-pl ,
625     name-pl ,
626     Name-sg-ab ,
627     name-sg-ab ,
628     Name-pl-ab ,
629     name-pl-ab ,
630   }

```

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

```

631 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
632 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
633   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
634   \g__zrefclever_rf_opts_tl_type_names_seq
635 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq

```

```

636 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
637 \g__zrefclever_rf_opts_tl_not_type_specific_seq
638 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

(End of 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.

```

639 \clist_map_inline:nn
640 {
641   reference ,
642   typesetup ,
643   langsetup ,
644   langfile ,
645 }
646 {
647   \keys_define:nn { zref-clever/ #1 }
648   {
649     +refbounds-first .meta:n =
650     {
651       refbounds-first = {##1} ,
652       refbounds-first-sg = {##1} ,
653       refbounds-first-pb = {##1} ,
654       refbounds-first-rb = {##1} ,
655     } ,
656     +refbounds-mid .meta:n =
657     {
658       refbounds-mid = {##1} ,
659       refbounds-mid-rb = {##1} ,
660       refbounds-mid-re = {##1} ,
661     } ,
662     +refbounds-last .meta:n =
663     {
664       refbounds-last = {##1} ,
665       refbounds-last-pe = {##1} ,
666       refbounds-last-re = {##1} ,
667     } ,
668     +refbounds-rb .meta:n =
669     {
670       refbounds-first-rb = {##1} ,
671       refbounds-mid-rb = {##1} ,
672     } ,
673     +refbounds-re .meta:n =
674     {
675       refbounds-mid-re = {##1} ,
676       refbounds-last-re = {##1} ,
677     } ,
678     +refbounds .meta:n =
679     {
680       +refbounds-first = {##1} ,
681       +refbounds-mid = {##1} ,
682       +refbounds-last = {##1} ,
683     } ,
684     refbounds .meta:n = { +refbounds = {##1} } ,
685   }

```

```

686 }
687 \clist_map_inline:nn
688 {
689   reference ,
690   typesetup ,
691 }
692 {
693   \keys_define:nn { zref-clever/ #1 }
694   {
695     +refbounds-first .default:o = \c_novalue_tl ,
696     +refbounds-mid .default:o = \c_novalue_tl ,
697     +refbounds-last .default:o = \c_novalue_tl ,
698     +refbounds-rb .default:o = \c_novalue_tl ,
699     +refbounds-re .default:o = \c_novalue_tl ,
700     +refbounds .default:o = \c_novalue_tl ,
701     refbounds .default:o = \c_novalue_tl ,
702   }
703 }
704 \clist_map_inline:nn
705 {
706   langsetup ,
707   langfile ,
708 }
709 {
710   \keys_define:nn { zref-clever/ #1 }
711   {
712     +refbounds-first .value_required:n = true ,
713     +refbounds-mid .value_required:n = true ,
714     +refbounds-last .value_required:n = true ,
715     +refbounds-rb .value_required:n = true ,
716     +refbounds-re .value_required:n = true ,
717     +refbounds .value_required:n = true ,
718     refbounds .value_required:n = true ,
719   }
720 }

```

## 4.6 Languages

`\l__zrefclever_current_language_tl` is an internal alias for babel's `\languagename` 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.

```

721 \tl_new:N \l__zrefclever_ref_language_tl
722 \tl_new:N \l__zrefclever_current_language_tl
723 \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`.

```

724 \tl_new:N \l_zrefclever_ref_language_tl
725 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```



(End of 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⟩}
```

```
726 \cs_new:Npn \_zrefclever_language_varname:n #1
727   { g_zrefclever_declared_language_ #1 _tl }
```

(End of definition for `\_zrefclever_language_varname:n`.)

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

```
728 \cs_set_eq:NN \zrefclever_language_varname:n
729   \_zrefclever_language_varname:n
```

(End of 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⟩}
```

```
730 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
731   {
732     \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
733     { \prg_return_true: }
734     { \prg_return_false: }
735   }
736 \prg_generate_conditional_variant:Nnn
737   \_zrefclever_language_if_declared:n { e } { T , F , TF }
```

(End of definition for `\_zrefclever_language_if_declared:nTF`.)

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

```
738 \prg_set_eq_conditional:NNn \zrefclever_language_if_declared:n
739   \_zrefclever_language_if_declared:n { TF }
```

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

`\zcDeclareLanguage` Declare a new language for use with zref-clever. *⟨language⟩* 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”. [*⟨options⟩*] receive a **k=v** set of options, with three valid options. The first, **declension**, takes the noun declension cases prefixes for *⟨language⟩* as a comma separated list, whose first element is taken to be the default case. The second, **gender**, receives the genders for *⟨language⟩* as comma separated list. The third, **allcaps**, is a boolean, and indicates that for *⟨language⟩* all nouns must be capitalized for grammatical reasons, in which case, the **cap** option is disregarded for *⟨language⟩*. If *⟨language⟩* is already known, just warn. This implies a particular restriction regarding [*⟨options⟩*], 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}
740 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
741 {
742   \group_begin:
743   \tl_if_empty:nF {#2}
744   {
745     \__zrefclever_language_if_declared:nTF {#2}
746     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
747     {
748       \tl_new:c { \__zrefclever_language_varname:n {#2} }
749       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
750       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
751       \keys_set:nn { zref-clever/declarelang } {#1}
752     }
753   }
754   \group_end:
755 }
756 \@onlypreamble \zcDeclareLanguage

```

(End of definition for `\zcDeclareLanguage`.)

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

```

\zcDeclareLanguageAlias {language alias} {aliased language}
757 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
758 {
759   \tl_if_empty:nF {#1}
760   {
761     \__zrefclever_language_if_declared:nTF {#2}
762     {
763       \tl_new:c { \__zrefclever_language_varname:n {#1} }
764       \tl_gset:ce { \__zrefclever_language_varname:n {#1} }
765       { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
766     }
767     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
768   }
769 }
770 \@onlypreamble \zcDeclareLanguageAlias

```

(End of definition for `\zcDeclareLanguageAlias`.)

```

771 \keys_define:nn { zref-clever/declarelang }
772 {
773   declension .code:n =
774   {
775     \seq_new:c
776     {
777       \__zrefclever_opt_varname_language:enn
778       { \l__zrefclever_setup_language_tl } { declension } { seq }
779     }
780     \seq_gset_from_clist:cn
781     {

```

```

782         \_zrefclever_opt_varname_language:enn
783         { \l_zrefclever_setup_language_tl } { declension } { seq }
784     }
785     {#1}
786 },
787 declension .value_required:n = true ,
788 gender .code:n =
789 {
790     \seq_new:c
791     {
792         \_zrefclever_opt_varname_language:enn
793         { \l_zrefclever_setup_language_tl } { gender } { seq }
794     }
795     \seq_gset_from_clist:cn
796     {
797         \_zrefclever_opt_varname_language:enn
798         { \l_zrefclever_setup_language_tl } { gender } { seq }
799     }
800     {#1}
801 },
802 gender .value_required:n = true ,
803 allcaps .choices:nn =
804 { true , false }
805 {
806     \bool_new:c
807     {
808         \_zrefclever_opt_varname_language:enn
809         { \l_zrefclever_setup_language_tl } { allcaps } { bool }
810     }
811     \use:c { bool_gset_ \l_keys_choice_tl :c }
812     {
813         \_zrefclever_opt_varname_language:enn
814         { \l_zrefclever_setup_language_tl } { allcaps } { bool }
815     }
816 },
817 allcaps .default:n = true ,
818 }

```

`\_zrefclever_process_language_settings:` Auxiliary function for `\_zrefclever_zceref: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_zceref:nnn`, where current values for `\l_zrefclever_ref_language_tl` and `\l_zrefclever_ref_decl_case_tl` are in place.

```

819 \cs_new_protected:Npn \_zrefclever_process_language_settings:
820 {
821     \_zrefclever_language_if_declared:eTF
822     { \l_zrefclever_ref_language_tl }
823     {

```

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.

```

824     \__zrefclever_opt_seq_get:cNF
825     {
826         \__zrefclever_opt_varname_language:enn
827         { \l__zrefclever_ref_language_tl } { declension } { seq }
828     }
829     \l__zrefclever_lang_declension_seq
830     { \seq_clear:N \l__zrefclever_lang_declension_seq }
831     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
832     {
833         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
834         {
835             \msg_warning:nnee { zref-clever }
836             { language-no-decl-ref }
837             { \l__zrefclever_ref_language_tl }
838             { \l__zrefclever_ref_decl_case_tl }
839             \tl_clear:N \l__zrefclever_ref_decl_case_tl
840         }
841     }
842     {
843         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
844         {
845             \seq_get_left:NN \l__zrefclever_lang_declension_seq
846             \l__zrefclever_ref_decl_case_tl
847         }
848         {
849             \seq_if_in:NVF \l__zrefclever_lang_declension_seq
850             \l__zrefclever_ref_decl_case_tl
851             {
852                 \msg_warning:nnee { zref-clever }
853                 { unknown-decl-case }
854                 { \l__zrefclever_ref_decl_case_tl }
855                 { \l__zrefclever_ref_language_tl }
856                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
857                 \l__zrefclever_ref_decl_case_tl
858             }
859         }
860     }

```

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.

```

861     \__zrefclever_opt_seq_get:cNF
862     {
863         \__zrefclever_opt_varname_language:enn
864         { \l__zrefclever_ref_language_tl } { gender } { seq }
865     }
866     \l__zrefclever_lang_gender_seq
867     { \seq_clear:N \l__zrefclever_lang_gender_seq }
868     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq

```

```

869     {
870     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
871     {
872     \msg_warning:nneee { zref-clever }
873     { language-no-gender }
874     { \l__zrefclever_ref_language_tl }
875     { g }
876     { \l__zrefclever_ref_gender_tl }
877     \tl_clear:N \l__zrefclever_ref_gender_tl
878     }
879     }
880     {
881     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
882     {
883     \seq_if_in:NVF \l__zrefclever_lang_gender_seq
884     \l__zrefclever_ref_gender_tl
885     {
886     \msg_warning:nnee { zref-clever }
887     { gender-not-declared }
888     { \l__zrefclever_ref_language_tl }
889     { \l__zrefclever_ref_gender_tl }
890     \tl_clear:N \l__zrefclever_ref_gender_tl
891     }
892     }
893     }

```

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

```

894     \__zrefclever_opt_bool_if:cT
895     {
896     \__zrefclever_opt_varname_language:enn
897     { \l__zrefclever_ref_language_tl } { allcaps } { bool }
898     }
899     { \keys_set:nn { zref-clever/reference } { cap = true } }
900     }
901     {

```

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

```

902     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
903     {
904     \msg_warning:nnee { zref-clever } { unknown-language-decl }
905     { \l__zrefclever_ref_decl_case_tl }
906     { \l__zrefclever_ref_language_tl }
907     \tl_clear:N \l__zrefclever_ref_decl_case_tl
908     }
909     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
910     {
911     \msg_warning:nneee { zref-clever }
912     { language-no-gender }
913     { \l__zrefclever_ref_language_tl }
914     { g }
915     { \l__zrefclever_ref_gender_tl }
916     \tl_clear:N \l__zrefclever_ref_gender_tl
917     }

```

```

918     }
919 }

```

(End of 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 `.lbr` 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.

```

920 \seq_new:N \g_zrefclever_loaded_langfiles_seq

```

(End of definition for `\g_zrefclever_loaded_langfiles_seq`.)

`\_zrefclever_provide_langfile:n` Load language file for known *<language>* if it is available and if it has not already been loaded.

```

    \__zrefclever_provide_langfile:n {<language>}
921 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
922 {
923   \group_begin:
924   \@bsphack
925   \__zrefclever_language_if_declared:nT {#1}
926   {
927     \seq_if_in:NeF
928     \g__zrefclever_loaded_langfiles_seq
929     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
930     {
931       \exp_args:Ne \file_get:nnNTF
932       {
933         zref-clever-
934         \tl_use:c { \__zrefclever_language_varname:n {#1} }
935         .lang
936       }
937       { \ExplSyntaxOn }
938       \l__zrefclever_tmpa_tl
939       {
940         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
941         \tl_clear:N \l__zrefclever_setup_type_tl
942         \__zrefclever_opt_seq_get:cNF
943         {
944           \__zrefclever_opt_varname_language:nnn
945           {#1} { declension } { seq }
946         }
947         \l__zrefclever_lang_declension_seq
948         { \seq_clear:N \l__zrefclever_lang_declension_seq }
949         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
950         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
951         {
952           \seq_get_left:NN \l__zrefclever_lang_declension_seq
953           \l__zrefclever_lang_decl_case_tl
954         }
955         \__zrefclever_opt_seq_get:cNF
956         {
957           \__zrefclever_opt_varname_language:nnn
958           {#1} { gender } { seq }
959         }
960         \l__zrefclever_lang_gender_seq
961         { \seq_clear:N \l__zrefclever_lang_gender_seq }
962         \keys_set:nV { zref-clever/langfile } \l__zrefclever_tmpa_tl
963         \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
964         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
965         \msg_info:nne { zref-clever } { langfile-loaded }
966         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
967       }
968     }

```

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.

```

969             \seq_gput_right:Nc \g__zrefclever_loaded_langfiles_seq
970             { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
971         }
972     }
973 }
974 \@esphack
975 \group_end:
976 }
977 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { e }

```

(End of 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.

```

978 \keys_define:nn { zref-clever/langfile }
979 {
980     type .code:n =
981     {
982         \tl_if_empty:NTF {#1}
983         { \tl_clear:N \l__zrefclever_setup_type_tl }
984         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
985     } ,
986
987     case .code:n =
988     {
989         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
990         {
991             \msg_info:nnee { zref-clever } { language-no-decl-setup }
992             { \l__zrefclever_setup_language_tl } {#1}
993         }
994         {
995             \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
996             { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
997             {
998                 \msg_info:nnee { zref-clever } { unknown-decl-case }
999                 {#1} { \l__zrefclever_setup_language_tl }
1000                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
1001                 \l__zrefclever_lang_decl_case_tl
1002             }
1003         }
1004     } ,
1005     case .value_required:n = true ,
1006
1007     gender .value_required:n = true ,
1008     gender .code:n =
1009     {
1010         \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1011         {
1012             \msg_info:nneee { zref-clever } { language-no-gender }
1013             { \l__zrefclever_setup_language_tl } { gender } {#1}
1014         }
1015         {

```



```

1016 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1017 {
1018   \msg_info:nnn { zref-clever }
1019   { option-only-type-specific } { gender }
1020 }
1021 {
1022   \seq_clear:N \l__zrefclever_tmpa_seq
1023   \clist_map_inline:nn {#1}
1024   {
1025     \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1026     { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
1027     {
1028       \msg_info:nnee { zref-clever }
1029       { gender-not-declared }
1030       { \l__zrefclever_setup_language_tl } {##1}
1031     }
1032   }
1033   \__zrefclever_opt_seq_if_set:cF
1034   {
1035     \__zrefclever_opt_varname_lang_type:eenn
1036     { \l__zrefclever_setup_language_tl }
1037     { \l__zrefclever_setup_type_tl }
1038     { gender }
1039     { seq }
1040   }
1041   {
1042     \seq_new:c
1043     {
1044       \__zrefclever_opt_varname_lang_type:eenn
1045       { \l__zrefclever_setup_language_tl }
1046       { \l__zrefclever_setup_type_tl }
1047       { gender }
1048       { seq }
1049     }
1050     \seq_gset_eq:cN
1051     {
1052       \__zrefclever_opt_varname_lang_type:eenn
1053       { \l__zrefclever_setup_language_tl }
1054       { \l__zrefclever_setup_type_tl }
1055       { gender }
1056       { seq }
1057     }
1058     \l__zrefclever_tmpa_seq
1059   }
1060 }
1061 }
1062 } ,
1063 }
1064 \seq_map_inline:Nn
1065 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1066 {
1067   \keys_define:nn { zref-clever/langfile }
1068   {
1069     #1 .value_required:n = true ,

```

```

1070     #1 .code:n =
1071     {
1072         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1073         {
1074             \__zrefclever_opt_tl_gset_if_new:cn
1075             {
1076                 \__zrefclever_opt_varname_lang_default:enn
1077                 { \l__zrefclever_setup_language_tl }
1078                 {#1} { t1 }
1079             }
1080             {##1}
1081         }
1082         {
1083             \msg_info:nnn { zref-clever }
1084             { option-not-type-specific } {#1}
1085         }
1086     } ,
1087 }
1088 }
1089 \seq_map_inline:Nn
1090 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1091 {
1092     \keys_define:nn { zref-clever/langfile }
1093     {
1094         #1 .value_required:n = true ,
1095         #1 .code:n =
1096         {
1097             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1098             {
1099                 \__zrefclever_opt_tl_gset_if_new:cn
1100                 {
1101                     \__zrefclever_opt_varname_lang_default:enn
1102                     { \l__zrefclever_setup_language_tl }
1103                     {#1} { t1 }
1104                 }
1105                 {##1}
1106             }
1107             {
1108                 \__zrefclever_opt_tl_gset_if_new:cn
1109                 {
1110                     \__zrefclever_opt_varname_lang_type:eenn
1111                     { \l__zrefclever_setup_language_tl }
1112                     { \l__zrefclever_setup_type_tl }
1113                     {#1} { t1 }
1114                 }
1115                 {##1}
1116             }
1117         } ,
1118     }
1119 }
1120 \keys_define:nn { zref-clever/langfile }
1121 {
1122     endrange .value_required:n = true ,
1123     endrange .code:n =

```

```

1124 {
1125   \str_case:nnF {#1}
1126   {
1127     { ref }
1128     {
1129       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1130       {
1131         \__zrefclever_opt_tl_gclear_if_new:c
1132         {
1133           \__zrefclever_opt_varname_lang_default:enn
1134           { \l__zrefclever_setup_language_tl }
1135           { endrangefunc } { tl }
1136         }
1137         \__zrefclever_opt_tl_gclear_if_new:c
1138         {
1139           \__zrefclever_opt_varname_lang_default:enn
1140           { \l__zrefclever_setup_language_tl }
1141           { endrangeprop } { tl }
1142         }
1143       }
1144     {
1145       \__zrefclever_opt_tl_gclear_if_new:c
1146       {
1147         \__zrefclever_opt_varname_lang_type:eenn
1148         { \l__zrefclever_setup_language_tl }
1149         { \l__zrefclever_setup_type_tl }
1150         { endrangefunc } { tl }
1151       }
1152       \__zrefclever_opt_tl_gclear_if_new:c
1153       {
1154         \__zrefclever_opt_varname_lang_type:eenn
1155         { \l__zrefclever_setup_language_tl }
1156         { \l__zrefclever_setup_type_tl }
1157         { endrangeprop } { tl }
1158       }
1159     }
1160   }
1161
1162   { stripprefix }
1163   {
1164     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1165     {
1166       \__zrefclever_opt_tl_gset_if_new:cn
1167       {
1168         \__zrefclever_opt_varname_lang_default:enn
1169         { \l__zrefclever_setup_language_tl }
1170         { endrangefunc } { tl }
1171       }
1172       { __zrefclever_get_endrange_stripprefix }
1173       \__zrefclever_opt_tl_gclear_if_new:c
1174       {
1175         \__zrefclever_opt_varname_lang_default:enn
1176         { \l__zrefclever_setup_language_tl }
1177         { endrangeprop } { tl }

```

```

1178     }
1179   }
1180   {
1181     \zrefclever_opt_tl_gset_if_new:cn
1182     {
1183       \zrefclever_opt_varname_lang_type:eenn
1184       { \l_zrefclever_setup_language_tl }
1185       { \l_zrefclever_setup_type_tl }
1186       { endrangefunc } { tl }
1187     }
1188     { __zrefclever_get_endrange_stripprefix }
1189     \zrefclever_opt_tl_gclear_if_new:c
1190     {
1191       \zrefclever_opt_varname_lang_type:eenn
1192       { \l_zrefclever_setup_language_tl }
1193       { \l_zrefclever_setup_type_tl }
1194       { endrangeprop } { tl }
1195     }
1196   }
1197 }
1198
1199 { pagecomp }
1200 {
1201   \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1202   {
1203     \zrefclever_opt_tl_gset_if_new:cn
1204     {
1205       \zrefclever_opt_varname_lang_default:enn
1206       { \l_zrefclever_setup_language_tl }
1207       { endrangefunc } { tl }
1208     }
1209     { __zrefclever_get_endrange_pagecomp }
1210     \zrefclever_opt_tl_gclear_if_new:c
1211     {
1212       \zrefclever_opt_varname_lang_default:enn
1213       { \l_zrefclever_setup_language_tl }
1214       { endrangeprop } { tl }
1215     }
1216   }
1217   {
1218     \zrefclever_opt_tl_gset_if_new:cn
1219     {
1220       \zrefclever_opt_varname_lang_type:eenn
1221       { \l_zrefclever_setup_language_tl }
1222       { \l_zrefclever_setup_type_tl }
1223       { endrangefunc } { tl }
1224     }
1225     { __zrefclever_get_endrange_pagecomp }
1226     \zrefclever_opt_tl_gclear_if_new:c
1227     {
1228       \zrefclever_opt_varname_lang_type:eenn
1229       { \l_zrefclever_setup_language_tl }
1230       { \l_zrefclever_setup_type_tl }
1231       { endrangeprop } { tl }

```

```

1232     }
1233   }
1234 }
1235
1236 { pagecomp2 }
1237 {
1238   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1239   {
1240     \__zrefclever_opt_tl_gset_if_new:cn
1241     {
1242       \__zrefclever_opt_varname_lang_default:enn
1243       { \l__zrefclever_setup_language_tl }
1244       { endrangefunc } { tl }
1245     }
1246     { __zrefclever_get_endrange_pagecomptwo }
1247     \__zrefclever_opt_tl_gclear_if_new:c
1248     {
1249       \__zrefclever_opt_varname_lang_default:enn
1250       { \l__zrefclever_setup_language_tl }
1251       { endrangeprop } { tl }
1252     }
1253   }
1254   {
1255     \__zrefclever_opt_tl_gset_if_new:cn
1256     {
1257       \__zrefclever_opt_varname_lang_type:eenn
1258       { \l__zrefclever_setup_language_tl }
1259       { \l__zrefclever_setup_type_tl }
1260       { endrangefunc } { tl }
1261     }
1262     { __zrefclever_get_endrange_pagecomptwo }
1263     \__zrefclever_opt_tl_gclear_if_new:c
1264     {
1265       \__zrefclever_opt_varname_lang_type:eenn
1266       { \l__zrefclever_setup_language_tl }
1267       { \l__zrefclever_setup_type_tl }
1268       { endrangeprop } { tl }
1269     }
1270   }
1271 }
1272 }
1273 {
1274   \tl_if_empty:nTF {#1}
1275   {
1276     \msg_info:nnn { zref-clever }
1277     { endrange-property-undefined } {#1}
1278   }
1279   {
1280     \zref@ifpropundefined {#1}
1281     {
1282       \msg_info:nnn { zref-clever }
1283       { endrange-property-undefined } {#1}
1284     }
1285   }

```

```

1286 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1287 {
1288   \__zrefclever_opt_tl_gset_if_new:cn
1289   {
1290     \__zrefclever_opt_varname_lang_default:enn
1291     { \l__zrefclever_setup_language_tl }
1292     { endrangefunc } { tl }
1293   }
1294   { __zrefclever_get_endrange_property }
1295   \__zrefclever_opt_tl_gset_if_new:cn
1296   {
1297     \__zrefclever_opt_varname_lang_default:enn
1298     { \l__zrefclever_setup_language_tl }
1299     { endrangeprop } { tl }
1300   }
1301   {#1}
1302 }
1303 {
1304   \__zrefclever_opt_tl_gset_if_new:cn
1305   {
1306     \__zrefclever_opt_varname_lang_type:eenn
1307     { \l__zrefclever_setup_language_tl }
1308     { \l__zrefclever_setup_type_tl }
1309     { endrangefunc } { tl }
1310   }
1311   { __zrefclever_get_endrange_property }
1312   \__zrefclever_opt_tl_gset_if_new:cn
1313   {
1314     \__zrefclever_opt_varname_lang_type:eenn
1315     { \l__zrefclever_setup_language_tl }
1316     { \l__zrefclever_setup_type_tl }
1317     { endrangeprop } { tl }
1318   }
1319   {#1}
1320 }
1321 }
1322 }
1323 }
1324 } ,
1325 }
1326 \seq_map_inline:Nn
1327 \g__zrefclever_rf_opts_tl_type_names_seq
1328 {
1329   \keys_define:nm { zref-clever/langfile }
1330   {
1331     #1 .value_required:n = true ,
1332     #1 .code:n =
1333     {
1334       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1335       {
1336         \msg_info:nnn { zref-clever }
1337         { option-only-type-specific } {#1}
1338       }
1339     }

```

```

1340     \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1341     {
1342         \__zrefclever_opt_tl_gset_if_new:cn
1343         {
1344             \__zrefclever_opt_varname_lang_type:een
1345             { \l__zrefclever_setup_language_tl }
1346             { \l__zrefclever_setup_type_tl }
1347             {#1} { tl }
1348         }
1349         {##1}
1350     }
1351     {
1352         \__zrefclever_opt_tl_gset_if_new:cn
1353         {
1354             \__zrefclever_opt_varname_lang_type:een
1355             { \l__zrefclever_setup_language_tl }
1356             { \l__zrefclever_setup_type_tl }
1357             { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
1358         }
1359         {##1}
1360     }
1361     } ,
1362 }
1363 }
1364 }
1365 \seq_map_inline:Nn
1366   \g__zrefclever_rf_opts_seq_refbounds_seq
1367   {
1368     \keys_define:nm { zref-clever/langfile }
1369     {
1370       #1 .value_required:n = true ,
1371       #1 .code:n =
1372       {
1373         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1374         {
1375           \__zrefclever_opt_seq_if_set:cF
1376           {
1377             \__zrefclever_opt_varname_lang_default:enn
1378             { \l__zrefclever_setup_language_tl } {#1} { seq }
1379           }
1380           {
1381             \seq_gclear:N \g__zrefclever_tmpa_seq
1382             \__zrefclever_opt_seq_gset_clist_split:Nn
1383             \g__zrefclever_tmpa_seq {##1}
1384             \bool_lazy_or:nnTF
1385             { \tl_if_empty_p:n {##1} }
1386             {
1387               \int_compare_p:nNn
1388               { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1389             }
1390             {
1391               \__zrefclever_opt_seq_gset_eq:cN
1392               {
1393                 \__zrefclever_opt_varname_lang_default:enn

```

```

1394         { \l__zrefclever_setup_language_tl }
1395         {#1} { seq }
1396     }
1397     \g__zrefclever_tmpa_seq
1398 }
1399 {
1400     \msg_info:nnee { zref-clever }
1401     { refbounds-must-be-four }
1402     {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
1403 }
1404 }
1405 }
1406 {
1407     \__zrefclever_opt_seq_if_set:cF
1408     {
1409         \__zrefclever_opt_varname_lang_type:eenn
1410         { \l__zrefclever_setup_language_tl }
1411         { \l__zrefclever_setup_type_tl } {#1} { seq }
1412     }
1413     {
1414         \seq_gclear:N \g__zrefclever_tmpa_seq
1415         \__zrefclever_opt_seq_gset_clist_split:Nn
1416         \g__zrefclever_tmpa_seq {##1}
1417         \bool_lazy_or:nnTF
1418         { \tl_if_empty_p:n {##1} }
1419         {
1420             \int_compare_p:nNn
1421             { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1422         }
1423         {
1424             \__zrefclever_opt_seq_gset_eq:cN
1425             {
1426                 \__zrefclever_opt_varname_lang_type:eenn
1427                 { \l__zrefclever_setup_language_tl }
1428                 { \l__zrefclever_setup_type_tl }
1429                 {#1} { seq }
1430             }
1431             \g__zrefclever_tmpa_seq
1432         }
1433         {
1434             \msg_info:nnee { zref-clever }
1435             { refbounds-must-be-four }
1436             {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
1437         }
1438     }
1439 } ,
1440 }
1441 }
1442 }
1443 \seq_map_inline:Nn
1444 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1445 {
1446     \keys_define:nn { zref-clever/langfile }
1447     {

```



```

1448 #1 .choice: ,
1449 #1 / true .code:n =
1450 {
1451   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1452   {
1453     \__zrefclever_opt_bool_if_set:cF
1454     {
1455       \__zrefclever_opt_varname_lang_default:enn
1456       { \l__zrefclever_setup_language_tl }
1457       {#1} { bool }
1458     }
1459     {
1460       \__zrefclever_opt_bool_gset_true:c
1461       {
1462         \__zrefclever_opt_varname_lang_default:enn
1463         { \l__zrefclever_setup_language_tl }
1464         {#1} { bool }
1465       }
1466     }
1467   }
1468   {
1469     \__zrefclever_opt_bool_if_set:cF
1470     {
1471       \__zrefclever_opt_varname_lang_type:eenn
1472       { \l__zrefclever_setup_language_tl }
1473       { \l__zrefclever_setup_type_tl }
1474       {#1} { bool }
1475     }
1476     {
1477       \__zrefclever_opt_bool_gset_true:c
1478       {
1479         \__zrefclever_opt_varname_lang_type:eenn
1480         { \l__zrefclever_setup_language_tl }
1481         { \l__zrefclever_setup_type_tl }
1482         {#1} { bool }
1483       }
1484     }
1485   }
1486 } ,
1487 #1 / false .code:n =
1488 {
1489   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1490   {
1491     \__zrefclever_opt_bool_if_set:cF
1492     {
1493       \__zrefclever_opt_varname_lang_default:enn
1494       { \l__zrefclever_setup_language_tl }
1495       {#1} { bool }
1496     }
1497     {
1498       \__zrefclever_opt_bool_gset_false:c
1499       {
1500         \__zrefclever_opt_varname_lang_default:enn
1501         { \l__zrefclever_setup_language_tl }

```

```

1502             {#1} { bool }
1503         }
1504     }
1505 }
1506 {
1507     \__zrefclever_opt_bool_if_set:cF
1508     {
1509         \__zrefclever_opt_varname_lang_type:eenn
1510         { \l__zrefclever_setup_language_tl }
1511         { \l__zrefclever_setup_type_tl }
1512         {#1} { bool }
1513     }
1514     {
1515         \__zrefclever_opt_bool_gset_false:c
1516         {
1517             \__zrefclever_opt_varname_lang_type:eenn
1518             { \l__zrefclever_setup_language_tl }
1519             { \l__zrefclever_setup_type_tl }
1520             {#1} { bool }
1521         }
1522     }
1523 }
1524 } ,
1525 #1 .default:n = true ,
1526 no #1 .meta:n = { #1 = false } ,
1527 no #1 .value_forbidden:n = true ,
1528 }
1529 }

```

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.

```

1530 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1531 {
1532     \tl_const:cn
1533     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1534 }
1535 \keyval_parse:nnn
1536 { }
1537 { \__zrefclever_opt_tl_cset_fallback:nn }
1538 {
1539     tpairsep = {,~} ,
1540     tlistsep = {,~} ,
1541     tlastsep = {,~} ,
1542     notesep = {~} ,
1543     namesep = {\nobreakspace} ,
1544     pairsep = {,~} ,
1545     listsep = {,~} ,
1546     lastsep = {,~} ,
1547     rangeseq = {\textendash} ,
1548 }

```

## 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 \}
1549 \cs_new_protected:Npn \_zrefclever_prop_put_non_empty:Nnn #1#2#3
1550 {
1551   \tl_if_empty:nTF {#3}
1552     { \prop_remove:Nn #1 {#2} }
1553     { \prop_put:Nnn #1 {#2} {#3} }
1554 }
```

(End of 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`.

```

1555 \tl_new:N \l_zrefclever_ref_property_tl
1556 \keys_define:nn { zref-clever/reference }
1557 {
1558   ref .code:n =
1559   {
1560     \tl_if_empty:nTF {#1}
1561     {
1562       \msg_warning:nnn { zref-clever }
1563       { zref-property-undefined } {#1}
1564       \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1565     }
1566     {
1567       \zref@ifpropundefined {#1}
1568       {
1569         \msg_warning:nnn { zref-clever }
1570         { zref-property-undefined } {#1}
1571         \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1572       }
1573       { \tl_set:Nn \l_zrefclever_ref_property_tl {#1} }
1574     }
1575   } ,
1576   ref .initial:n = default ,
1577   ref .value_required:n = true ,
1578   page .meta:n = { ref = page } ,
1579   page .value_forbidden:n = true ,
1580 }
```

### typeset option

```
1581 \bool_new:N \l__zrefclever_typeset_ref_bool
1582 \bool_new:N \l__zrefclever_typeset_name_bool
1583 \keys_define:nn { zref-clever/reference }
1584 {
1585     typeset .choice: ,
1586     typeset / both .code:n =
1587     {
1588         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1589         \bool_set_true:N \l__zrefclever_typeset_name_bool
1590     } ,
1591     typeset / ref .code:n =
1592     {
1593         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1594         \bool_set_false:N \l__zrefclever_typeset_name_bool
1595     } ,
1596     typeset / name .code:n =
1597     {
1598         \bool_set_false:N \l__zrefclever_typeset_ref_bool
1599         \bool_set_true:N \l__zrefclever_typeset_name_bool
1600     } ,
1601     typeset .initial:n = both ,
1602     typeset .value_required:n = true ,
1603
1604     noname .meta:n = { typeset = ref } ,
1605     noname .value_forbidden:n = true ,
1606     noref .meta:n = { typeset = name } ,
1607     noref .value_forbidden:n = true ,
1608 }
```

### sort option

```
1609 \bool_new:N \l__zrefclever_typeset_sort_bool
1610 \keys_define:nn { zref-clever/reference }
1611 {
1612     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1613     sort .initial:n = true ,
1614     sort .default:n = true ,
1615     nosort .meta:n = { sort = false } ,
1616     nosort .value_forbidden:n = true ,
1617 }
```

### 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`.

```
1618 \seq_new:N \l__zrefclever_typesort_seq
1619 \keys_define:nn { zref-clever/reference }
1620 {
1621     typesort .code:n =
1622     {
1623         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1624         \seq_reverse:N \l__zrefclever_typesort_seq
1625     }
```

```

1625     } ,
1626     typesort .initial:n =
1627     { part , chapter , section , paragraph } ,
1628     typesort .value_required:n = true ,
1629     notypesort .code:n =
1630     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1631     notypesort .value_forbidden:n = true ,
1632 }

```

### comp option

```

1633 \bool_new:N \l__zrefclever_typeset_compress_bool
1634 \keys_define:nn { zref-clever/reference }
1635 {
1636   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1637   comp .initial:n = true ,
1638   comp .default:n = true ,
1639   nocomp .meta:n = { comp = false } ,
1640   nocomp .value_forbidden:n = true ,
1641 }

```

### 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:VWN`, 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 `VWN`. For this reason, `endrangefunc` should be stored without the signature, which is added, and hard-coded, at the calling place. The first argument is *<beg 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 `\crefstriprefix`). 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`.

```

1642 \NewHook { zref-clever/endrange-setup }
1643 \keys_define:nn { zref-clever/reference }
1644 {
1645   endrange .code:n =
1646   {
1647     \str_case:nnF {#1}
1648     {
1649       { ref }
1650       {
1651         \__zrefclever_opt_tl_clear:c
1652         {
1653           \__zrefclever_opt_varname_general:nn
1654           { endrangefunc } { t1 }
1655         }
1656         \__zrefclever_opt_tl_clear:c
1657         {
1658           \__zrefclever_opt_varname_general:nn
1659           { endrangeprop } { t1 }
1660         }
1661       }
1662     }
1663     { stripprefix }
1664     {
1665       \__zrefclever_opt_tl_set:cn
1666       {
1667         \__zrefclever_opt_varname_general:nn
1668         { endrangefunc } { t1 }
1669       }
1670       { __zrefclever_get_endrange_stripprefix }
1671       \__zrefclever_opt_tl_clear:c
1672       {
1673         \__zrefclever_opt_varname_general:nn
1674         { endrangeprop } { t1 }
1675       }
1676     }
1677     { pagecomp }
1678     {
1679       \__zrefclever_opt_tl_set:cn
1680       {
1681         \__zrefclever_opt_varname_general:nn
1682         { endrangefunc } { t1 }
1683       }
1684       { __zrefclever_get_endrange_pagecomp }
1685       \__zrefclever_opt_tl_clear:c
1686       {
1687         \__zrefclever_opt_varname_general:nn
1688         { endrangeprop } { t1 }
1689       }
1690     }
1691   }
1692

```

```

1693 { pagecomp2 }
1694 {
1695   \_zrefclever_opt_tl_set:cn
1696   {
1697     \_zrefclever_opt_varname_general:nn
1698     { endrangefunc } { t1 }
1699   }
1700   { \_zrefclever_get_endrange_pagecomptwo }
1701   \_zrefclever_opt_tl_clear:c
1702   {
1703     \_zrefclever_opt_varname_general:nn
1704     { endrangeprop } { t1 }
1705   }
1706 }
1707
1708 { unset }
1709 {
1710   \_zrefclever_opt_tl_unset:c
1711   {
1712     \_zrefclever_opt_varname_general:nn
1713     { endrangefunc } { t1 }
1714   }
1715   \_zrefclever_opt_tl_unset:c
1716   {
1717     \_zrefclever_opt_varname_general:nn
1718     { endrangeprop } { t1 }
1719   }
1720 }
1721 }
1722 {
1723   \tl_if_empty:nTF {#1}
1724   {
1725     \msg_warning:nnn { zref-clever }
1726     { endrange-property-undefined } {#1}
1727   }
1728   {
1729     \zref@ifpropundefined {#1}
1730     {
1731       \msg_warning:nnn { zref-clever }
1732       { endrange-property-undefined } {#1}
1733     }
1734     {
1735       \_zrefclever_opt_tl_set:cn
1736       {
1737         \_zrefclever_opt_varname_general:nn
1738         { endrangefunc } { t1 }
1739       }
1740       { \_zrefclever_get_endrange_property }
1741       \_zrefclever_opt_tl_set:cn
1742       {
1743         \_zrefclever_opt_varname_general:nn
1744         { endrangeprop } { t1 }
1745       }
1746       {#1}

```

```

1747     }
1748   }
1749 } ,
1750   \endrange .value_required:n = true ,
1751 }
1752
1753 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1754 {
1755   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1756   {
1757     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1758     {
1759       \__zrefclever_extract_default:Nnvn #3
1760       {#2} { \l__zrefclever_ref_property_tl } { }
1761     }
1762     { \tl_set:Nn #3 { zc@missingproperty } }
1763   }
1764   {
1765     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1766     {

```

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.

```

1767     \bool_if:NTF \l__zrefclever_typeset_range_bool
1768     {
1769       \group_begin:
1770       \bool_set_false:N \l__zrefclever_tmpa_bool
1771       \exp_args:Nee \tl_if_eq:nnT
1772       {
1773         \__zrefclever_extract_unexp:nnn
1774         {#1} { externaldocument } { }
1775       }
1776       {
1777         \__zrefclever_extract_unexp:nnn
1778         {#2} { externaldocument } { }
1779       }
1780     }
1781     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1782     {
1783       \exp_args:Nee \tl_if_eq:nnT
1784       {
1785         \__zrefclever_extract_unexp:nnn
1786         {#1} { zc@pgfmt } { }
1787       }
1788       {
1789         \__zrefclever_extract_unexp:nnn
1790         {#2} { zc@pgfmt } { }
1791       }
1792     }
1793     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1794   }

```



```

1795         \exp_args:Nee \tl_if_eq:nnT
1796         {
1797             \__zrefclever_extract_unexp:nnn
1798             {#1} { zc@counter } { }
1799         }
1800         {
1801             \__zrefclever_extract_unexp:nnn
1802             {#2} { zc@counter } { }
1803         }
1804         {
1805             \exp_args:Nee \tl_if_eq:nnT
1806             {
1807                 \__zrefclever_extract_unexp:nnn
1808                 {#1} { zc@enclval } { }
1809             }
1810             {
1811                 \__zrefclever_extract_unexp:nnn
1812                 {#2} { zc@enclval } { }
1813             }
1814             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1815         }
1816     }
1817 }
1818 \bool_if:NTF \l__zrefclever_tmpa_bool
1819 {
1820     \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1821     {#2} { l__zrefclever_endrangeprop_tl } { }
1822 }
1823 {
1824     \zref@ifrefcontainsprop
1825     {#2} { \l__zrefclever_ref_property_tl }
1826     {
1827         \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1828         {#2} { l__zrefclever_ref_property_tl } { }
1829     }
1830     { \tl_set:Nn \l__zrefclever_tmpb_tl { zc@missingproperty } }
1831 }
1832 \exp_args:NNNV
1833 \group_end:
1834 \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1835 }
1836 {
1837     \__zrefclever_extract_default:Nnvn #3
1838     {#2} { l__zrefclever_endrangeprop_tl } { }
1839 }
1840 }
1841 {
1842     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1843     {
1844         \__zrefclever_extract_default:Nnvn #3
1845         {#2} { l__zrefclever_ref_property_tl } { }
1846     }
1847     { \tl_set:Nn #3 { zc@missingproperty } }
1848 }

```

```

1849     }
1850   }
1851 \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>.

```

1852 \cs_new_protected:Npn \_zrefclever_get_endrange_stripprefix:nnN #1#2#3
1853 {
1854   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1855   {
1856     \group_begin:
1857     \UseHook { zref-clever/endrange-setup }
1858     \tl_set:Nc \l__zrefclever_tmpa_tl
1859       {
1860         \_zrefclever_extract:nnn
1861         {#1} { \l__zrefclever_ref_property_tl } { }
1862       }
1863     \tl_set:Nc \l__zrefclever_tmpb_tl
1864       {
1865         \_zrefclever_extract:nnn
1866         {#2} { \l__zrefclever_ref_property_tl } { }
1867       }
1868     \bool_set_false:N \l__zrefclever_tmpa_bool
1869     \bool_until_do:Nn \l__zrefclever_tmpa_bool
1870     {
1871       \exp_args:Nee \tl_if_eq:nnTF
1872       { \tl_head:V \l__zrefclever_tmpa_tl }
1873       { \tl_head:V \l__zrefclever_tmpb_tl }
1874       {
1875         \tl_set:Nc \l__zrefclever_tmpa_tl
1876           { \tl_tail:V \l__zrefclever_tmpa_tl }
1877         \tl_set:Nc \l__zrefclever_tmpb_tl
1878           { \tl_tail:V \l__zrefclever_tmpb_tl }
1879         \tl_if_empty:NT \l__zrefclever_tmpb_tl
1880           { \bool_set_true:N \l__zrefclever_tmpa_bool }
1881       }
1882       { \bool_set_true:N \l__zrefclever_tmpa_bool }
1883     }
1884     \exp_args:NNNV
1885     \group_end:
1886     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1887   }
1888   { \tl_set:Nn #3 { zc@missingproperty } }
1889 }
1890 \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>).

```

1891 \prg_new_protected_conditional:Npnn
1892   \_zrefclever_is_integer_rgx:n #1 { F , TF }
1893 {
1894   \regex_match:nnTF { \A\d+\Z } {#1}
1895   { \prg_return_true: }
1896   { \prg_return_false: }

```

```

1897 }
1898 \prg_generate_conditional_variant:Nnn
1899 \__zrefclever_is_integer_rgx:n { V } { F , TF }

(End of definition for \__zrefclever_is_integer_rgx:n.)

1900 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1901 {
1902   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1903   {
1904     \group_begin:
1905     \UseHook { zref-clever/endrange-setup }
1906     \tl_set:Nc \l__zrefclever_tmpa_tl
1907     {
1908       \__zrefclever_extract:nnn
1909       {#1} { \l__zrefclever_ref_property_tl } { }
1910     }
1911     \tl_set:Nc \l__zrefclever_tmpb_tl
1912     {
1913       \__zrefclever_extract:nnn
1914       {#2} { \l__zrefclever_ref_property_tl } { }
1915     }
1916     \bool_set_false:N \l__zrefclever_tmpa_bool
1917     \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1918     {
1919       \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1920       { \bool_set_true:N \l__zrefclever_tmpa_bool }
1921     }
1922     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1923     \bool_until_do:Nn \l__zrefclever_tmpa_bool
1924     {
1925       \exp_args:Nee \tl_if_eq:nnTF
1926       { \tl_head:V \l__zrefclever_tmpa_tl }
1927       { \tl_head:V \l__zrefclever_tmpb_tl }
1928       {
1929         \tl_set:Nc \l__zrefclever_tmpa_tl
1930         { \tl_tail:V \l__zrefclever_tmpa_tl }
1931         \tl_set:Nc \l__zrefclever_tmpb_tl
1932         { \tl_tail:V \l__zrefclever_tmpb_tl }
1933         \tl_if_empty:NT \l__zrefclever_tmpb_tl
1934         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1935       }
1936       { \bool_set_true:N \l__zrefclever_tmpa_bool }
1937     }
1938     \exp_args:NNNV
1939     \group_end:
1940     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1941   }
1942   { \tl_set:Nn #3 { zc@missingproperty } }
1943 }
1944 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1945 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1946 {
1947   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1948   {

```

```

1949 \group_begin:
1950 \UseHook { zref-clever/endrange-setup }
1951 \tl_set:Ne \l__zrefclever_tmpa_tl
1952 {
1953   \__zrefclever_extract:nnn
1954   {#1} { \l__zrefclever_ref_property_tl } { }
1955 }
1956 \tl_set:Ne \l__zrefclever_tmpb_tl
1957 {
1958   \__zrefclever_extract:nnn
1959   {#2} { \l__zrefclever_ref_property_tl } { }
1960 }
1961 \bool_set_false:N \l__zrefclever_tmpa_bool
1962 \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1963 {
1964   \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1965   { \bool_set_true:N \l__zrefclever_tmpa_bool }
1966 }
1967 { \bool_set_true:N \l__zrefclever_tmpa_bool }
1968 \bool_until_do:Nn \l__zrefclever_tmpa_bool
1969 {
1970   \exp_args:Nee \tl_if_eq:nnTF
1971   { \tl_head:V \l__zrefclever_tmpa_tl }
1972   { \tl_head:V \l__zrefclever_tmpb_tl }
1973   {
1974     \bool_lazy_or:nnTF
1975     { \int_compare_p:nNn { \l__zrefclever_tmpb_tl } > { 99 } }
1976     {
1977       \int_compare_p:nNn
1978       { \tl_head:V \l__zrefclever_tmpb_tl } = { 0 }
1979     }
1980     {
1981       \tl_set:Ne \l__zrefclever_tmpa_tl
1982       { \tl_tail:V \l__zrefclever_tmpa_tl }
1983       \tl_set:Ne \l__zrefclever_tmpb_tl
1984       { \tl_tail:V \l__zrefclever_tmpb_tl }
1985     }
1986     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1987   }
1988   { \bool_set_true:N \l__zrefclever_tmpa_bool }
1989 }
1990 \exp_args:NNNV
1991 \group_end:
1992 \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1993 }
1994 { \tl_set:Nn #3 { zc@missingproperty } }
1995 }
1996 \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`.

```

1997 \bool_new:N \l__zrefclever_typeset_range_bool
1998 \keys_define:nn { zref-clever/reference }
1999 {
2000   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
2001   range .initial:n = false ,
2002   range .default:n = true ,
2003 }

```

### 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`.

```

2004 \bool_new:N \l__zrefclever_capfirst_bool
2005 \keys_define:nn { zref-clever/reference }
2006 {
2007   capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
2008   capfirst .initial:n = false ,
2009   capfirst .default:n = true ,
2010 }

```

### 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`.

```

2011 \bool_new:N \l__zrefclever_noabbrev_first_bool
2012 \keys_define:nn { zref-clever/reference }
2013 {
2014   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
2015   noabbrevfirst .initial:n = false ,
2016   noabbrevfirst .default:n = true ,
2017 }

```

### S option

```

2018 \keys_define:nn { zref-clever/reference }
2019 {
2020   S .meta:n =
2021     { capfirst = {#1} , noabbrevfirst = {#1} },
2022   S .default:n = true ,
2023 }

```

### hyperref option

```

2024 \bool_new:N \l__zrefclever_hyperlink_bool
2025 \bool_new:N \l__zrefclever_hyperref_warn_bool
2026 \keys_define:nn { zref-clever/reference }
2027 {
2028   hyperref .choice: ,
2029   hyperref / auto .code:n =
2030     {
2031       \bool_set_true:N \l__zrefclever_hyperlink_bool
2032       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2033     } ,
2034   hyperref / true .code:n =

```

```

2035     {
2036       \bool_set_true:N \l__zrefclever_hyperlink_bool
2037       \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2038     } ,
2039     hyperref / false .code:n =
2040     {
2041       \bool_set_false:N \l__zrefclever_hyperlink_bool
2042       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2043     } ,
2044     hyperref .initial:n = auto ,
2045     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.

```

2046     nohyperref .meta:n = { hyperref = false } ,
2047     nohyperref .value_forbidden:n = true ,
2048   }
2049 \AddToHook { begindocument }
2050 {
2051   \__zrefclever_if_package_loaded:nTF { hyperref }
2052   {
2053     \bool_if:NT \l__zrefclever_hyperlink_bool
2054     { \RequirePackage { zref-hyperref } }
2055   }
2056   {
2057     \bool_if:NT \l__zrefclever_hyperref_warn_bool
2058     { \msg_warning:nn { zref-clever } { missing-hyperref } }
2059     \bool_set_false:N \l__zrefclever_hyperlink_bool
2060   }
2061   \keys_define:nn { zref-clever/reference }
2062   {
2063     hyperref .code:n =
2064     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2065     nohyperref .code:n =
2066     { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2067   }
2068 }

```

### **nameinlink option**

```

2069 \str_new:N \l__zrefclever_nameinlink_str
2070 \keys_define:nn { zref-clever/reference }
2071 {
2072   nameinlink .choice: ,
2073   nameinlink / true .code:n =
2074   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2075   nameinlink / false .code:n =
2076   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2077   nameinlink / single .code:n =
2078   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2079   nameinlink / tsingle .code:n =
2080   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,

```

```

2081     nameinlink .initial:n = tsingle ,
2082     nameinlink .default:n = true ,
2083 }

```

### preposinlink option (deprecated)

```

2084 \keys_define:nn { zref-clever/reference }
2085 {
2086     preposinlink .code:n =
2087     {
2088         % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2089         \msg_warning:nmm { zref-clever }{ option-deprecated }
2090         { preposinlink } { refbounds }
2091     } ,
2092 }

```

### 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`.

```

2093 \AddToHook { begindocument }
2094 {
2095     \__zrefclever_if_package_loaded:nTF { babel }
2096     {
2097         \tl_set:Nn \l__zrefclever_current_language_tl { \language }
2098         \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2099     }
2100     {
2101         \__zrefclever_if_package_loaded:nTF { polyglossia }
2102         {
2103             \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2104             \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2105         }
2106         {
2107             \tl_set:Nn \l__zrefclever_current_language_tl { english }
2108             \tl_set:Nn \l__zrefclever_main_language_tl { english }
2109         }
2110     }

```

```

2111 }
2112 \keys_define:nn { zref-clever/reference }
2113 {
2114   lang .code:n =
2115   {
2116     \AddToHook { begindocument }
2117     {
2118       \str_case:nnF {#1}
2119       {
2120         { current }
2121         {
2122           \tl_set:Nn \l__zrefclever_ref_language_tl
2123             { \l__zrefclever_current_language_tl }
2124         }
2125
2126         { main }
2127         {
2128           \tl_set:Nn \l__zrefclever_ref_language_tl
2129             { \l__zrefclever_main_language_tl }
2130         }
2131       }
2132       {
2133         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2134         \__zrefclever_language_if_declared:nF {#1}
2135         {
2136           \msg_warning:nnn { zref-clever }
2137             { unknown-language-opt } {#1}
2138         }
2139       }
2140       \__zrefclever_provide_langfile:e
2141         { \l__zrefclever_ref_language_tl }
2142     }
2143   } ,
2144   lang .initial:n = current ,
2145   lang .value_required:n = true ,
2146 }
2147 \AddToHook { begindocument / before }
2148 {
2149   \AddToHook { begindocument }
2150   {

```

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.

```

2151   \keys_define:nn { zref-clever/reference }
2152   {
2153     lang .code:n =
2154     {
2155       \str_case:nnF {#1}
2156       {
2157         { current }
2158         {
2159           \tl_set:Nn \l__zrefclever_ref_language_tl

```



```

2160         { \l__zrefclever_current_language_tl }
2161     }
2162
2163     { main }
2164     {
2165         \tl_set:Nn \l__zrefclever_ref_language_tl
2166             { \l__zrefclever_main_language_tl }
2167     }
2168 }
2169 {
2170     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2171     \__zrefclever_language_if_declared:nF {#1}
2172     {
2173         \msg_warning:nnn { zref-clever }
2174             { unknown-language-opt } {#1}
2175     }
2176 }
2177 } ,
2178 }
2179 }
2180 }

```

#### 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.

```

2181 \tl_new:N \l__zrefclever_ref_decl_case_tl
2182 \keys_define:nn { zref-clever/reference }
2183 {
2184     d .code:n =
2185         { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2186 }
2187 \AddToHook { begindocument }
2188 {
2189     \keys_define:nn { zref-clever/reference }
2190     {

```

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

```

2191         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2192         d .value_required:n = true ,
2193     }
2194 }

```

#### nudge & co. options

```

2195 \bool_new:N \l__zrefclever_nudge_enabled_bool
2196 \bool_new:N \l__zrefclever_nudge_multitype_bool
2197 \bool_new:N \l__zrefclever_nudge_comptosing_bool

```

```

2198 \bool_new:N \l__zrefclever_nudge_singular_bool
2199 \bool_new:N \l__zrefclever_nudge_gender_bool
2200 \tl_new:N \l__zrefclever_ref_gender_tl
2201 \keys_define:nn { zref-clever/reference }
2202 {
2203   nudge .choice: ,
2204   nudge / true .code:n =
2205     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2206   nudge / false .code:n =
2207     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2208   nudge / ifdraft .code:n =
2209     {
2210       \ifdraft
2211         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2212         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2213     } ,
2214   nudge / iffina1 .code:n =
2215     {
2216       \ifoptionfinal
2217         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2218         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2219     } ,
2220   nudge .initial:n = false ,
2221   nudge .default:n = true ,
2222   nonudge .meta:n = { nudge = false } ,
2223   nonudge .value_forbidden:n = true ,
2224   nudgeif .code:n =
2225     {
2226       \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2227       \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2228       \bool_set_false:N \l__zrefclever_nudge_gender_bool
2229       \clist_map_inline:nn {#1}
2230       {
2231         \str_case:nnF {##1}
2232         {
2233           { multitype }
2234           { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2235           { comptosing }
2236           { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2237           { gender }
2238           { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2239           { all }
2240           {
2241             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2242             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2243             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2244           }
2245         }
2246       }
2247       \msg_warning:nnn { zref-clever }
2248       { nudgeif-unknown-value } {##1}
2249     }
2250   }
2251 } ,

```

```

2252     nudgeif .value_required:n = true ,
2253     nudgeif .initial:n = all ,
2254     sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2255     sg .initial:n = false ,
2256     sg .default:n = true ,
2257     g .code:n =
2258     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2259   }
2260 \AddToHook { begindocument }
2261 {
2262   \keys_define:nn { zref-clever/reference }
2263   {

```

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

```

2264     g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2265     g .value_required:n = true ,
2266   }
2267 }

```

#### font option

```

2268 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2269 \keys_define:nn { zref-clever/reference }
2270 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

#### titleref option

```

2271 \keys_define:nn { zref-clever/reference }
2272 {
2273   titleref .code:n =
2274   {
2275     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2276     \msg_warning:nnee { zref-clever } { option-deprecated } { titleref }
2277     { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2278   } ,
2279 }

```

#### vario option

```

2280 \keys_define:nn { zref-clever/reference }
2281 {
2282   vario .code:n =
2283   {
2284     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2285     \msg_warning:nnee { zref-clever } { option-deprecated } { vario }
2286     { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2287   } ,
2288 }

```

#### note option

```

2289 \tl_new:N \l__zrefclever_zcref_note_tl
2290 \keys_define:nn { zref-clever/reference }
2291 {
2292   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2293   note .value_required:n = true ,
2294 }

```

## check option

Integration with zref-check.

```
2295 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2296 \bool_new:N \l__zrefclever_zceref_with_check_bool
2297 \keys_define:nn { zref-clever/reference }
2298 {
2299   check .code:n =
2300     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2301 }
2302 \AddToHook { begindocument }
2303 {
2304   \__zrefclever_if_package_loaded:nTF { zref-check }
2305   {
2306     \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2307     {
2308       \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2309       \keys_define:nn { zref-clever/reference }
2310       {
2311         check .code:n =
2312         {
2313           \bool_set_true:N \l__zrefclever_zceref_with_check_bool
2314           \keys_set:nn { zref-check / zcheck } {#1}
2315         } ,
2316         check .value_required:n = true ,
2317       }
2318     }
2319     {
2320       \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2321       \keys_define:nn { zref-clever/reference }
2322       {
2323         check .code:n =
2324         {
2325           \msg_warning:nnn { zref-clever }
2326             { zref-check-too-old } { 2021-09-16-v0.2.1 }
2327         } ,
2328       }
2329     }
2330   }
2331   {
2332     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2333     \keys_define:nn { zref-clever/reference }
2334     {
2335       check .code:n =
2336       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2337     }
2338   }
2339 }
```

## reftype option

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

NOTE `tcolorbox` uses the `reftype` option to support its `label type` option when `label` is `zlabel`. Hence *don't* make any breaking changes here without previous communication.

```

2340 \tl_new:N \l__zrefclever_reftype_override_tl
2341 \keys_define:nn { zref-clever/label }
2342 {
2343   reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2344   reftype .default:n = {} ,
2345   reftype .initial:n = {} ,
2346 }

```

### 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`.

```

2347 \prop_new:N \l__zrefclever_counter_type_prop
2348 \keys_define:nn { zref-clever/label }
2349 {
2350   countertype .code:n =
2351   {
2352     \keyval_parse:nnn
2353     {
2354       \msg_warning:nnnn { zref-clever }
2355       { key-requires-value } { countertype }
2356     }
2357     {
2358       \__zrefclever_prop_put_non_empty:Nnn
2359       \l__zrefclever_counter_type_prop
2360     }
2361     {#1}
2362   } ,
2363   countertype .value_required:n = true ,
2364   countertype .initial:n =
2365   {
2366     subsection    = section ,
2367     subsubsection = section ,
2368     subparagraph  = paragraph ,
2369     enumi         = item ,
2370     enumii        = item ,
2371     enumiii       = item ,
2372     enumiv        = item ,
2373     mpfootnote   = footnote ,
2374   } ,
2375 }

```

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.

```

2376 \seq_new:N \l__zrefclever_counter_resetters_seq
2377 \keys_define:nn { zref-clever/label }
2378   {
2379     counterresetters .code:n =
2380     {
2381       \clist_map_inline:nn {#1}
2382       {
2383         \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2384         {
2385           \seq_put_right:Nn
2386             \l__zrefclever_counter_resetters_seq {##1}
2387         }
2388       }
2389     } ,
2390     counterresetters .initial:n =
2391     {
2392       part ,
2393       chapter ,
2394       section ,
2395       subsection ,
2396       subsubsection ,
2397       paragraph ,
2398       subparagraph ,
2399     },
2400     counterresetters .value_required:n = true ,
2401   }

```

### counterresetby option

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

```

2402 \prop_new:N \l__zrefclever_counter_resetby_prop
2403 \keys_define:nn { zref-clever/label }
2404 {
2405   counterresetby .code:n =
2406   {
2407     \keyval_parse:nnn
2408     {
2409       \msg_warning:nnn { zref-clever }
2410       { key-requires-value } { counterresetby }
2411     }
2412     {
2413       \__zrefclever_prop_put_non_empty:Nnn
2414       \l__zrefclever_counter_resetby_prop
2415     }
2416     {#1}
2417   } ,
2418   counterresetby .value_required:n = true ,
2419   counterresetby .initial:n =
2420   {

```

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.

```

2421     enumii = enumi ,
2422     enumiii = enumii ,
2423     enumiv = enumiii ,
2424   } ,
2425 }

```

### 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.

```

2426 \tl_new:N \l__zrefclever_current_counter_tl
2427 \keys_define:nn { zref-clever/label }
2428 {
2429   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2430   currentcounter .default:n = \@currentcounter ,
2431   currentcounter .initial:n = \@currentcounter ,
2432 }

```

### labelhook option

```

2433 \bool_new:N \l__zrefclever_labelhook_bool
2434 \keys_define:nn { zref-clever/label }
2435 {
2436   labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2437   labelhook .initial:n = true ,
2438   labelhook .default:n = true ,
2439 }

```

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 `amsmath`'s `multline` environment (and possibly elsewhere?). See <https://tex.stackexchange.com/a/402297> and <https://github.com/ho-tex/zref/issues/4>.

```

2440 \AddToHookWithArguments { label }
2441   {
2442     \bool_if:NT \l__zrefclever_labelhook_bool
2443       { \zref@wrapper@babel \zref@label {#1} }
2444   }

```

### nocompat option

```

2445 \bool_new:N \g__zrefclever_nocompat_bool
2446 \seq_new:N \g__zrefclever_nocompat_modules_seq
2447 \keys_define:nn { zref-clever/reference }
2448   {
2449     nocompat .code:n =
2450       {
2451         \tl_if_empty:nTF {#1}
2452           { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2453           {
2454             \clist_map_inline:nn {#1}
2455               {
2456                 \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2457                   {
2458                     \seq_gput_right:Nn
2459                       \g__zrefclever_nocompat_modules_seq {##1}
2460                   }
2461               }
2462           } ,
2463     } ,
2464   }
2465 \AddToHook { begindocument }
2466   {
2467     \keys_define:nn { zref-clever/reference }
2468       {
2469         nocompat .code:n =
2470           {
2471             \msg_warning:nnn { zref-clever }
2472               { option-preamble-only } { nocompat }
2473           }
2474       }
2475   }
2476 \AtEndOfPackage
2477   {
2478     \AddToHook { begindocument }
2479       {
2480         \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2481           { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2482       }
2483   }

```

`\__zrefclever_compat_module:nn` Function to be used for compatibility modules loading. It should load the module as long as `\l__zrefclever_nocompat_bool` is false and `\langle module \rangle` is not in `\l__zrefclever_-`



`nocompat_modules_seq`. The `begindocument` hook is needed so that we can have the option functional along the whole preamble, not just at package load time. This requirement might be relaxed if we made the option only available at load time, but this would not buy us much leeway anyway, since for most compatibility modules, we must test for the presence of packages at `begindocument`, only kernel features and document classes could be checked reliably before that. Besides, since we are using the new hook management system, there is always its functionality to deal with potential loading order issues.

```

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

2484 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2485 {
2486   \AddToHook { begindocument }
2487   {
2488     \bool_if:NF \g__zrefclever_nocompat_bool
2489     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2490     \seq_remove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2491   }
2492 }

```

*(End of 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`, only “not necessarily type-specific” options are pertinent here.

```

2493 \seq_map_inline:Nn
2494   \g__zrefclever_rf_opts_tl_reference_seq
2495   {
2496     \keys_define:nn { zref-clever/reference }
2497     {
2498       #1 .default:o = \c_novalue_tl ,
2499       #1 .code:n =
2500       {
2501         \tl_if_novalue:nTF {##1}
2502         {
2503           \__zrefclever_opt_tl_unset:c
2504           { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2505         }
2506         {
2507           \__zrefclever_opt_tl_set:cn
2508           { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2509           {##1}
2510         }
2511       } ,
2512     }
2513 }
2514 \keys_define:nn { zref-clever/reference }
2515 {
2516   refpre .code:n =
2517   {
2518     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.

```

```

2519     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2520     { refpre } { rebounds }
2521   } ,
2522   refpos .code:n =
2523   {
2524     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2525     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2526     { refpos } { rebounds }
2527   } ,
2528   preref .code:n =
2529   {
2530     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2531     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2532     { preref } { rebounds }
2533   } ,
2534   postref .code:n =
2535   {
2536     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2537     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2538     { postref } { rebounds }
2539   } ,
2540 }
2541 \seq_map_inline:Nn
2542 \g__zrefclever_rf_opts_seq_rebounds_seq
2543 {
2544   \keys_define:nn { zref-clever/reference }
2545   {
2546     #1 .default:o = \c_novalue_tl ,
2547     #1 .code:n =
2548     {
2549       \tl_if_novalue:nTF {##1}
2550       {
2551         \__zrefclever_opt_seq_unset:c
2552         { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2553       }
2554       {
2555         \seq_clear:N \l__zrefclever_tmpa_seq
2556         \__zrefclever_opt_seq_set_clist_split:Nn
2557         \l__zrefclever_tmpa_seq {##1}
2558         \bool_lazy_or:nnTF
2559         { \tl_if_empty_p:n {##1} }
2560         {
2561           \int_compare_p:nNn
2562           { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2563         }
2564         {
2565           \__zrefclever_opt_seq_set_eq:cN
2566           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2567           \l__zrefclever_tmpa_seq
2568         }
2569       }
2570       \msg_warning:nnee { zref-clever }
2571       { rebounds-must-be-four }
2572       {#1} { \seq_count:N \l__zrefclever_tmpa_seq }

```

```

2573         }
2574     } ,
2575 } ,
2576 }
2577 }
2578 \seq_map_inline:Nn
2579 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2580 {
2581     \keys_define:nn { zref-clever/reference }
2582     {
2583         #1 .choice: ,
2584         #1 / true .code:n =
2585         {
2586             \__zrefclever_opt_bool_set_true:c
2587             { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2588         } ,
2589         #1 / false .code:n =
2590         {
2591             \__zrefclever_opt_bool_set_false:c
2592             { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2593         } ,
2594         #1 / unset .code:n =
2595         {
2596             \__zrefclever_opt_bool_unset:c
2597             { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2598         } ,
2599         #1 .default:n = true ,
2600         no #1 .meta:n = { #1 = false } ,
2601         no #1 .value_forbidden:n = true ,
2602     }
2603 }

```

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

```

2604 \keys_define:nn { }
2605 {
2606     zref-clever/zcsetup .inherit:n =
2607     {
2608         zref-clever/label ,
2609         zref-clever/reference ,
2610     }
2611 }

```

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

```

2612 \bool_lazy_and:nnT
2613 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2614 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2615 { \msg_warning:nn { zref-clever } { load-time-options } }

```

## 5 Configuration

### 5.1 \zcsetup

\zcsetup Provide \zcsetup.

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

```

2616 \NewDocumentCommand \zcsetup { m }
2617 { \__zrefclever_zcsetup:n {#1} }

```

(End of definition for \zcsetup.)

\\_\_zrefclever\_zcsetup:n A version of \zcsetup for internal use with variant.

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

```

2618 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2619 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2620 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { e }

```

(End of 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>}

```

2621 \NewDocumentCommand \zcRefTypeSetup { m m }
2622 {
2623   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2624   \keys_set:nn { zref-clever/typesetup } {#2}
2625   \tl_clear:N \l__zrefclever_setup_type_tl
2626 }

```

(End of definition for \zcRefTypeSetup.)

```

2627 \seq_map_inline:Nn
2628 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2629 {
2630   \keys_define:nn { zref-clever/typesetup }
2631   {
2632     #1 .code:n =
2633     {

```

```

2634         \msg_warning:nnn { zref-clever }
2635         { option-not-type-specific } {#1}
2636     } ,
2637 }
2638 }
2639 \seq_map_inline:Nn
2640 \g__zrefclever_rf_opts_tl_typesetup_seq
2641 {
2642     \keys_define:nn { zref-clever/typesetup }
2643     {
2644         #1 .default:o = \c_novalue_tl ,
2645         #1 .code:n =
2646         {
2647             \tl_if_novalue:nTF {##1}
2648             {
2649                 \__zrefclever_opt_tl_unset:c
2650                 {
2651                     \__zrefclever_opt_varname_type:enn
2652                     { \l__zrefclever_setup_type_tl } {#1} { tl }
2653                 }
2654             }
2655             {
2656                 \__zrefclever_opt_tl_set:cn
2657                 {
2658                     \__zrefclever_opt_varname_type:enn
2659                     { \l__zrefclever_setup_type_tl } {#1} { tl }
2660                 }
2661                 {##1}
2662             }
2663         } ,
2664     }
2665 }
2666 \keys_define:nn { zref-clever/typesetup }
2667 {
2668     \endrange .code:n =
2669     {
2670         \str_case:nnF {#1}
2671         {
2672             { ref }
2673             {
2674                 \__zrefclever_opt_tl_clear:c
2675                 {
2676                     \__zrefclever_opt_varname_type:enn
2677                     { \l__zrefclever_setup_type_tl } { \endrangefunc } { tl }
2678                 }
2679                 \__zrefclever_opt_tl_clear:c
2680                 {
2681                     \__zrefclever_opt_varname_type:enn
2682                     { \l__zrefclever_setup_type_tl } { \endrangeprop } { tl }
2683                 }
2684             }
2685         }
2686         { stripprefix }
2687         {

```

```

2688     \_zrefclever_opt_t1_set:cn
2689     {
2690         \_zrefclever_opt_varname_type:enn
2691         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2692     }
2693     { __zrefclever_get_endrange_stripprefix }
2694     \_zrefclever_opt_t1_clear:c
2695     {
2696         \_zrefclever_opt_varname_type:enn
2697         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2698     }
2699 }
2700
2701 { pagecomp }
2702 {
2703     \_zrefclever_opt_t1_set:cn
2704     {
2705         \_zrefclever_opt_varname_type:enn
2706         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2707     }
2708     { __zrefclever_get_endrange_pagecomp }
2709     \_zrefclever_opt_t1_clear:c
2710     {
2711         \_zrefclever_opt_varname_type:enn
2712         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2713     }
2714 }
2715
2716 { pagecomp2 }
2717 {
2718     \_zrefclever_opt_t1_set:cn
2719     {
2720         \_zrefclever_opt_varname_type:enn
2721         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2722     }
2723     { __zrefclever_get_endrange_pagecomptwo }
2724     \_zrefclever_opt_t1_clear:c
2725     {
2726         \_zrefclever_opt_varname_type:enn
2727         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2728     }
2729 }
2730
2731 { unset }
2732 {
2733     \_zrefclever_opt_t1_unset:c
2734     {
2735         \_zrefclever_opt_varname_type:enn
2736         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2737     }
2738     \_zrefclever_opt_t1_unset:c
2739     {
2740         \_zrefclever_opt_varname_type:enn
2741         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }

```

```

2742     }
2743   }
2744 }
2745 {
2746   \tl_if_empty:nTF {#1}
2747   {
2748     \msg_warning:nnn { zref-clever }
2749     { endrange-property-undefined } {#1}
2750   }
2751   {
2752     \zref@ifpropundefined {#1}
2753     {
2754       \msg_warning:nnn { zref-clever }
2755       { endrange-property-undefined } {#1}
2756     }
2757     {
2758       \__zrefclever_opt_tl_set:cn
2759       {
2760         \__zrefclever_opt_varname_type:enn
2761         { \l__zrefclever_setup_type_tl }
2762         { endrangefunc } { tl }
2763       }
2764       { __zrefclever_get_endrange_property }
2765       \__zrefclever_opt_tl_set:cn
2766       {
2767         \__zrefclever_opt_varname_type:enn
2768         { \l__zrefclever_setup_type_tl }
2769         { endrangeprop } { tl }
2770       }
2771       {#1}
2772     }
2773   }
2774 }
2775 } ,
2776 endrange .value_required:n = true ,
2777 }
2778 \keys_define:nn { zref-clever/typesetup }
2779 {
2780   refpre .code:n =
2781   {
2782     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2783     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2784     { refpre } { refbounds }
2785   } ,
2786   refpos .code:n =
2787   {
2788     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2789     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2790     { refpos } { refbounds }
2791   } ,
2792   preref .code:n =
2793   {
2794     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2795     \msg_warning:nnnn { zref-clever }{ option-deprecated }

```

```

2796         { preref } { refbounds }
2797     } ,
2798     postref .code:n =
2799     {
2800         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2801         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2802         { postref } { refbounds }
2803     } ,
2804 }
2805 \seq_map_inline:Nn
2806 \g__zrefclever_rf_opts_seq_refbounds_seq
2807 {
2808     \keys_define:nm { zref-clever/typesetup }
2809     {
2810         #1 .default:o = \c_novalue_tl ,
2811         #1 .code:n =
2812         {
2813             \tl_if_novalue:nTF {##1}
2814             {
2815                 \__zrefclever_opt_seq_unset:c
2816                 {
2817                     \__zrefclever_opt_varname_type:enn
2818                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2819                 }
2820             }
2821             {
2822                 \seq_clear:N \l__zrefclever_tmpa_seq
2823                 \__zrefclever_opt_seq_set_clist_split:Nn
2824                 \l__zrefclever_tmpa_seq {##1}
2825                 \bool_lazy_or:nnTF
2826                 { \tl_if_empty_p:n {##1} }
2827                 {
2828                     \int_compare_p:nNn
2829                     { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2830                 }
2831                 {
2832                     \__zrefclever_opt_seq_set_eq:cN
2833                     {
2834                         \__zrefclever_opt_varname_type:enn
2835                         { \l__zrefclever_setup_type_tl } {#1} { seq }
2836                     }
2837                     \l__zrefclever_tmpa_seq
2838                 }
2839                 {
2840                     \msg_warning:nnee { zref-clever }
2841                     { refbounds-must-be-four }
2842                     {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2843                 }
2844             }
2845         } ,
2846     }
2847 }
2848 \seq_map_inline:Nn
2849 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq

```



```

2850 {
2851   \keys_define:nm { zref-clever/typesetup }
2852   {
2853     #1 .choice: ,
2854     #1 / true .code:n =
2855     {
2856       \__zrefclever_opt_bool_set_true:c
2857       {
2858         \__zrefclever_opt_varname_type:enn
2859         { \l__zrefclever_setup_type_tl }
2860         {#1} { bool }
2861       }
2862     } ,
2863     #1 / false .code:n =
2864     {
2865       \__zrefclever_opt_bool_set_false:c
2866       {
2867         \__zrefclever_opt_varname_type:enn
2868         { \l__zrefclever_setup_type_tl }
2869         {#1} { bool }
2870       }
2871     } ,
2872     #1 / unset .code:n =
2873     {
2874       \__zrefclever_opt_bool_unset:c
2875       {
2876         \__zrefclever_opt_varname_type:enn
2877         { \l__zrefclever_setup_type_tl }
2878         {#1} { bool }
2879       }
2880     } ,
2881     #1 .default:n = true ,
2882     no #1 .meta:n = { #1 = false } ,
2883     no #1 .value_forbidden:n = true ,
2884   }
2885 }

```

### 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 `\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>}
2886 \NewDocumentCommand \zcLanguageSetup { m m }
2887 {
2888   \group_begin:
2889   \__zrefclever_language_if_declared:nTF {#1}
2890   {

```

```

2891 \tl_clear:N \l__zrefclever_setup_type_tl
2892 \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2893 \__zrefclever_opt_seq_get:cNF
2894 {
2895   \__zrefclever_opt_varname_language:nnn
2896   {#1} { declension } { seq }
2897 }
2898 \l__zrefclever_lang_declension_seq
2899 { \seq_clear:N \l__zrefclever_lang_declension_seq }
2900 \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2901 { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2902 {
2903   \seq_get_left:NN \l__zrefclever_lang_declension_seq
2904   \l__zrefclever_lang_decl_case_tl
2905 }
2906 \__zrefclever_opt_seq_get:cNF
2907 {
2908   \__zrefclever_opt_varname_language:nnn
2909   {#1} { gender } { seq }
2910 }
2911 \l__zrefclever_lang_gender_seq
2912 { \seq_clear:N \l__zrefclever_lang_gender_seq }
2913 \keys_set:nn { zref-clever/langsetup } {#2}
2914 }
2915 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2916 \group_end:
2917 }
2918 \@onlypreamble \zcLanguageSetup

```

*(End of definition for \zcLanguageSetup.)*

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

```

2919 \keys_define:nn { zref-clever/langsetup }
2920 {
2921   type .code:n =
2922   {
2923     \tl_if_empty:nTF {#1}
2924     { \tl_clear:N \l__zrefclever_setup_type_tl }
2925     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2926   } ,
2927
2928   case .code:n =
2929   {
2930     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2931     {
2932       \msg_warning:nnee { zref-clever } { language-no-decl-setup }
2933       { \l__zrefclever_setup_language_tl } {#1}
2934     }
2935     {
2936       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2937       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2938       {
2939         \msg_warning:nnee { zref-clever } { unknown-decl-case }
2940         {#1} { \l__zrefclever_setup_language_tl }

```

```

2941         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2942         \l__zrefclever_lang_decl_case_tl
2943     }
2944 }
2945 },
2946 case .value_required:n = true ,
2947
2948 gender .value_required:n = true ,
2949 gender .code:n =
2950 {
2951     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2952     {
2953         \msg_warning:nneee { zref-clever } { language-no-gender }
2954         { \l__zrefclever_setup_language_tl } { gender } {#1}
2955     }
2956     {
2957         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2958         {
2959             \msg_warning:nnn { zref-clever }
2960             { option-only-type-specific } { gender }
2961         }
2962         {
2963             \seq_clear:N \l__zrefclever_tmpa_seq
2964             \clist_map_inline:nn {#1}
2965             {
2966                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2967                 { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
2968                 {
2969                     \msg_warning:nnee { zref-clever }
2970                     { gender-not-declared }
2971                     { \l__zrefclever_setup_language_tl } {##1}
2972                 }
2973             }
2974             \__zrefclever_opt_seq_gset_eq:cN
2975             {
2976                 \__zrefclever_opt_varname_lang_type:eenn
2977                 { \l__zrefclever_setup_language_tl }
2978                 { \l__zrefclever_setup_type_tl }
2979                 { gender }
2980                 { seq }
2981             }
2982             \l__zrefclever_tmpa_seq
2983         }
2984     }
2985 },
2986 }
2987 \seq_map_inline:Nn
2988 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2989 {
2990     \keys_define:nn { zref-clever/langsetup }
2991     {
2992         #1 .value_required:n = true ,
2993         #1 .code:n =
2994         {

```

```

2995         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2996         {
2997             \__zrefclever_opt_tl_gset:cn
2998             {
2999                 \__zrefclever_opt_varname_lang_default:enn
3000                 { \l__zrefclever_setup_language_tl } {#1} { t1 }
3001             }
3002             {##1}
3003         }
3004         {
3005             \msg_warning:nnn { zref-clever }
3006             { option-not-type-specific } {#1}
3007         }
3008     } ,
3009 }
3010 }
3011 \seq_map_inline:Nn
3012 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
3013 {
3014     \keys_define:nn { zref-clever/langsetup }
3015     {
3016         #1 .value_required:n = true ,
3017         #1 .code:n =
3018         {
3019             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3020             {
3021                 \__zrefclever_opt_tl_gset:cn
3022                 {
3023                     \__zrefclever_opt_varname_lang_default:enn
3024                     { \l__zrefclever_setup_language_tl } {#1} { t1 }
3025                 }
3026                 {##1}
3027             }
3028             {
3029                 \__zrefclever_opt_tl_gset:cn
3030                 {
3031                     \__zrefclever_opt_varname_lang_type:eenn
3032                     { \l__zrefclever_setup_language_tl }
3033                     { \l__zrefclever_setup_type_tl }
3034                     {#1} { t1 }
3035                 }
3036                 {##1}
3037             }
3038         } ,
3039     }
3040 }
3041 \keys_define:nn { zref-clever/langsetup }
3042 {
3043     endrange .value_required:n = true ,
3044     endrange .code:n =
3045     {
3046         \str_case:nnF {#1}
3047         {
3048             { ref }

```

```

3049 {
3050   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3051   {
3052     \__zrefclever_opt_tl_gclear:c
3053     {
3054       \__zrefclever_opt_varname_lang_default:enn
3055       { \l__zrefclever_setup_language_tl }
3056       { endrangefunc } { tl }
3057     }
3058     \__zrefclever_opt_tl_gclear:c
3059     {
3060       \__zrefclever_opt_varname_lang_default:enn
3061       { \l__zrefclever_setup_language_tl }
3062       { endrangeprop } { tl }
3063     }
3064   }
3065   {
3066     \__zrefclever_opt_tl_gclear:c
3067     {
3068       \__zrefclever_opt_varname_lang_type:eenn
3069       { \l__zrefclever_setup_language_tl }
3070       { \l__zrefclever_setup_type_tl }
3071       { endrangefunc } { tl }
3072     }
3073     \__zrefclever_opt_tl_gclear:c
3074     {
3075       \__zrefclever_opt_varname_lang_type:eenn
3076       { \l__zrefclever_setup_language_tl }
3077       { \l__zrefclever_setup_type_tl }
3078       { endrangeprop } { tl }
3079     }
3080   }
3081 }
3082
3083 { stripprefix }
3084 {
3085   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3086   {
3087     \__zrefclever_opt_tl_gset:cn
3088     {
3089       \__zrefclever_opt_varname_lang_default:enn
3090       { \l__zrefclever_setup_language_tl }
3091       { endrangefunc } { tl }
3092     }
3093     { __zrefclever_get_endrange_stripprefix }
3094     \__zrefclever_opt_tl_gclear:c
3095     {
3096       \__zrefclever_opt_varname_lang_default:enn
3097       { \l__zrefclever_setup_language_tl }
3098       { endrangeprop } { tl }
3099     }
3100   }
3101   {
3102     \__zrefclever_opt_tl_gset:cn

```

```

3103         {
3104             \__zrefclever_opt_varname_lang_type:eenn
3105             { \l__zrefclever_setup_language_tl }
3106             { \l__zrefclever_setup_type_tl }
3107             { endrangefunc } { tl }
3108         }
3109         { __zrefclever_get_endrange_stripprefix }
3110     \__zrefclever_opt_tl_gclear:c
3111     {
3112         \__zrefclever_opt_varname_lang_type:eenn
3113         { \l__zrefclever_setup_language_tl }
3114         { \l__zrefclever_setup_type_tl }
3115         { endrangeprop } { tl }
3116     }
3117 }
3118 }
3119
3120 { pagecomp }
3121 {
3122     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3123     {
3124         \__zrefclever_opt_tl_gset:cn
3125         {
3126             \__zrefclever_opt_varname_lang_default:enn
3127             { \l__zrefclever_setup_language_tl }
3128             { endrangefunc } { tl }
3129         }
3130         { __zrefclever_get_endrange_pagecomp }
3131         \__zrefclever_opt_tl_gclear:c
3132         {
3133             \__zrefclever_opt_varname_lang_default:enn
3134             { \l__zrefclever_setup_language_tl }
3135             { endrangeprop } { tl }
3136         }
3137     }
3138     {
3139         \__zrefclever_opt_tl_gset:cn
3140         {
3141             \__zrefclever_opt_varname_lang_type:eenn
3142             { \l__zrefclever_setup_language_tl }
3143             { \l__zrefclever_setup_type_tl }
3144             { endrangefunc } { tl }
3145         }
3146         { __zrefclever_get_endrange_pagecomp }
3147         \__zrefclever_opt_tl_gclear:c
3148         {
3149             \__zrefclever_opt_varname_lang_type:eenn
3150             { \l__zrefclever_setup_language_tl }
3151             { \l__zrefclever_setup_type_tl }
3152             { endrangeprop } { tl }
3153         }
3154     }
3155 }
3156

```

```

3157 { pagecomp2 }
3158 {
3159   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3160   {
3161     \__zrefclever_opt_tl_gset:cn
3162     {
3163       \__zrefclever_opt_varname_lang_default:enn
3164       { \l__zrefclever_setup_language_tl }
3165       { endrangefunc } { tl }
3166     }
3167     { __zrefclever_get_endrange_pagecomptwo }
3168     \__zrefclever_opt_tl_gclear:c
3169     {
3170       \__zrefclever_opt_varname_lang_default:enn
3171       { \l__zrefclever_setup_language_tl }
3172       { endrangeprop } { tl }
3173     }
3174   }
3175   {
3176     \__zrefclever_opt_tl_gset:cn
3177     {
3178       \__zrefclever_opt_varname_lang_type:eenn
3179       { \l__zrefclever_setup_language_tl }
3180       { \l__zrefclever_setup_type_tl }
3181       { endrangefunc } { tl }
3182     }
3183     { __zrefclever_get_endrange_pagecomptwo }
3184     \__zrefclever_opt_tl_gclear:c
3185     {
3186       \__zrefclever_opt_varname_lang_type:eenn
3187       { \l__zrefclever_setup_language_tl }
3188       { \l__zrefclever_setup_type_tl }
3189       { endrangeprop } { tl }
3190     }
3191   }
3192 }
3193 }
3194 {
3195   \tl_if_empty:nTF {#1}
3196   {
3197     \msg_warning:nnn { zref-clever }
3198     { endrange-property-undefined } {#1}
3199   }
3200   {
3201     \zref@ifpropundefined {#1}
3202     {
3203       \msg_warning:nnn { zref-clever }
3204       { endrange-property-undefined } {#1}
3205     }
3206     {
3207       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3208       {
3209         \__zrefclever_opt_tl_gset:cn
3210         {

```

```

3211         \_zrefclever_opt_varname_lang_default:enn
3212         { \l__zrefclever_setup_language_tl }
3213         { endrangefunc } { tl }
3214     }
3215     { __zrefclever_get_endrange_property }
3216 \_zrefclever_opt_tl_gset:cn
3217     {
3218         \_zrefclever_opt_varname_lang_default:enn
3219         { \l__zrefclever_setup_language_tl }
3220         { endrangeprop } { tl }
3221     }
3222     {#1}
3223 }
3224 {
3225     \_zrefclever_opt_tl_gset:cn
3226     {
3227         \_zrefclever_opt_varname_lang_type:eenn
3228         { \l__zrefclever_setup_language_tl }
3229         { \l__zrefclever_setup_type_tl }
3230         { endrangefunc } { tl }
3231     }
3232     { __zrefclever_get_endrange_property }
3233 \_zrefclever_opt_tl_gset:cn
3234     {
3235         \_zrefclever_opt_varname_lang_type:eenn
3236         { \l__zrefclever_setup_language_tl }
3237         { \l__zrefclever_setup_type_tl }
3238         { endrangeprop } { tl }
3239     }
3240     {#1}
3241     }
3242 }
3243 }
3244 }
3245 } ,
3246 }
3247 \keys_define:nn { zref-clever/langsetup }
3248 {
3249     refpre .code:n =
3250     {
3251         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3252         \msg_warning:nmmm { zref-clever }{ option-deprecated }
3253         { refpre } { refbounds }
3254     } ,
3255     refpos .code:n =
3256     {
3257         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3258         \msg_warning:nmmm { zref-clever }{ option-deprecated }
3259         { refpos } { refbounds }
3260     } ,
3261     preref .code:n =
3262     {
3263         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3264         \msg_warning:nmmm { zref-clever }{ option-deprecated }

```



```

3265     { preref } { refbounds }
3266   } ,
3267   postref .code:n =
3268   {
3269     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3270     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3271     { postref } { refbounds }
3272   } ,
3273 }
3274 \seq_map_inline:Nn
3275 \g__zrefclever_rf_opts_tl_type_names_seq
3276 {
3277   \keys_define:nn { zref-clever/langsetup }
3278   {
3279     #1 .value_required:n = true ,
3280     #1 .code:n =
3281     {
3282       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3283       {
3284         \msg_warning:nnn { zref-clever }
3285         { option-only-type-specific } {#1}
3286       }
3287       {
3288         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3289         {
3290           \__zrefclever_opt_tl_gset:cn
3291           {
3292             \__zrefclever_opt_varname_lang_type:een
3293             { \l__zrefclever_setup_language_tl }
3294             { \l__zrefclever_setup_type_tl }
3295             {#1} { tl }
3296           }
3297           {##1}
3298         }
3299         {
3300           \__zrefclever_opt_tl_gset:cn
3301           {
3302             \__zrefclever_opt_varname_lang_type:een
3303             { \l__zrefclever_setup_language_tl }
3304             { \l__zrefclever_setup_type_tl }
3305             { \l__zrefclever_lang_decl_case_tl - #1 }
3306             { tl }
3307           }
3308           {##1}
3309         }
3310       }
3311     } ,
3312   }
3313 }
3314 \seq_map_inline:Nn
3315 \g__zrefclever_rf_opts_seq_refbounds_seq
3316 {
3317   \keys_define:nn { zref-clever/langsetup }
3318   {

```

```

3319 #1 .value_required:n = true ,
3320 #1 .code:n =
3321 {
3322   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3323   {
3324     \seq_gclear:N \g__zrefclever_tmpa_seq
3325     \__zrefclever_opt_seq_gset_clist_split:Nn
3326     \g__zrefclever_tmpa_seq {##1}
3327     \bool_lazy_or:nnTF
3328     { \tl_if_empty_p:n {##1} }
3329     {
3330       \int_compare_p:nNn
3331       { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3332     }
3333     {
3334       \__zrefclever_opt_seq_gset_eq:cN
3335       {
3336         \__zrefclever_opt_varname_lang_default:enn
3337         { \l__zrefclever_setup_language_tl }
3338         {##1} { seq }
3339       }
3340       \g__zrefclever_tmpa_seq
3341     }
3342     {
3343       \msg_warning:nnee { zref-clever }
3344       { refbounds-must-be-four }
3345       {##1} { \seq_count:N \g__zrefclever_tmpa_seq }
3346     }
3347   }
3348   {
3349     \seq_gclear:N \g__zrefclever_tmpa_seq
3350     \__zrefclever_opt_seq_gset_clist_split:Nn
3351     \g__zrefclever_tmpa_seq {##1}
3352     \bool_lazy_or:nnTF
3353     { \tl_if_empty_p:n {##1} }
3354     {
3355       \int_compare_p:nNn
3356       { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3357     }
3358     {
3359       \__zrefclever_opt_seq_gset_eq:cN
3360       {
3361         \__zrefclever_opt_varname_lang_type:enn
3362         { \l__zrefclever_setup_language_tl }
3363         { \l__zrefclever_setup_type_tl } {##1} { seq }
3364       }
3365       \g__zrefclever_tmpa_seq
3366     }
3367     {
3368       \msg_warning:nnee { zref-clever }
3369       { refbounds-must-be-four }
3370       {##1} { \seq_count:N \g__zrefclever_tmpa_seq }
3371     }
3372   }

```

```

3373     } ,
3374   }
3375 }
3376 \seq_map_inline:Nn
3377   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3378   {
3379   \keys_define:nn { zref-clever/langsetup }
3380   {
3381     #1 .choice: ,
3382     #1 / true .code:n =
3383     {
3384       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3385       {
3386         \__zrefclever_opt_bool_gset_true:c
3387         {
3388           \__zrefclever_opt_varname_lang_default:enn
3389           { \l__zrefclever_setup_language_tl }
3390           {#1} { bool }
3391         }
3392       }
3393       {
3394         \__zrefclever_opt_bool_gset_true:c
3395         {
3396           \__zrefclever_opt_varname_lang_type:eenn
3397           { \l__zrefclever_setup_language_tl }
3398           { \l__zrefclever_setup_type_tl }
3399           {#1} { bool }
3400         }
3401       }
3402     } ,
3403     #1 / false .code:n =
3404     {
3405       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3406       {
3407         \__zrefclever_opt_bool_gset_false:c
3408         {
3409           \__zrefclever_opt_varname_lang_default:enn
3410           { \l__zrefclever_setup_language_tl }
3411           {#1} { bool }
3412         }
3413       }
3414       {
3415         \__zrefclever_opt_bool_gset_false:c
3416         {
3417           \__zrefclever_opt_varname_lang_type:eenn
3418           { \l__zrefclever_setup_language_tl }
3419           { \l__zrefclever_setup_type_tl }
3420           {#1} { bool }
3421         }
3422       }
3423     } ,
3424     #1 .default:n = true ,
3425     no #1 .meta:n = { #1 = false } ,
3426     no #1 .value_forbidden:n = true ,

```

```

3427     }
3428 }

```

## 6 User interface

### 6.1 `\zcref`

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

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

```

3429 \NewDocumentCommand \zcref { s O { } m }
3430 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

*(End of 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}
```

```

3431 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3432 {
3433   \group_begin:

```

Set options.

```
3434   \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

3435   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3436   \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:e` does nothing if the language file is already loaded.

```
3437   \__zrefclever_provide_langfile:e { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3438   \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```

3439   \bool_lazy_and:nnT
3440     { \l__zrefclever_zrefcheck_available_bool }
3441     { \l__zrefclever_zcref_with_check_bool }
3442     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

3443   \bool_lazy_or:nnT
3444     { \l__zrefclever_typeset_sort_bool }
3445     { \l__zrefclever_typeset_range_bool }
3446     { \__zrefclever_sort_labels: }

```

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

```

3447     \group_begin:
3448     \l__zrefclever_ref_typeset_font_tl
3449     \__zrefclever_typeset_refs:
3450     \group_end:

```

Typeset note.

```

3451     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3452     {
3453         \__zrefclever_get_rf_opt_tl:neeN { noteseq }
3454         { \l__zrefclever_label_type_a_tl }
3455         { \l__zrefclever_ref_language_tl }
3456         \l__zrefclever_tmpa_tl
3457         \l__zrefclever_tmpa_tl
3458         \l__zrefclever_zcref_note_tl
3459     }

```

Integration with zref-check.

```

3460     \bool_lazy_and:nnT
3461     { \l__zrefclever_zrefcheck_available_bool }
3462     { \l__zrefclever_zcref_with_check_bool }
3463     {
3464         \zrefcheck_zcref_end_label_maybe:
3465         \zrefcheck_zcref_run_checks_on_labels:n
3466         { \l__zrefclever_zcref_labels_seq }
3467     }

```

Integration with mathtools.

```

3468     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3469     {
3470         \__zrefclever_mathtools_showonlyrefs:n
3471         { \l__zrefclever_zcref_labels_seq }
3472     }
3473     \group_end:
3474 }

```

*(End of definition for \\_\_zrefclever\_zcref:nmnn.)*

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3475 \seq_new:N \l__zrefclever_zcref_labels_seq
3476 \bool_new:N \l__zrefclever_link_star_bool

```

*(End of definition for \l\_\_zrefclever\_zcref\_labels\_seq and \l\_\_zrefclever\_link\_star\_bool.)*

## 6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```

\zcpageref*[\options]{\labels}

3477 \NewDocumentCommand \zcpageref { s O { } m }
3478 {
3479     \group_begin:
3480     \IfBooleanT {#1}

```

```

3481     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3482     \zcref [#2, ref = page] {#3}
3483     \group_end:
3484 }

```

*(End of 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.

<pre> \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl \l__zrefclever_label_enclval_a_tl \l__zrefclever_label_enclval_b_tl \l__zrefclever_label_extdoc_a_tl \l__zrefclever_label_extdoc_b_tl </pre>	<pre> 3485 \tl_new:N \l__zrefclever_label_type_a_tl 3486 \tl_new:N \l__zrefclever_label_type_b_tl 3487 \tl_new:N \l__zrefclever_label_enclval_a_tl 3488 \tl_new:N \l__zrefclever_label_enclval_b_tl 3489 \tl_new:N \l__zrefclever_label_extdoc_a_tl 3490 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre>	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p>
--	--	--

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

<pre> \l__zrefclever_sort_decided_bool </pre>	<pre> 3491 \bool_new:N \l__zrefclever_sort_decided_bool </pre>	<p>Auxiliary variable for <code>\__zrefclever_sort_default_same_type:nn</code>, signals if the sorting between two labels has been decided or not.</p>
---	--	--

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

<pre> \l__zrefclever_sort_prior_a_int \l__zrefclever_sort_prior_b_int </pre>	<pre> 3492 \int_new:N \l__zrefclever_sort_prior_a_int 3493 \int_new:N \l__zrefclever_sort_prior_b_int </pre>	<p>Auxiliary variables for <code>\__zrefclever_sort_default_different_types:nn</code>. Store the sort priority of the “current” and “next” labels.</p>
--	--	--

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

<pre> \l__zrefclever_label_types_seq </pre>	<pre> 3494 \seq_new:N \l__zrefclever_label_types_seq </pre>	<p>Stores the order in which reference types appear in the label list supplied by the user in <code>\zcref</code>. This variable is populated by <code>\__zrefclever_label_type_put_new_right:n</code> at the start of <code>\__zrefclever_sort_labels:.</code> This order is required as a “last resort” sort criterion between the reference types, for use in <code>\__zrefclever_sort_default_different_types:nn</code>.</p>
---	---	--

*(End of 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.

```
3495 \cs_new_protected:Npn \__zrefclever_sort_labels:
3496 {
```

Store label types sequence.

```
3497   \seq_clear:N \l__zrefclever_label_types_seq
3498   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
3499   {
3500     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3501     \__zrefclever_label_type_put_new_right:n
3502   }
```

Sort.

```
3503   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3504   {
3505     \zref@ifrefundefined {##1}
3506     {
3507       \zref@ifrefundefined {##2}
3508       {
3509         % Neither label is defined.
3510         \sort_return_same:
3511       }
3512       {
3513         % The second label is defined, but the first isn't, leave the
3514         % undefined first (to be more visible).
3515         \sort_return_same:
3516       }
3517     }
3518     {
3519       \zref@ifrefundefined {##2}
3520       {
3521         % The first label is defined, but the second isn't, bring the
3522         % second forward.
3523         \sort_return_swapped:
3524       }
3525       {
3526         % The interesting case: both labels are defined. References
3527         % to the "default" property or to the "page" are quite
3528         % different with regard to sorting, so we branch them here to
3529         % specialized functions.
3530         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3531         { \__zrefclever_sort_page:nn {##1} {##2} }
3532         { \__zrefclever_sort_default:nn {##1} {##2} }
3533       }
3534     }
3535   }
3536 }
```

*(End of definition for `\__zrefclever_sort_labels:.`)*

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

3537 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3538 {
3539   \__zrefclever_extract_default:Nnnn
3540   \l__zrefclever_label_type_a_tl {#1} {zc@type} { }
3541   \seq_if_in:NVF \l__zrefclever_label_types_seq
3542   \l__zrefclever_label_type_a_tl
3543   {
3544     \seq_put_right:NV \l__zrefclever_label_types_seq
3545     \l__zrefclever_label_type_a_tl
3546   }
3547 }

```

(End of definition for `\_zrefclever_label_type_put_new_right:n`.)

`\_zrefclever_sort_default:mn` 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:mn {<label a>} {<label b>}

3548 \cs_new_protected:Npn \__zrefclever_sort_default:mn #1#2
3549 {
3550   \__zrefclever_extract_default:Nnnn
3551   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3552   \__zrefclever_extract_default:Nnnn
3553   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3554
3555   \tl_if_eq:NNTF
3556   \l__zrefclever_label_type_a_tl
3557   \l__zrefclever_label_type_b_tl
3558   { \__zrefclever_sort_default_same_type:mn {#1} {#2} }
3559   { \__zrefclever_sort_default_different_types:mn {#1} {#2} }
3560 }

```

(End of definition for `\_zrefclever_sort_default:mn`.)

```

\_zrefclever_sort_default_same_type:mn

\__zrefclever_sort_default_same_type:mn {<label a>} {<label b>}

3561 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:mn #1#2
3562 {
3563   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3564   {#1} {zc@enclval} { }

```



```

3565 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3566 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3567 {#2} { zc@enclval } { }
3568 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3569 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3570 {#1} { externaldocument } { }
3571 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3572 {#2} { externaldocument } { }
3573
3574 \bool_set_false:N \l__zrefclever_sort_decided_bool
3575
3576 % First we check if there's any "external document" difference (coming
3577 % from 'zref-xr') and, if so, sort based on that.
3578 \tl_if_eq:NMF
3579 \l__zrefclever_label_extdoc_a_tl
3580 \l__zrefclever_label_extdoc_b_tl
3581 {
3582   \bool_if:nTF
3583   {
3584     \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3585     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3586   }
3587   {
3588     \bool_set_true:N \l__zrefclever_sort_decided_bool
3589     \sort_return_same:
3590   }
3591   {
3592     \bool_if:nTF
3593     {
3594       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3595       \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3596     }
3597     {
3598       \bool_set_true:N \l__zrefclever_sort_decided_bool
3599       \sort_return_swapped:
3600     }
3601     {
3602       \bool_set_true:N \l__zrefclever_sort_decided_bool
3603       % Two different "external documents": last resort, sort by the
3604       % document name itself.
3605       \str_compare:eNeTF
3606       { \l__zrefclever_label_extdoc_b_tl } <
3607       { \l__zrefclever_label_extdoc_a_tl }
3608       { \sort_return_swapped: }
3609       { \sort_return_same: }
3610     }
3611   }
3612 }
3613
3614 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3615 {
3616   \bool_if:nTF
3617   {
3618     % Both are empty: neither label has any (further) "enclosing

```

```

3619 % counters" (left).
3620 \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3621 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3622 }
3623 {
3624 \bool_set_true:N \l__zrefclever_sort_decided_bool
3625 \int_compare:nNnTF
3626 { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3627 >
3628 { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3629 { \sort_return_swapped: }
3630 { \sort_return_same: }
3631 }
3632 {
3633 \bool_if:nTF
3634 {
3635 % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
3636 \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3637 }
3638 {
3639 \bool_set_true:N \l__zrefclever_sort_decided_bool
3640 \int_compare:nNnTF
3641 { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3642 >
3643 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3644 { \sort_return_swapped: }
3645 { \sort_return_same: }
3646 }
3647 {
3648 \bool_if:nTF
3649 {
3650 % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3651 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3652 }
3653 {
3654 \bool_set_true:N \l__zrefclever_sort_decided_bool
3655 \int_compare:nNnTF
3656 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3657 <
3658 { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3659 { \sort_return_same: }
3660 { \sort_return_swapped: }
3661 }
3662 {
3663 % Neither is empty: we can compare the values of the
3664 % current enclosing counter in the loop, if they are
3665 % equal, we are still in the loop, if they are not, a
3666 % sorting decision can be made directly.
3667 \int_compare:nNnTF
3668 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3669 =
3670 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3671 {
3672 \tl_set:Ne \l__zrefclever_label_enclval_a_tl

```

```

3673         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3674         \tl_set:Ne \l__zrefclever_label_enclval_b_tl
3675         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3676     }
3677     {
3678     \bool_set_true:N \l__zrefclever_sort_decided_bool
3679     \int_compare:nNnTF
3680     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3681     >
3682     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3683     { \sort_return_swapped: }
3684     { \sort_return_same: }
3685     }
3686     }
3687   }
3688 }
3689 }
3690 }

```

(End of definition for `\__zrefclever_sort_default_same_type:nn`.)

`\__zrefclever_sort_default_different_types:nn`

```

\__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}

```

```

3691 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3692 {

```

Retrieve sort priorities for *<label a>* and *<label b>*. `\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”.

```

3693   \int_zero:N \l__zrefclever_sort_prior_a_int
3694   \int_zero:N \l__zrefclever_sort_prior_b_int
3695   \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3696   {
3697     \tl_if_eq:nnTF {##2} {{othertypes}}
3698     {
3699       \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3700       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3701       \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3702       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3703     }
3704     {
3705       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3706       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3707       {
3708         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3709         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3710       }
3711     }
3712   }

```

Then do the actual sorting.

```

3713   \bool_if:nTF
3714   {
3715     \int_compare_p:nNn
3716     { \l__zrefclever_sort_prior_a_int } <

```

```

3717     { \l__zrefclever_sort_prior_b_int }
3718   }
3719   { \sort_return_same: }
3720   {
3721     \bool_if:nTF
3722     {
3723       \int_compare_p:nNn
3724       { \l__zrefclever_sort_prior_a_int } >
3725       { \l__zrefclever_sort_prior_b_int }
3726     }
3727     { \sort_return_swapped: }
3728     {
3729       % Sort priorities are equal: the type that occurs first in
3730       % ‘labels’, as given by the user, is kept (or brought) forward.
3731       \seq_map_inline:Nn \l__zrefclever_label_types_seq
3732       {
3733         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3734         { \seq_map_break:n { \sort_return_same: } }
3735         {
3736           \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3737           { \seq_map_break:n { \sort_return_swapped: } }
3738         }
3739       }
3740     }
3741   }
3742 }

```

(End of 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>}
3743 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3744 {
3745   \int_compare:nNnTF
3746   { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3747   >
3748   { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3749   { \sort_return_swapped: }
3750   { \sort_return_same: }
3751 }

```

(End of definition for `\__zrefclever_sort_page:nn`.)

## 8 Typesetting

“Typesetting” the reference, which here includes the parsing of the labels and eventual compression of labels in sequence into ranges, is definitely the “crux” of `zref-clever`. This

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

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

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

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

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

`\l__zrefclever_next_is_same_bool` when their values are actually equal.

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

## Variables

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

```

\l_zrefclever_typeset_last_bool 3752 \seq_new:N \l__zrefclever_typeset_labels_seq
\l_zrefclever_last_of_type_bool 3753 \bool_new:N \l_zrefclever_typeset_last_bool
3754 \bool_new:N \l__zrefclever_last_of_type_bool

```

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

`\l_zrefclever_type_count_int` Auxiliary variables for `\__zrefclever_typeset_refs`: main counters.

```

\l_zrefclever_label_count_int 3755 \int_new:N \l_zrefclever_type_count_int
\l__zrefclever_ref_count_int 3756 \int_new:N \l_zrefclever_label_count_int
3757 \int_new:N \l__zrefclever_ref_count_int

```

*(End of definition for `\l_zrefclever_type_count_int`, `\l_zrefclever_label_count_int`, and `\l__zrefclever_ref_count_int`.)*

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

```

\l__zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl 3758 \tl_new:N \l__zrefclever_label_a_tl
\l_zrefclever_typeset_queue_curr_tl 3759 \tl_new:N \l__zrefclever_label_b_tl
\l_zrefclever_type_first_label_tl 3760 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
\l_zrefclever_type_first_label_type_tl 3761 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
3762 \tl_new:N \l__zrefclever_type_first_label_tl
3763 \tl_new:N \l__zrefclever_type_first_label_type_tl

```

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

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

```

\l_zrefclever_name_in_link_bool 3764 \tl_new:N \l__zrefclever_type_name_tl
\l__zrefclever_type_name_missing_bool 3765 \bool_new:N \l__zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl 3766 \bool_new:N \l__zrefclever_type_name_missing_bool
\l_zrefclever_name_format_fallback_tl 3767 \tl_new:N \l__zrefclever_name_format_tl
\l__zrefclever_type_name_gender_seq 3768 \tl_new:N \l__zrefclever_name_format_fallback_tl
3769 \seq_new:N \l__zrefclever_type_name_gender_seq

```

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

`\l_zrefclever_range_count_int` Auxiliary variables for `\_zrefclever_typeset_refs`: range handling.  
`\l_zrefclever_range_same_count_int` 3770 `\int_new:N \l_zrefclever_range_count_int`  
`\l_zrefclever_range_beg_label_tl` 3771 `\int_new:N \l_zrefclever_range_same_count_int`  
`\l_zrefclever_range_beg_is_first_bool` 3772 `\tl_new:N \l_zrefclever_range_beg_label_tl`  
`\l_zrefclever_range_end_ref_tl` 3773 `\bool_new:N \l_zrefclever_range_beg_is_first_bool`  
`\l_zrefclever_next_maybe_range_bool` 3774 `\tl_new:N \l_zrefclever_range_end_ref_tl`  
`\l_zrefclever_next_is_same_bool` 3775 `\bool_new:N \l_zrefclever_next_maybe_range_bool`  
3776 `\bool_new:N \l_zrefclever_next_is_same_bool`

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

`\l_zrefclever_tpairsep_tl` Auxiliary variables for `\_zrefclever_typeset_refs`: separators, and font and other  
`\l_zrefclever_tlistsep_tl` options.  
`\l_zrefclever_tlastsep_tl` 3777 `\tl_new:N \l_zrefclever_tpairsep_tl`  
`\l_zrefclever_namesep_tl` 3778 `\tl_new:N \l_zrefclever_tlistsep_tl`  
`\l_zrefclever_pairsep_tl` 3779 `\tl_new:N \l_zrefclever_tlastsep_tl`  
`\l_zrefclever_listsep_tl` 3780 `\tl_new:N \l_zrefclever_namesep_tl`  
`\l_zrefclever_lastsep_tl` 3781 `\tl_new:N \l_zrefclever_pairsep_tl`  
`\l_zrefclever_rangeseptl` 3782 `\tl_new:N \l_zrefclever_listsep_tl`  
`\l_zrefclever_namefont_tl` 3783 `\tl_new:N \l_zrefclever_lastsep_tl`  
`\l_zrefclever_reffont_tl` 3784 `\tl_new:N \l_zrefclever_rangeseptl`  
`\l_zrefclever_endrangefunc_tl` 3785 `\tl_new:N \l_zrefclever_namefont_tl`  
`\l_zrefclever_endrangeprop_tl` 3786 `\tl_new:N \l_zrefclever_reffont_tl`  
`\l_zrefclever_cap_bool` 3787 `\tl_new:N \l_zrefclever_endrangefunc_tl`  
`\l_zrefclever_abbrev_bool` 3788 `\tl_new:N \l_zrefclever_endrangeprop_tl`  
`\l_zrefclever_rangetopair_bool` 3789 `\bool_new:N \l_zrefclever_cap_bool`  
3790 `\bool_new:N \l_zrefclever_abbrev_bool`  
3791 `\bool_new:N \l_zrefclever_rangetopair_bool`

*(End of definition for `\l_zrefclever_tpairsep_tl` and others.)*

`\l_zrefclever_refbounds_first_seq` Auxiliary variables for `\_zrefclever_typeset_refs::` advanced reference format op-  
`\l_zrefclever_refbounds_first_sg_seq` tions.  
`\l_zrefclever_refbounds_first_pb_seq` 3792 `\seq_new:N \l_zrefclever_refbounds_first_seq`  
`\l_zrefclever_refbounds_first_rb_seq` 3793 `\seq_new:N \l_zrefclever_refbounds_first_sg_seq`  
`\l_zrefclever_refbounds_mid_seq` 3794 `\seq_new:N \l_zrefclever_refbounds_first_pb_seq`  
`\l_zrefclever_refbounds_mid_rb_seq` 3795 `\seq_new:N \l_zrefclever_refbounds_first_rb_seq`  
`\l_zrefclever_refbounds_mid_re_seq` 3796 `\seq_new:N \l_zrefclever_refbounds_mid_seq`  
`\l_zrefclever_refbounds_last_seq` 3797 `\seq_new:N \l_zrefclever_refbounds_mid_rb_seq`  
`\l_zrefclever_refbounds_last_pe_seq` 3798 `\seq_new:N \l_zrefclever_refbounds_mid_re_seq`  
`\l_zrefclever_refbounds_last_re_seq` 3799 `\seq_new:N \l_zrefclever_refbounds_last_seq`  
`\l_zrefclever_type_first_refbounds_seq` 3800 `\seq_new:N \l_zrefclever_refbounds_last_pe_seq`  
`\l_zrefclever_type_first_refbounds_set_bool` 3801 `\seq_new:N \l_zrefclever_refbounds_last_re_seq`  
3802 `\seq_new:N \l_zrefclever_type_first_refbounds_seq`  
3803 `\bool_new:N \l_zrefclever_type_first_refbounds_set_bool`

*(End of definition for `\l_zrefclever_refbounds_first_seq` and others.)*

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

3804 `\bool_new:N \l_zrefclever_verbose_testing_bool`

*(End of definition for `\l_zrefclever_verbose_testing_bool`.)*

## Main functions

`\_zrefclever_typeset_refs:` Main typesetting function for `\zcref`.

```
3805 \cs_new_protected:Npn \_zrefclever_typeset_refs:
3806 {
3807   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3808     \l__zrefclever_zcref_labels_seq
3809   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3810   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3811   \tl_clear:N \l__zrefclever_type_first_label_tl
3812   \tl_clear:N \l__zrefclever_type_first_label_type_tl
3813   \tl_clear:N \l__zrefclever_range_beg_label_tl
3814   \tl_clear:N \l__zrefclever_range_end_ref_tl
3815   \int_zero:N \l__zrefclever_label_count_int
3816   \int_zero:N \l__zrefclever_type_count_int
3817   \int_zero:N \l__zrefclever_ref_count_int
3818   \int_zero:N \l__zrefclever_range_count_int
3819   \int_zero:N \l__zrefclever_range_same_count_int
3820   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3821   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3822
3823   % Get type block options (not type-specific).
3824   \_zrefclever_get_rf_opt_tl:neeN { tpairsep }
3825     { \l__zrefclever_label_type_a_tl }
3826     { \l__zrefclever_ref_language_tl }
3827     \l__zrefclever_tpairsep_tl
3828   \_zrefclever_get_rf_opt_tl:neeN { tlistsep }
3829     { \l__zrefclever_label_type_a_tl }
3830     { \l__zrefclever_ref_language_tl }
3831     \l__zrefclever_tlistsep_tl
3832   \_zrefclever_get_rf_opt_tl:neeN { tlastsep }
3833     { \l__zrefclever_label_type_a_tl }
3834     { \l__zrefclever_ref_language_tl }
3835     \l__zrefclever_tlastsep_tl
3836
3837   % Process label stack.
3838   \bool_set_false:N \l__zrefclever_typeset_last_bool
3839   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3840     {
3841       \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3842         \l__zrefclever_label_a_tl
3843       \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3844         {
3845           \tl_clear:N \l__zrefclever_label_b_tl
3846           \bool_set_true:N \l__zrefclever_typeset_last_bool
3847         }
3848         {
3849           \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3850             \l__zrefclever_label_b_tl
3851         }
3852
3853       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3854         {
3855           \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
```



```

3856     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3857   }
3858   {
3859     \__zrefclever_extract_default:NVnn
3860     \l__zrefclever_label_type_a_tl
3861     \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3862     \__zrefclever_extract_default:NVnn
3863     \l__zrefclever_label_type_b_tl
3864     \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3865   }
3866
3867   % First, we establish whether the "current label" (i.e. 'a') is the
3868   % last one of its type. This can happen because the "next label"
3869   % (i.e. 'b') is of a different type (or different definition status),
3870   % or because we are at the end of the list.
3871   \bool_if:NTF \l__zrefclever_typeset_last_bool
3872     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3873     {
3874       \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3875         {
3876           \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3877             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3878             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3879         }
3880         {
3881           \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3882             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3883             {
3884               % Neither is undefined, we must check the types.
3885               \tl_if_eq:NNTF
3886                 \l__zrefclever_label_type_a_tl
3887                 \l__zrefclever_label_type_b_tl
3888                 { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3889                 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3890             }
3891         }
3892     }
3893
3894   % Handle warnings in case of reference or type undefined.
3895   % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3896   \zref@refused { \l__zrefclever_label_a_tl }
3897   % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3898   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3899     {}
3900     {
3901       \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3902         {
3903           \msg_warning:nne { zref-clever } { missing-type }
3904           { \l__zrefclever_label_a_tl }
3905         }
3906       \zref@ifrefcontainsprop
3907         { \l__zrefclever_label_a_tl }
3908         { \l__zrefclever_ref_property_tl }
3909         { }

```

```

3910     {
3911     \msg_warning:nnee { zref-clever } { missing-property }
3912     { \l__zrefclever_ref_property_tl }
3913     { \l__zrefclever_label_a_tl }
3914     }
3915   }
3916
3917   % Get possibly type-specific separators, rebounds, font and other
3918   % options, once per type.
3919   \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3920   {
3921     \__zrefclever_get_rf_opt_tl:neeN { namesep }
3922     { \l__zrefclever_label_type_a_tl }
3923     { \l__zrefclever_ref_language_tl }
3924     \l__zrefclever_namesep_tl
3925     \__zrefclever_get_rf_opt_tl:neeN { pairsep }
3926     { \l__zrefclever_label_type_a_tl }
3927     { \l__zrefclever_ref_language_tl }
3928     \l__zrefclever_pairsep_tl
3929     \__zrefclever_get_rf_opt_tl:neeN { listsep }
3930     { \l__zrefclever_label_type_a_tl }
3931     { \l__zrefclever_ref_language_tl }
3932     \l__zrefclever_listsep_tl
3933     \__zrefclever_get_rf_opt_tl:neeN { lastsep }
3934     { \l__zrefclever_label_type_a_tl }
3935     { \l__zrefclever_ref_language_tl }
3936     \l__zrefclever_lastsep_tl
3937     \__zrefclever_get_rf_opt_tl:neeN { rangesep }
3938     { \l__zrefclever_label_type_a_tl }
3939     { \l__zrefclever_ref_language_tl }
3940     \l__zrefclever_rangesep_tl
3941     \__zrefclever_get_rf_opt_tl:neeN { namefont }
3942     { \l__zrefclever_label_type_a_tl }
3943     { \l__zrefclever_ref_language_tl }
3944     \l__zrefclever_namefont_tl
3945     \__zrefclever_get_rf_opt_tl:neeN { reffont }
3946     { \l__zrefclever_label_type_a_tl }
3947     { \l__zrefclever_ref_language_tl }
3948     \l__zrefclever_reffont_tl
3949     \__zrefclever_get_rf_opt_tl:neeN { endrangefunc }
3950     { \l__zrefclever_label_type_a_tl }
3951     { \l__zrefclever_ref_language_tl }
3952     \l__zrefclever_endrangefunc_tl
3953     \__zrefclever_get_rf_opt_tl:neeN { endrangeprop }
3954     { \l__zrefclever_label_type_a_tl }
3955     { \l__zrefclever_ref_language_tl }
3956     \l__zrefclever_endrangeprop_tl
3957     \__zrefclever_get_rf_opt_bool:nneeN { cap } { false }
3958     { \l__zrefclever_label_type_a_tl }
3959     { \l__zrefclever_ref_language_tl }
3960     \l__zrefclever_cap_bool
3961     \__zrefclever_get_rf_opt_bool:nneeN { abbrev } { false }
3962     { \l__zrefclever_label_type_a_tl }
3963     { \l__zrefclever_ref_language_tl }

```

```

3964         \l__zrefclever_abbrev_bool
3965     \__zrefclever_get_rf_opt_bool:neeN { rangetopair } { true }
3966     { \l__zrefclever_label_type_a_tl }
3967     { \l__zrefclever_ref_language_tl }
3968     \l__zrefclever_rangetopair_bool
3969     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first }
3970     { \l__zrefclever_label_type_a_tl }
3971     { \l__zrefclever_ref_language_tl }
3972     \l__zrefclever_refbounds_first_seq
3973     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-sg }
3974     { \l__zrefclever_label_type_a_tl }
3975     { \l__zrefclever_ref_language_tl }
3976     \l__zrefclever_refbounds_first_sg_seq
3977     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-pb }
3978     { \l__zrefclever_label_type_a_tl }
3979     { \l__zrefclever_ref_language_tl }
3980     \l__zrefclever_refbounds_first_pb_seq
3981     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-rb }
3982     { \l__zrefclever_label_type_a_tl }
3983     { \l__zrefclever_ref_language_tl }
3984     \l__zrefclever_refbounds_first_rb_seq
3985     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid }
3986     { \l__zrefclever_label_type_a_tl }
3987     { \l__zrefclever_ref_language_tl }
3988     \l__zrefclever_refbounds_mid_seq
3989     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-rb }
3990     { \l__zrefclever_label_type_a_tl }
3991     { \l__zrefclever_ref_language_tl }
3992     \l__zrefclever_refbounds_mid_rb_seq
3993     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-re }
3994     { \l__zrefclever_label_type_a_tl }
3995     { \l__zrefclever_ref_language_tl }
3996     \l__zrefclever_refbounds_mid_re_seq
3997     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last }
3998     { \l__zrefclever_label_type_a_tl }
3999     { \l__zrefclever_ref_language_tl }
4000     \l__zrefclever_refbounds_last_seq
4001     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-pe }
4002     { \l__zrefclever_label_type_a_tl }
4003     { \l__zrefclever_ref_language_tl }
4004     \l__zrefclever_refbounds_last_pe_seq
4005     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-re }
4006     { \l__zrefclever_label_type_a_tl }
4007     { \l__zrefclever_ref_language_tl }
4008     \l__zrefclever_refbounds_last_re_seq
4009     }
4010
4011     % Here we send this to a couple of auxiliary functions.
4012     \bool_if:NTF \l__zrefclever_last_of_type_bool
4013     % There exists no next label of the same type as the current.
4014     { \__zrefclever_typeset_refs_last_of_type: }
4015     % There exists a next label of the same type as the current.
4016     { \__zrefclever_typeset_refs_not_last_of_type: }
4017 }

```

```
4018 }
```

(End of 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.

```
4019 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
4020 {
4021   % Process the current label to the current queue.
4022   \int_case:nnF { \l__zrefclever_label_count_int }
4023   {
4024     % It is the last label of its type, but also the first one, and that's
4025     % what matters here: just store it.
4026     % Test: 'zc-typeset01.lvt': "Last of type: single"
4027     { 0 }
4028     {
4029       \tl_set:NV \l__zrefclever_type_first_label_tl
4030       \l__zrefclever_label_a_tl
4031       \tl_set:NV \l__zrefclever_type_first_label_type_tl
4032       \l__zrefclever_label_type_a_tl
4033       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4034       \l__zrefclever_refbounds_first_sg_seq
4035       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4036     }
4037
4038     % The last is the second: we have a pair (if not repeated).
4039     % Test: 'zc-typeset01.lvt': "Last of type: pair"
4040     { 1 }
4041     {
4042       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4043       {
4044         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4045         \l__zrefclever_refbounds_first_sg_seq
4046         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4047       }
4048       {
4049         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4050         {
4051           \exp_not:V \l__zrefclever_pairsep_tl
4052           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4053           \l__zrefclever_refbounds_last_pe_seq
4054         }
4055         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4056         \l__zrefclever_refbounds_first_pb_seq
4057         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4058       }
4059     }
4060   }
4061 }
```

```

4059     }
4060   }
4061   % Last is third or more of its type: without repetition, we'd have the
4062   % last element on a list, but control for possible repetition.
4063   {
4064     \int_case:nNF { \l__zrefclever_range_count_int }
4065     {
4066       % There was no range going on.
4067       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4068       { 0 }
4069       {
4070         \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4071         {
4072           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4073           {
4074             \exp_not:V \l__zrefclever_pairsep_tl
4075             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4076             \l__zrefclever_refbounds_last_pe_seq
4077           }
4078         }
4079         {
4080           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4081           {
4082             \exp_not:V \l__zrefclever_lastsep_tl
4083             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4084             \l__zrefclever_refbounds_last_seq
4085           }
4086         }
4087       }
4088       % Last in the range is also the second in it.
4089       % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4090       { 1 }
4091       {
4092         \int_compare:nNnTF
4093         { \l__zrefclever_range_same_count_int } = { 1 }
4094         {
4095           % We know 'range_beg_is_first_bool' is false, since this is
4096           % the second element in the range, but the third or more in
4097           % the type list.
4098           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4099           {
4100             \exp_not:V \l__zrefclever_pairsep_tl
4101             \__zrefclever_get_ref:VN
4102             \l__zrefclever_range_beg_label_tl
4103             \l__zrefclever_refbounds_last_pe_seq
4104           }
4105           \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4106           \l__zrefclever_refbounds_first_pb_seq
4107           \bool_set_true:N
4108           \l__zrefclever_type_first_refbounds_set_bool
4109         }
4110         {
4111           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4112           {

```

```

4113         \exp_not:V \l__zrefclever_listsep_tl
4114         \__zrefclever_get_ref:VN
4115         \l__zrefclever_range_beg_label_tl
4116         \l__zrefclever_refbounds_mid_seq
4117         \exp_not:V \l__zrefclever_lastsep_tl
4118         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4119         \l__zrefclever_refbounds_last_seq
4120     }
4121 }
4122 }
4123 }
4124 % Last in the range is third or more in it.
4125 {
4126     \int_case:nnF
4127     {
4128         \l__zrefclever_range_count_int -
4129         \l__zrefclever_range_same_count_int
4130     }
4131     {
4132         % Repetition, not a range.
4133         % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4134         { 0 }
4135         {
4136             % If 'range_beg_is_first_bool' is true, it means it was also
4137             % the first of the type, and hence its typesetting was
4138             % already handled, and we just have to set refbounds.
4139             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4140             {
4141                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4142                 \l__zrefclever_refbounds_first_sg_seq
4143                 \bool_set_true:N
4144                 \l__zrefclever_type_first_refbounds_set_bool
4145             }
4146             {
4147                 \int_compare:nNnTF
4148                 { \l__zrefclever_ref_count_int } < { 2 }
4149                 {
4150                     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4151                     {
4152                         \exp_not:V \l__zrefclever_pairsep_tl
4153                         \__zrefclever_get_ref:VN
4154                         \l__zrefclever_range_beg_label_tl
4155                         \l__zrefclever_refbounds_last_pe_seq
4156                     }
4157                 }
4158                 {
4159                     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4160                     {
4161                         \exp_not:V \l__zrefclever_lastsep_tl
4162                         \__zrefclever_get_ref:VN
4163                         \l__zrefclever_range_beg_label_tl
4164                         \l__zrefclever_refbounds_last_seq
4165                     }
4166                 }

```

```

4167     }
4168   }
4169   % A 'range', but with no skipped value, treat as pair if range
4170   % started with first of type, otherwise as list.
4171   % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4172   { 1 }
4173   {
4174     % Ditto.
4175     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4176     {
4177       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4178       \l__zrefclever_refbounds_first_pb_seq
4179       \bool_set_true:N
4180       \l__zrefclever_type_first_refbounds_set_bool
4181       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4182       {
4183         \exp_not:V \l__zrefclever_pairsep_tl
4184         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4185         \l__zrefclever_refbounds_last_pe_seq
4186       }
4187     }
4188     {
4189       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4190       {
4191         \exp_not:V \l__zrefclever_listsep_tl
4192         \__zrefclever_get_ref:VN
4193         \l__zrefclever_range_beg_label_tl
4194         \l__zrefclever_refbounds_mid_seq
4195       }
4196       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4197       {
4198         \exp_not:V \l__zrefclever_lastsep_tl
4199         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4200         \l__zrefclever_refbounds_last_seq
4201       }
4202     }
4203   }
4204 }
4205 {
4206 % An actual range.
4207 % Test: 'zc-typeset01.lvt': "Last of type: range"
4208 % Ditto.
4209 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4210 {
4211   \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4212   \l__zrefclever_refbounds_first_rb_seq
4213   \bool_set_true:N
4214   \l__zrefclever_type_first_refbounds_set_bool
4215 }
4216 {
4217   \int_compare:nNnTF
4218   { \l__zrefclever_ref_count_int } < { 2 }
4219   {
4220     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl

```

```

4221         {
4222             \exp_not:V \l__zrefclever_pairsep_tl
4223             \__zrefclever_get_ref:VN
4224             \l__zrefclever_range_beg_label_tl
4225             \l__zrefclever_refbounds_mid_rb_seq
4226         }
4227         \seq_set_eq:NN
4228         \l__zrefclever_type_first_refbounds_seq
4229         \l__zrefclever_refbounds_first_pb_seq
4230         \bool_set_true:N
4231         \l__zrefclever_type_first_refbounds_set_bool
4232     }
4233     {
4234         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4235         {
4236             \exp_not:V \l__zrefclever_lastsep_tl
4237             \__zrefclever_get_ref:VN
4238             \l__zrefclever_range_beg_label_tl
4239             \l__zrefclever_refbounds_mid_rb_seq
4240         }
4241     }
4242 }
4243 \bool_lazy_and:nnTF
4244 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4245 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4246 {
4247     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4248     \l__zrefclever_range_beg_label_tl
4249     \l__zrefclever_label_a_tl
4250     \l__zrefclever_range_end_ref_tl
4251     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4252     {
4253         \exp_not:V \l__zrefclever_rangesep_tl
4254         \__zrefclever_get_ref_endrange:VVN
4255         \l__zrefclever_label_a_tl
4256         \l__zrefclever_range_end_ref_tl
4257         \l__zrefclever_refbounds_last_re_seq
4258     }
4259 }
4260 {
4261     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4262     {
4263         \exp_not:V \l__zrefclever_rangesep_tl
4264         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4265         \l__zrefclever_refbounds_last_re_seq
4266     }
4267 }
4268 }
4269 }
4270 }
4271
4272 % Handle "range" option. The idea is simple: if the queue is not empty,
4273 % we replace it with the end of the range (or pair). We can still
4274 % retrieve the end of the range from 'label_a' since we know to be

```



```

4275 % processing the last label of its type at this point.
4276 \bool_if:NT \l__zrefclever_typeset_range_bool
4277 {
4278   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4279   {
4280     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4281     { }
4282     {
4283       \msg_warning:nne { zref-clever } { single-element-range }
4284       { \l__zrefclever_type_first_label_type_tl }
4285     }
4286   }
4287 {
4288   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4289   \bool_if:NT \l__zrefclever_rangetopair_bool
4290   {
4291     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4292     { }
4293     {
4294       \__zrefclever_labels_in_sequence:nn
4295       { \l__zrefclever_type_first_label_tl }
4296       { \l__zrefclever_label_a_tl }
4297     }
4298   }
4299   % Test: 'zc-typeset01.lvt': "Last of type: option range"
4300   % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4301   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4302   {
4303     \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4304     {
4305       \exp_not:V \l__zrefclever_pairsep_tl
4306       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4307       \l__zrefclever_refbounds_last_pe_seq
4308     }
4309     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4310     \l__zrefclever_refbounds_first_pb_seq
4311     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4312   }
4313   {
4314     \bool_lazy_and:nnTF
4315     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4316     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VWN } }
4317     {
4318       % We must get 'type_first_label_tl' instead of
4319       % 'range_beg_label_tl' here, since it is not necessary
4320       % that the first of type was actually starting a range for
4321       % the 'range' option to be used.
4322       \use:c { \l__zrefclever_endrangefunc_tl :VWN }
4323       \l__zrefclever_type_first_label_tl
4324       \l__zrefclever_label_a_tl
4325       \l__zrefclever_range_end_ref_tl
4326       \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4327       {
4328         \exp_not:V \l__zrefclever_rangeseq_tl

```

```

4329         \_zrefclever_get_ref_endrange:VVN
4330         \l__zrefclever_label_a_tl
4331         \l__zrefclever_range_end_ref_tl
4332         \l__zrefclever_refbounds_last_re_seq
4333     }
4334 }
4335 {
4336     \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4337     {
4338         \exp_not:V \l__zrefclever_rangeseq_tl
4339         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4340         \l__zrefclever_refbounds_last_re_seq
4341     }
4342 }
4343 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4344 \l__zrefclever_refbounds_first_rb_seq
4345 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4346 }
4347 }
4348 }
4349
4350 % If none of the special cases for the first of type refbounds have been
4351 % set, do it.
4352 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4353 {
4354     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4355     \l__zrefclever_refbounds_first_seq
4356 }
4357
4358 % Now that the type block is finished, we can add the name and the first
4359 % ref to the queue. Also, if "typeset" option is not "both", handle it
4360 % here as well.
4361 \_zrefclever_type_name_setup:
4362 \bool_if:nTF
4363 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4364 {
4365     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4366     { \_zrefclever_get_ref_first: }
4367 }
4368 {
4369     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4370     {
4371         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4372         \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4373         {
4374             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4375             \l__zrefclever_type_first_refbounds_seq
4376         }
4377     }
4378     {
4379         \bool_if:NTF \l__zrefclever_typeset_name_bool
4380         {
4381             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4382             \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl

```

```

4383     {
4384         \bool_if:NTF \l__zrefclever_name_in_link_bool
4385         {
4386             \exp_not:N \group_begin:
4387             \exp_not:V \l__zrefclever_namefont_tl
4388             \__zrefclever_hyperlink:nnn
4389             {
4390                 \__zrefclever_extract_url_unexp:V
4391                 \l__zrefclever_type_first_label_tl
4392             }
4393             {
4394                 \__zrefclever_extract_unexp:Vnn
4395                 \l__zrefclever_type_first_label_tl
4396                 { anchor } { }
4397             }
4398             { \exp_not:V \l__zrefclever_type_name_tl }
4399             \exp_not:N \group_end:
4400         }
4401         {
4402             \exp_not:N \group_begin:
4403             \exp_not:V \l__zrefclever_namefont_tl
4404             \exp_not:V \l__zrefclever_type_name_tl
4405             \exp_not:N \group_end:
4406         }
4407     }
4408 }
4409 {
4410     % Logically, this case would correspond to "typeset=none", but
4411     % it should not occur, given that the options are set up to
4412     % typeset either "ref" or "name". Still, leave here a
4413     % sensible fallback, equal to the behavior of "both".
4414     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4415     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4416     { \__zrefclever_get_ref_first: }
4417 }
4418 }
4419 }
4420
4421 % Typeset the previous type block, if there is one.
4422 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4423 {
4424     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4425     { \l__zrefclever_tlistsep_tl }
4426     \l__zrefclever_typeset_queue_prev_tl
4427 }
4428
4429 % Extra log for testing.
4430 \bool_if:NT \l__zrefclever_verbose_testing_bool
4431 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4432
4433 % Wrap up loop, or prepare for next iteration.
4434 \bool_if:NTF \l__zrefclever_typeset_last_bool
4435 {
4436     % We are finishing, typeset the current queue.

```

```

4437 \int_case:nnF { \l__zrefclever_type_count_int }
4438 {
4439   % Single type.
4440   % Test: 'zc-typeset01.lvt': "Last of type: single type"
4441   { 0 }
4442   { \l__zrefclever_typeset_queue_curr_tl }
4443   % Pair of types.
4444   % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4445   { 1 }
4446   {
4447     \l__zrefclever_tpairsep_tl
4448     \l__zrefclever_typeset_queue_curr_tl
4449   }
4450 }
4451 {
4452   % Last in list of types.
4453   % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4454   \l__zrefclever_tlastsep_tl
4455   \l__zrefclever_typeset_queue_curr_tl
4456 }
4457 % And nudge in case of multitype reference.
4458 \bool_lazy_all:nT
4459 {
4460   { \l__zrefclever_nudge_enabled_bool }
4461   { \l__zrefclever_nudge_multitype_bool }
4462   { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4463 }
4464 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4465 }
4466 {
4467   % There are further labels, set variables for next iteration.
4468   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4469     \l__zrefclever_typeset_queue_curr_tl
4470   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4471   \tl_clear:N \l__zrefclever_type_first_label_tl
4472   \tl_clear:N \l__zrefclever_type_first_label_type_tl
4473   \tl_clear:N \l__zrefclever_range_beg_label_tl
4474   \tl_clear:N \l__zrefclever_range_end_ref_tl
4475   \int_zero:N \l__zrefclever_label_count_int
4476   \int_zero:N \l__zrefclever_ref_count_int
4477   \int_incr:N \l__zrefclever_type_count_int
4478   \int_zero:N \l__zrefclever_range_count_int
4479   \int_zero:N \l__zrefclever_range_same_count_int
4480   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4481   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4482 }
4483 }

```

(End of 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.

```

4484 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4485 {
4486   % Signal if next label may form a range with the current one (only

```

```

4487 % considered if compression is enabled in the first place).
4488 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4489 \bool_set_false:N \l__zrefclever_next_is_same_bool
4490 \bool_if:NT \l__zrefclever_typeset_compress_bool
4491 {
4492   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4493   { }
4494   {
4495     \__zrefclever_labels_in_sequence:mn
4496     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
4497   }
4498 }
4499
4500 % Process the current label to the current queue.
4501 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4502 {
4503   % Current label is the first of its type (also not the last, but it
4504   % doesn't matter here): just store the label.
4505   \tl_set:NV \l__zrefclever_type_first_label_tl
4506   \l__zrefclever_label_a_tl
4507   \tl_set:NV \l__zrefclever_type_first_label_type_tl
4508   \l__zrefclever_label_type_a_tl
4509   \int_incr:N \l__zrefclever_ref_count_int
4510
4511   % If the next label may be part of a range, signal it (we deal with it
4512   % as the "first", and must do it there, to handle hyperlinking), but
4513   % also step the range counters.
4514   % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4515   \bool_if:NT \l__zrefclever_next_maybe_range_bool
4516   {
4517     \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4518     \tl_set:NV \l__zrefclever_range_beg_label_tl
4519     \l__zrefclever_label_a_tl
4520     \tl_clear:N \l__zrefclever_range_end_ref_tl
4521     \int_incr:N \l__zrefclever_range_count_int
4522     \bool_if:NT \l__zrefclever_next_is_same_bool
4523     { \int_incr:N \l__zrefclever_range_same_count_int }
4524   }
4525 }
4526 {
4527   % Current label is neither the first (nor the last) of its type.
4528   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4529   {
4530     % Starting, or continuing a range.
4531     \int_compare:nNnTF
4532     { \l__zrefclever_range_count_int } = { 0 }
4533     {
4534       % There was no range going, we are starting one.
4535       \tl_set:NV \l__zrefclever_range_beg_label_tl
4536       \l__zrefclever_label_a_tl
4537       \tl_clear:N \l__zrefclever_range_end_ref_tl
4538       \int_incr:N \l__zrefclever_range_count_int
4539       \bool_if:NT \l__zrefclever_next_is_same_bool
4540       { \int_incr:N \l__zrefclever_range_same_count_int }

```

```

4541     }
4542     {
4543     % Second or more in the range, but not the last.
4544     \int_incr:N \l__zrefclever_range_count_int
4545     \bool_if:NT \l__zrefclever_next_is_same_bool
4546     { \int_incr:N \l__zrefclever_range_same_count_int }
4547     }
4548   }
4549   {
4550   % Next element is not in sequence: there was no range, or we are
4551   % closing one.
4552   \int_case:nnF { \l__zrefclever_range_count_int }
4553   {
4554     % There was no range going on.
4555     % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4556     { 0 }
4557     {
4558       \int_incr:N \l__zrefclever_ref_count_int
4559       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4560       {
4561         \exp_not:V \l__zrefclever_listsep_tl
4562         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4563         \l__zrefclever_refbounds_mid_seq
4564       }
4565     }
4566     % Last is second in the range: if 'range_same_count' is also
4567     % '1', it's a repetition (drop it), otherwise, it's a "pair
4568     % within a list", treat as list.
4569     % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4570     % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4571     { 1 }
4572     {
4573       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4574       {
4575         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4576         \l__zrefclever_refbounds_first_seq
4577         \bool_set_true:N
4578         \l__zrefclever_type_first_refbounds_set_bool
4579       }
4580       {
4581         \int_incr:N \l__zrefclever_ref_count_int
4582         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4583         {
4584           \exp_not:V \l__zrefclever_listsep_tl
4585           \__zrefclever_get_ref:VN
4586           \l__zrefclever_range_beg_label_tl
4587           \l__zrefclever_refbounds_mid_seq
4588         }
4589       }
4590     \int_compare:nNnF
4591     { \l__zrefclever_range_same_count_int } = { 1 }
4592     {
4593       \int_incr:N \l__zrefclever_ref_count_int
4594       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl

```

```

4595         {
4596             \exp_not:V \l__zrefclever_listsep_tl
4597             \__zrefclever_get_ref:VN
4598             \l__zrefclever_label_a_tl
4599             \l__zrefclever_refbounds_mid_seq
4600         }
4601     }
4602 }
4603 }
4604 {
4605     % Last is third or more in the range: if 'range_count' and
4606     % 'range_same_count' are the same, its a repetition (drop it),
4607     % if they differ by '1', its a list, if they differ by more,
4608     % it is a real range.
4609     \int_case:nnF
4610     {
4611         \l__zrefclever_range_count_int -
4612         \l__zrefclever_range_same_count_int
4613     }
4614     {
4615         % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4616         { 0 }
4617         {
4618             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4619             {
4620                 \seq_set_eq:NN
4621                 \l__zrefclever_type_first_refbounds_seq
4622                 \l__zrefclever_refbounds_first_seq
4623                 \bool_set_true:N
4624                 \l__zrefclever_type_first_refbounds_set_bool
4625             }
4626             {
4627                 \int_incr:N \l__zrefclever_ref_count_int
4628                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4629                 {
4630                     \exp_not:V \l__zrefclever_listsep_tl
4631                     \__zrefclever_get_ref:VN
4632                     \l__zrefclever_range_beg_label_tl
4633                     \l__zrefclever_refbounds_mid_seq
4634                 }
4635             }
4636         }
4637         % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4638         { 1 }
4639         {
4640             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4641             {
4642                 \seq_set_eq:NN
4643                 \l__zrefclever_type_first_refbounds_seq
4644                 \l__zrefclever_refbounds_first_seq
4645                 \bool_set_true:N
4646                 \l__zrefclever_type_first_refbounds_set_bool
4647             }
4648             {

```

```

4649         \int_incr:N \l__zrefclever_ref_count_int
4650         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4651         {
4652             \exp_not:V \l__zrefclever_listsep_tl
4653             \__zrefclever_get_ref:VN
4654             \l__zrefclever_range_beg_label_tl
4655             \l__zrefclever_refbounds_mid_seq
4656         }
4657     }
4658     \int_incr:N \l__zrefclever_ref_count_int
4659     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4660     {
4661         \exp_not:V \l__zrefclever_listsep_tl
4662         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4663         \l__zrefclever_refbounds_mid_seq
4664     }
4665 }
4666 }
4667 {
4668 % Test: 'zc-typeset01.lvt': "Not last of type: range"
4669 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4670 {
4671     \seq_set_eq:NN
4672     \l__zrefclever_type_first_refbounds_seq
4673     \l__zrefclever_refbounds_first_rb_seq
4674     \bool_set_true:N
4675     \l__zrefclever_type_first_refbounds_set_bool
4676 }
4677 {
4678     \int_incr:N \l__zrefclever_ref_count_int
4679     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4680     {
4681         \exp_not:V \l__zrefclever_listsep_tl
4682         \__zrefclever_get_ref:VN
4683         \l__zrefclever_range_beg_label_tl
4684         \l__zrefclever_refbounds_mid_rb_seq
4685     }
4686 }
4687 % For the purposes of the serial comma, and thus for the
4688 % distinction of 'lastsep' and 'pairsep', a "range" counts
4689 % as one. Since 'range_beg' has already been counted
4690 % (here or with the first of type), we refrain from
4691 % incrementing 'ref_count_int'.
4692 \bool_lazy_and:nnTF
4693 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4694 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4695 {
4696     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4697     \l__zrefclever_range_beg_label_tl
4698     \l__zrefclever_label_a_tl
4699     \l__zrefclever_range_end_ref_tl
4700     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4701     {
4702         \exp_not:V \l__zrefclever_rangeseq_tl

```



```

4703         \zrefclever_get_ref_endrange:V\N
4704         \l_zrefclever_label_a_tl
4705         \l_zrefclever_range_end_ref_tl
4706         \l_zrefclever_refbounds_mid_re_seq
4707     }
4708 }
4709 {
4710     \tl_put_right:Ne \l_zrefclever_typeset_queue_curr_tl
4711     {
4712         \exp_not:V \l_zrefclever_rangeseq_tl
4713         \zrefclever_get_ref:VN \l_zrefclever_label_a_tl
4714         \l_zrefclever_refbounds_mid_re_seq
4715     }
4716 }
4717 }
4718 }
4719 % We just closed a range, reset 'range_beg_is_first' in case a
4720 % second range for the same type occurs, in which case its
4721 % 'range_beg' will no longer be 'first'.
4722 \bool_set_false:N \l_zrefclever_range_beg_is_first_bool
4723 % Reset counters.
4724 \int_zero:N \l_zrefclever_range_count_int
4725 \int_zero:N \l_zrefclever_range_same_count_int
4726 }
4727 }
4728 % Step label counter for next iteration.
4729 \int_incr:N \l_zrefclever_label_count_int
4730 }

```

(End of 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  $\TeX$ nic 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.

```
4731 \cs_new_protected:Npn \_zrefclever_ref_default:
4732   { \zref@default }
4733 \cs_new_protected:Npn \_zrefclever_name_default:
4734   { \zref@default }
```

*(End of 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>}
```

```
4735 \cs_new:Npn \_zrefclever_get_ref:nN #1#2
4736   {
4737     \zref@ifrefcontainsprop {#1} { \l_zrefclever_ref_property_tl }
4738     {
4739       \bool_if:nTF
4740         {
4741           \l_zrefclever_hyperlink_bool &&
4742           ! \l_zrefclever_link_star_bool
4743         }
4744         {
4745           \seq_item:Nn #2 { 1 }
4746           \_zrefclever_hyperlink:nnn
4747             { \_zrefclever_extract_url_unexp:n {#1} }
4748             { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4749             {
4750               \seq_item:Nn #2 { 2 }
4751               \exp_not:N \group_begin:
4752               \exp_not:V \l_zrefclever_reffont_tl
4753               \_zrefclever_extract_unexp:nvn {#1}
4754                 { \l_zrefclever_ref_property_tl } { }
4755               \exp_not:N \group_end:
4756               \seq_item:Nn #2 { 3 }
4757             }
4758           \seq_item:Nn #2 { 4 }
4759         }
4760         {
4761           \seq_item:Nn #2 { 1 }
4762           \seq_item:Nn #2 { 2 }
4763           \exp_not:N \group_begin:
4764           \exp_not:V \l_zrefclever_reffont_tl
4765           \_zrefclever_extract_unexp:nvn {#1}
4766             { \l_zrefclever_ref_property_tl } { }
4767           \exp_not:N \group_end:
4768           \seq_item:Nn #2 { 3 }
4769           \seq_item:Nn #2 { 4 }
4770         }
4771     }
```

```

4771     }
4772     { \_zrefclever_ref_default: }
4773   }
4774 \cs_generate_variant:Nn \_zrefclever_get_ref:nN { VN }

```

(End of definition for \\_zrefclever\_get\_ref:nN.)

```

\_zrefclever_get_ref_endrange:nnN      \_zrefclever_get_ref_endrange:nnN {<label>} {<reference>} {<refbounds>}
4775 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3
4776 {
4777   \str_if_eq:nnTF {#2} { zc@missingproperty }
4778   { \_zrefclever_ref_default: }
4779   {
4780     \bool_if:nTF
4781     {
4782       \l_zrefclever_hyperlink_bool &&
4783       ! \l_zrefclever_link_star_bool
4784     }
4785     {
4786       \seq_item:Nn #3 { 1 }
4787       \_zrefclever_hyperlink:nnn
4788       { \_zrefclever_extract_url_unexp:n {#1} }
4789       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4790       {
4791         \seq_item:Nn #3 { 2 }
4792         \exp_not:N \group_begin:
4793         \exp_not:V \l_zrefclever_reffont_tl
4794         \exp_not:n {#2}
4795         \exp_not:N \group_end:
4796         \seq_item:Nn #3 { 3 }
4797       }
4798       \seq_item:Nn #3 { 4 }
4799     }
4800     {
4801       \seq_item:Nn #3 { 1 }
4802       \seq_item:Nn #3 { 2 }
4803       \exp_not:N \group_begin:
4804       \exp_not:V \l_zrefclever_reffont_tl
4805       \exp_not:n {#2}
4806       \exp_not:N \group_end:
4807       \seq_item:Nn #3 { 3 }
4808       \seq_item:Nn #3 { 4 }
4809     }
4810   }
4811 }
4812 \cs_generate_variant:Nn \_zrefclever_get_ref_endrange:nnN { VVN }

```

(End of 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.

```

4813 \cs_new:Npn \__zrefclever_get_ref_first:
4814 {
4815   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4816   { \__zrefclever_ref_default: }
4817   {
4818     \bool_if:NTF \l__zrefclever_name_in_link_bool
4819     {
4820       \zref@ifrefcontainsprop
4821       { \l__zrefclever_type_first_label_tl }
4822       { \l__zrefclever_ref_property_tl }
4823       {
4824         \__zrefclever_hyperlink:nnn
4825         {
4826           \__zrefclever_extract_url_unexp:V
4827           \l__zrefclever_type_first_label_tl
4828         }
4829         {
4830           \__zrefclever_extract_unexp:Vnn
4831           \l__zrefclever_type_first_label_tl { anchor } { }
4832         }
4833         {
4834           \exp_not:N \group_begin:
4835           \exp_not:V \l__zrefclever_namefont_tl
4836           \exp_not:V \l__zrefclever_type_name_tl
4837           \exp_not:N \group_end:
4838           \exp_not:V \l__zrefclever_namesep_tl
4839           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4840           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4841           \exp_not:N \group_begin:
4842           \exp_not:V \l__zrefclever_reffont_tl
4843           \__zrefclever_extract_unexp:Vvn
4844           \l__zrefclever_type_first_label_tl
4845           { \l__zrefclever_ref_property_tl } { }
4846           \exp_not:N \group_end:
4847           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4848         }
4849         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4850       }
4851     }
4852     \exp_not:N \group_begin:
4853     \exp_not:V \l__zrefclever_namefont_tl
4854     \exp_not:V \l__zrefclever_type_name_tl
4855     \exp_not:N \group_end:
4856     \exp_not:V \l__zrefclever_namesep_tl
4857     \__zrefclever_ref_default:
4858   }
4859 }
4860 {
4861   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4862   {
4863     \__zrefclever_name_default:

```

```

4864         \exp_not:V \l__zrefclever_namesep_tl
4865     }
4866     {
4867         \exp_not:N \group_begin:
4868         \exp_not:V \l__zrefclever_namefont_tl
4869         \exp_not:V \l__zrefclever_type_name_tl
4870         \exp_not:N \group_end:
4871         \tl_if_empty:NF \l__zrefclever_type_name_tl
4872             { \exp_not:V \l__zrefclever_namesep_tl }
4873     }
4874 \zref@ifrefcontainsprop
4875 { \l__zrefclever_type_first_label_tl }
4876 { \l__zrefclever_ref_property_tl }
4877 {
4878     \bool_if:nTF
4879     {
4880         \l__zrefclever_hyperlink_bool &&
4881         ! \l__zrefclever_link_star_bool
4882     }
4883     {
4884         \seq_item:Nn
4885             \l__zrefclever_type_first_refbounds_seq { 1 }
4886         \__zrefclever_hyperlink:nnn
4887         {
4888             \__zrefclever_extract_url_unexp:V
4889             \l__zrefclever_type_first_label_tl
4890         }
4891         {
4892             \__zrefclever_extract_unexp:Vnn
4893             \l__zrefclever_type_first_label_tl { anchor } { }
4894         }
4895         {
4896             \seq_item:Nn
4897                 \l__zrefclever_type_first_refbounds_seq { 2 }
4898             \exp_not:N \group_begin:
4899             \exp_not:V \l__zrefclever_reffont_tl
4900             \__zrefclever_extract_unexp:Vvn
4901             \l__zrefclever_type_first_label_tl
4902             { \l__zrefclever_ref_property_tl } { }
4903             \exp_not:N \group_end:
4904             \seq_item:Nn
4905                 \l__zrefclever_type_first_refbounds_seq { 3 }
4906         }
4907         \seq_item:Nn
4908             \l__zrefclever_type_first_refbounds_seq { 4 }
4909     }
4910     {
4911         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4912         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4913         \exp_not:N \group_begin:
4914         \exp_not:V \l__zrefclever_reffont_tl
4915         \__zrefclever_extract_unexp:Vvn
4916         \l__zrefclever_type_first_label_tl
4917         { \l__zrefclever_ref_property_tl } { }

```

```

4918         \exp_not:N \group_end:
4919         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4920         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4921     }
4922 }
4923 { \__zrefclever_ref_default: }
4924 }
4925 }
4926 }

```

(End of 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.

```

4927 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4928 {
4929   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4930   {
4931     \tl_clear:N \l__zrefclever_type_name_tl
4932     \bool_set_true:N \l__zrefclever_type_name_missing_bool
4933   }
4934   {
4935     \tl_if_eq:NnTF
4936       \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4937       {
4938         \tl_clear:N \l__zrefclever_type_name_tl
4939         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4940       }
4941     {
4942       % Determine whether we should use capitalization, abbreviation,
4943       % and plural.
4944       \bool_lazy_or:nnTF
4945         { \l__zrefclever_cap_bool }
4946         {
4947           \l__zrefclever_capfirst_bool &&
4948           \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4949         }
4950         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4951         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4952       % If the queue is empty, we have a singular, otherwise, plural.
4953       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4954         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4955         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4956       \bool_lazy_and:nnTF

```

```

4957 { \l__zrefclever_abbrev_bool }
4958 {
4959 ! \int_compare_p:nNn
4960 { \l__zrefclever_type_count_int } = { 0 } ||
4961 ! \l__zrefclever_noabbrev_first_bool
4962 }
4963 {
4964 \tl_set:NV \l__zrefclever_name_format_fallback_tl
4965 \l__zrefclever_name_format_tl
4966 \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4967 }
4968 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4969
4970 % Handle number and gender nudges.
4971 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4972 {
4973 \bool_if:NTF \l__zrefclever_nudge_singular_bool
4974 {
4975 \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4976 {
4977 \msg_warning:nne { zref-clever }
4978 { nudge-plural-when-sg }
4979 { \l__zrefclever_type_first_label_type_tl }
4980 }
4981 }
4982 {
4983 \bool_lazy_all:nT
4984 {
4985 { \l__zrefclever_nudge_comptosing_bool }
4986 { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4987 {
4988 \int_compare_p:nNn
4989 { \l__zrefclever_label_count_int } > { 0 }
4990 }
4991 }
4992 {
4993 \msg_warning:nne { zref-clever }
4994 { nudge-comptosing }
4995 { \l__zrefclever_type_first_label_type_tl }
4996 }
4997 }
4998 \bool_lazy_and:nnT
4999 { \l__zrefclever_nudge_gender_bool }
5000 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
5001 {
5002 \__zrefclever_get_rf_opt_seq:neeN { gender }
5003 { \l__zrefclever_type_first_label_type_tl }
5004 { \l__zrefclever_ref_language_tl }
5005 \l__zrefclever_type_name_gender_seq
5006 \seq_if_in:NVF
5007 \l__zrefclever_type_name_gender_seq
5008 \l__zrefclever_ref_gender_tl
5009 {
5010 \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq

```

```

5011         {
5012             \msg_warning:nneee { zref-clever }
5013             { nudge-gender-not-declared-for-type }
5014             { \l__zrefclever_ref_gender_tl }
5015             { \l__zrefclever_type_first_label_type_tl }
5016             { \l__zrefclever_ref_language_tl }
5017         }
5018     {
5019         \msg_warning:nneeee { zref-clever }
5020         { nudge-gender-mismatch }
5021         { \l__zrefclever_type_first_label_type_tl }
5022         { \l__zrefclever_ref_gender_tl }
5023         {
5024             \seq_use:Nn
5025             \l__zrefclever_type_name_gender_seq { ,~ }
5026         }
5027         { \l__zrefclever_ref_language_tl }
5028     }
5029 }
5030 }
5031 }
5032
5033 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5034 {
5035     \__zrefclever_opt_tl_get:cNF
5036     {
5037         \__zrefclever_opt_varname_type:een
5038         { \l__zrefclever_type_first_label_type_tl }
5039         { \l__zrefclever_name_format_tl }
5040         { tl }
5041     }
5042     \l__zrefclever_type_name_tl
5043     {
5044         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5045         {
5046             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5047             \tl_put_left:NV \l__zrefclever_name_format_tl
5048             \l__zrefclever_ref_decl_case_tl
5049         }
5050         \__zrefclever_opt_tl_get:cNF
5051         {
5052             \__zrefclever_opt_varname_lang_type:eeen
5053             { \l__zrefclever_ref_language_tl }
5054             { \l__zrefclever_type_first_label_type_tl }
5055             { \l__zrefclever_name_format_tl }
5056             { tl }
5057         }
5058         \l__zrefclever_type_name_tl
5059         {
5060             \tl_clear:N \l__zrefclever_type_name_tl
5061             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5062             \msg_warning:nnee { zref-clever } { missing-name }
5063             { \l__zrefclever_name_format_tl }
5064             { \l__zrefclever_type_first_label_type_tl }

```



```

5065     }
5066   }
5067 }
5068 {
5069   \__zrefclever_opt_tl_get:cNF
5070   {
5071     \__zrefclever_opt_varname_type:een
5072     { \l__zrefclever_type_first_label_type_tl }
5073     { \l__zrefclever_name_format_tl }
5074     { tl }
5075   }
5076   \l__zrefclever_type_name_tl
5077   {
5078     \__zrefclever_opt_tl_get:cNF
5079     {
5080       \__zrefclever_opt_varname_type:een
5081       { \l__zrefclever_type_first_label_type_tl }
5082       { \l__zrefclever_name_format_fallback_tl }
5083       { tl }
5084     }
5085     \l__zrefclever_type_name_tl
5086     {
5087       \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5088       {
5089         \tl_put_left:Nn
5090           \l__zrefclever_name_format_tl { - }
5091         \tl_put_left:NV \l__zrefclever_name_format_tl
5092           \l__zrefclever_ref_decl_case_tl
5093         \tl_put_left:Nn
5094           \l__zrefclever_name_format_fallback_tl { - }
5095         \tl_put_left:NV
5096           \l__zrefclever_name_format_fallback_tl
5097           \l__zrefclever_ref_decl_case_tl
5098       }
5099       \__zrefclever_opt_tl_get:cNF
5100       {
5101         \__zrefclever_opt_varname_lang_type:eeen
5102         { \l__zrefclever_ref_language_tl }
5103         { \l__zrefclever_type_first_label_type_tl }
5104         { \l__zrefclever_name_format_tl }
5105         { tl }
5106       }
5107       \l__zrefclever_type_name_tl
5108       {
5109         \__zrefclever_opt_tl_get:cNF
5110         {
5111           \__zrefclever_opt_varname_lang_type:eeen
5112           { \l__zrefclever_ref_language_tl }
5113           { \l__zrefclever_type_first_label_type_tl }
5114           { \l__zrefclever_name_format_fallback_tl }
5115           { tl }
5116         }
5117         \l__zrefclever_type_name_tl
5118         {

```

```

5119         \tl_clear:N \l__zrefclever_type_name_tl
5120         \bool_set_true:N
5121             \l__zrefclever_type_name_missing_bool
5122         \msg_warning:nnee { zref-clever }
5123             { missing-name }
5124             { \l__zrefclever_name_format_tl }
5125             { \l__zrefclever_type_first_label_type_tl }
5126         }
5127     }
5128 }
5129 }
5130 }
5131 }
5132 }
5133
5134 % Signal whether the type name is to be included in the hyperlink or not.
5135 \bool_lazy_any:nTF
5136 {
5137     { ! \l__zrefclever_hyperlink_bool }
5138     { \l__zrefclever_link_star_bool }
5139     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5140     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } } }
5141 }
5142 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5143 {
5144     \bool_lazy_any:nTF
5145     {
5146         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } } }
5147         {
5148             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5149             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5150         }
5151         {
5152             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5153             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5154             \l__zrefclever_typeset_last_bool &&
5155             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5156         }
5157     }
5158     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5159     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5160 }
5161 }

```

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

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

```

\__zrefclever_hyperlink:nnn {<url/file>} {<anchor>} {<text>}
5162 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5163 {
5164     \tl_if_empty:nTF {#1}

```

```

5165     { \hyperlink {#2} {#3} }
5166     { \hyper@linkfile {#3} {#1} {#2} }
5167   }

```

(End of definition for `\__zrefclever_hyperlink:nnn`.)

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

```

5168 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5169 {
5170   \zref@ifpropundefined { urluse }
5171   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5172   {
5173     \zref@ifrefcontainsprop {#1} { urluse }
5174     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5175     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5176   }
5177 }
5178 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End of definition for `\__zrefclever_extract_url_unexp:n`.)

`\__zrefclever_labels_in_sequence:mn` Auxiliary function to `\__zrefclever_typeset_refs_not_last_of_type:`. Sets `\l__zrefclever_next_maybe_range_bool` to true if `<label b>` comes in immediate sequence from `<label a>`. 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:mn {<label a>} {<label b>}

```

```

5179 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:mn #1#2
5180 {
5181   \exp_args:Nee \tl_if_eq:nnT
5182   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5183   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5184   {
5185     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5186     {
5187       \exp_args:Nee \tl_if_eq:nnT
5188       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5189       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5190       {
5191         \int_compare:nNnTF
5192         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5193         =
5194         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5195         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5196         {
5197           \int_compare:nNnT
5198           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }

```

```

5199         =
5200         { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5201         {
5202             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5203             \bool_set_true:N \l__zrefclever_next_is_same_bool
5204         }
5205     }
5206 }
5207 }
5208 {
5209 \exp_args:Nee \tl_if_eq:nnT
5210 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5211 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5212 {
5213 \exp_args:Nee \tl_if_eq:nnT
5214 { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5215 { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5216 {
5217 \int_compare:nNnTF
5218 { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5219 =
5220 { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5221 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5222 {
5223 \int_compare:nNnT
5224 { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5225 =
5226 { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5227 {

```

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.

```

5228 \exp_args:Nee \tl_if_eq:nnT
5229 {
5230     \_zrefclever_extract_unexp:nvn {#1}
5231     { l__zrefclever_ref_property_tl } { }
5232 }
5233 {
5234     \_zrefclever_extract_unexp:nvn {#2}
5235     { l__zrefclever_ref_property_tl } { }
5236 }
5237 {
5238     \bool_set_true:N
5239     \l__zrefclever_next_maybe_range_bool
5240     \bool_set_true:N
5241     \l__zrefclever_next_is_same_bool
5242 }
5243 }
5244 }
5245 }
5246 }

```

```

5247     }
5248   }
5249 }

```

(End of 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>}
5250 \cs_new_protected:Npn \_zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5251 {
5252   % First attempt: general options.
5253   \_zrefclever_opt_tl_get:cNF
5254   { \_zrefclever_opt_varname_general:nn {#1} { tl } }
5255   #4
5256   {
5257     % If not found, try type specific options.
5258     \_zrefclever_opt_tl_get:cNF
5259     { \_zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5260     #4
5261     {
5262       % If not found, try type- and language-specific.
5263       \_zrefclever_opt_tl_get:cNF
5264       { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5265       #4
5266       {
5267         % If not found, try language-specific default.
5268         \_zrefclever_opt_tl_get:cNF
5269         { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5270         #4
5271         {
5272           % If not found, try fallback.
5273           \_zrefclever_opt_tl_get:cNF
5274           { \_zrefclever_opt_varname_fallback:nn {#1} { tl } }
5275           #4
5276           { \tl_clear:N #4 }
5277         }
5278       }
5279     }
5280   }
5281 }
5282 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_tl:nnnN { neeN }

```

(End of 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>}
5283 \cs_new_protected:Npn \_zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5284 {
5285   % First attempt: general options.
5286   \_zrefclever_opt_seq_get:cNF

```

```

5287 { \_zrefclever_opt_varname_general:nn {#1} { seq } }
5288 #4
5289 {
5290 % If not found, try type specific options.
5291 \_zrefclever_opt_seq_get:cNF
5292 { \_zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5293 #4
5294 {
5295 % If not found, try type- and language-specific.
5296 \_zrefclever_opt_seq_get:cNF
5297 { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5298 #4
5299 {
5300 % If not found, try language-specific default.
5301 \_zrefclever_opt_seq_get:cNF
5302 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5303 #4
5304 {
5305 % If not found, try fallback.
5306 \_zrefclever_opt_seq_get:cNF
5307 { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5308 #4
5309 { \seq_clear:N #4 }
5310 }
5311 }
5312 }
5313 }
5314 }
5315 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_seq:nnnN { neeN }

```

(End of 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>}
5316 \cs_new_protected:Npn \_zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5317 {
5318 % First attempt: general options.
5319 \_zrefclever_opt_bool_get:cNF
5320 { \_zrefclever_opt_varname_general:nn {#1} { bool } }
5321 #5
5322 {
5323 % If not found, try type specific options.
5324 \_zrefclever_opt_bool_get:cNF
5325 { \_zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5326 #5
5327 {
5328 % If not found, try type- and language-specific.
5329 \_zrefclever_opt_bool_get:cNF
5330 { \_zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5331 #5
5332 {
5333 % If not found, try language-specific default.
5334 \_zrefclever_opt_bool_get:cNF
5335 { \_zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }

```

```

5336             #5
5337             {
5338             % If not found, try fallback.
5339             \_zrefclever_opt_bool_get:cNF
5340             { \_zrefclever_opt_varname_fallback:nn {#1} { bool } }
5341             #5
5342             { \use:c { bool_set_ #2 :N } #5 }
5343             }
5344         }
5345     }
5346 }
5347 }
5348 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_bool:nnnnN { nneeN }

```

(End of 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.

```

5349 \_zrefclever_compat_module:nn { appendix }
5350 {
5351     \AddToHook { cmd / appendix / before }
5352     {
5353         \_zrefclever_zcsetup:n
5354         {
5355             countertype =
5356             {
5357                 chapter      = appendix ,
5358                 section      = appendix ,
5359                 subsection   = appendix ,
5360                 subsubsection = appendix ,

```

```

5361         paragraph      = appendix ,
5362         subparagraph   = appendix ,
5363     }
5364 }
5365 }
5366 }

```

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.

```

5367 \__zrefclever_compat_module:nn { appendices }
5368 {
5369     \__zrefclever_if_package_loaded:nT { appendix }
5370     {
5371         \newcounter { zc@appendix }
5372         \newcounter { zc@save@appendix }
5373         \setcounter { zc@appendix } { 0 }
5374         \setcounter { zc@save@appendix } { 0 }
5375         \cs_if_exist:cTF { chapter }
5376         {
5377             \__zrefclever_zcsetup:n
5378             { counterresetby = { chapter = zc@appendix } }
5379         }
5380         {
5381             \cs_if_exist:cT { section }
5382             {
5383                 \__zrefclever_zcsetup:n
5384                 { counterresetby = { section = zc@appendix } }
5385             }
5386         }
5387     }
5388     \AddToHook { env / appendices / begin }
5389     {
5390         \stepcounter { zc@save@appendix }
5391         \setcounter { zc@appendix } { \value { zc@save@appendix } }
5392         \__zrefclever_zcsetup:n
5393         {
5394             countertype =
5395             {
5396                 chapter      = appendix ,
5397                 section      = appendix ,
5398                 subsection   = appendix ,
5399                 subsubsection = appendix ,
5400                 paragraph    = appendix ,

```



```

5400         subparagraph = appendix ,
5401     }
5402 }
5403 }
5404 \AddToHook { env / appendices / end }
5405 { \setcounter { zc@appendix } { 0 } }
5406 \AddToHook { cmd / appendix / before }
5407 {
5408     \stepcounter { zc@save@appendix }
5409     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5410 }
5411 \AddToHook { env / subappendices / begin }
5412 {
5413     \__zrefclever_zcsetup:n
5414     {
5415         countertype =
5416         {
5417             section      = appendix ,
5418             subsection   = appendix ,
5419             subsubsection = appendix ,
5420             paragraph    = appendix ,
5421             subparagraph = appendix ,
5422         } ,
5423     }
5424 }
5425 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5426 }
5427 }

```

### 9.3 memoir

The `memoir` document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. It used to be the case that a good number of them were implemented in ways which made difficult the use of `zref`, particularly `\zlabel`. Problematic cases included: i) side captions; ii) bilingual captions; iii) subcaption references; and iv) footnotes, verfootnotes, sidefootnotes, and pagenotes.

However, since then, the situation has much improved, given two main upstream changes: i) the kernel's new `label` hook with argument, introduced in the release of 2023-06-01 (thanks to Ulrike Fischer and Phelype Oleinik) and ii) better support for `zref` and `zref-clever` from the `memoir` class itself, with release of 2023/08/08 v3.8 (thanks to Lars Madsen).

Also, 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.

```

5428 \__zrefclever_compat_module:nn { memoir }
5429 {
5430     \__zrefclever_if_class_loaded:nT { memoir }
5431     {

```

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`.

```

5432     \__zrefclever_zcsetup:n
5433     {
5434         countertype =
5435         {
5436             subfigure = figure ,
5437             subtable  = table  ,
5438             poemline  = line   ,
5439         } ,
5440         counterresetby =
5441         {
5442             subfigure = figure ,
5443             subtable  = table  ,
5444         } ,
5445     }

```

Support for subcaption references.

```

5446     \zref@newprop { subcaption }
5447     { \cs_if_exist_use:c { @@thesub \@capttype } }
5448     \AddToHook{ memoir/subcaption/aftercounter }
5449     { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

Support for `\sidefootnote` and `\pagenote`.

```

5450     \__zrefclever_zcsetup:n
5451     {
5452         countertype =
5453         {
5454             sidefootnote = footnote ,
5455             pagenote     = endnote  ,
5456         } ,
5457     }
5458     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5459 }
5460 }

```

## 9.4 amsmath

About this, see <https://tex.stackexchange.com/a/402297> and <https://github.com/ho-tex/zref/issues/4>.

```

5461 \__zrefclever_compat_module:nn { amsmath }
5462 {
5463     \__zrefclever_if_package_loaded:nT { amsmath }
5464     {

```

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).

```

5465     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5466     \AddToHook { env / subequations / begin }
5467     {
5468         \__zrefclever_zcsetup:e
5469         {
5470             counterresetby =
5471             {
5472                 parentequation =
5473                 \__zrefclever_counter_reset_by:n { equation } ,
5474                 equation = parentequation ,
5475             } ,
5476             currentcounter = parentequation ,
5477             countertype = { parentequation = equation } ,
5478         }
5479     \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5480     }

```

`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.

```

5481     \zref@newprop { subeq } { \alph { equation } }
5482     \clist_map_inline:nn
5483     {
5484         equation ,
5485         equation* ,
5486         align ,
5487         align* ,
5488         alignat ,
5489         alignat* ,
5490         flalign ,
5491         flalign* ,
5492         xalignat ,
5493         xalignat* ,
5494         gather ,
5495         gather* ,
5496         multiline ,
5497         multiline* ,
5498     }
5499     {
5500     \AddToHook { env / #1 / begin }
5501     {
5502         \__zrefclever_zcsetup:n { currentcounter = equation }

```

```

5503         \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5504             { \zref@localaddprop \ZREF@mainlist { subeq } }
5505     }
5506 }
5507 \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5508 }
5509 }

```

## 9.5 mathtools

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

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

```

5510 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5511 \__zrefclever_compat_module:nn { mathtools }
5512 {
5513   \__zrefclever_if_package_loaded:nT { mathtools }
5514   {
5515     \MH_if_boolean:nT { show_only_refs }
5516     {
5517       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5518       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5519         {
5520           \@bsphack
5521           \seq_map_inline:Nn #1
5522             {
5523               \exp_args:Ne \tl_if_eq:nnTF
5524                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5525                 { equation }
5526                 {
5527                   \protected@write \@auxout { }
5528                     { \string \MT@newlabel {##1} }
5529                 }
5530             }
5531           \exp_args:Ne \tl_if_eq:nnT
5532             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5533             { parentequation }
5534             {
5535               \protected@write \@auxout { }
5536                 { \string \MT@newlabel {##1} }
5537             }
5538         }
5539     }
5540   \@esphack
5541 }
5542 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5543 }

```

```

5544     }
5545 }

```

## 9.6 breqn

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for `\zlabel` just as well.

```

5546 \__zrefclever_compat_module:nn { breqn }
5547 {
5548   \__zrefclever_if_package_loaded:nT { breqn }
5549   {

```

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>).

```

5550     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5551     \AddToHook { env / dgroup / begin }
5552     {
5553       \__zrefclever_zcsetup:e
5554       {
5555         counterresetby =
5556         {
5557           parentequation =
5558             \__zrefclever_counter_reset_by:n { equation } ,
5559           equation = parentequation ,
5560         } ,
5561         currentcounter = parentequation ,
5562         countertype = { parentequation = equation } ,
5563       }
5564     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5565     }
5566     \zref@ifpropundefined { subeq }
5567     { \zref@newprop { subeq } { \alph { equation } } }
5568     { }
5569     \clist_map_inline:nn
5570     {
5571       dmath ,
5572       dseries ,
5573       darray ,
5574     }
5575     {
5576       \AddToHook { env / #1 / begin }
5577       {
5578         \__zrefclever_zcsetup:n { currentcounter = equation }
5579         \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5580         { \zref@localaddprop \ZREF@mainlist { subeq } }
5581       }
5582     }
5583     \msg_info:nnn { zref-clever } { compat-package } { breqn }

```

```

5584     }
5585 }

```

## 9.7 listings

```

5586 \__zrefclever_compat_module:nn { listings }
5587 {
5588   \__zrefclever_if_package_loaded:nT { listings }
5589   {
5590     \__zrefclever_zcsetup:n
5591     {
5592       countertype =
5593       {
5594         lstlisting = listing ,
5595         lstnumber = line ,
5596       } ,
5597       counterresetby = { lstnumber = lstlisting } ,
5598     }

```

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.

```

5599   \lst@AddToHook { Init }
5600     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5601   \msg_info:nnn { zref-clever } { compat-package } { listings }
5602 }
5603 }

```

## 9.8 enumitem

The procedure below will “see” any changes made to the `enumerate` environment (made with `enumitem`’s `\renewlist`) as long as it is done in the preamble. Though, technically, `\renewlist` can be issued anywhere in the document, this should be more than enough for the purpose at hand. Besides, trying to retrieve this information “on the fly” would be much overkill.

The only real reason to “renew” `enumerate` itself is to change `{\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.

```

5604 \__zrefclever_compat_module:nn { enumitem }
5605 {
5606   \__zrefclever_if_package_loaded:nT { enumitem }
5607   {
5608     \int_set:Nn \l__zrefclever_tmpa_int { 5 }

```

```

5609     \bool_while_do:nn
5610     {
5611         \cs_if_exist_p:c
5612         { c@ enum \int_to_roman:n { \l__zrefclever_tmpa_int } }
5613     }
5614     {
5615         \__zrefclever_zcsetup:e
5616         {
5617             counterresetby =
5618             {
5619                 enum \int_to_roman:n { \l__zrefclever_tmpa_int } =
5620                 enum \int_to_roman:n { \l__zrefclever_tmpa_int - 1 }
5621             } ,
5622             countertype =
5623             { enum \int_to_roman:n { \l__zrefclever_tmpa_int } = item } ,
5624         }
5625         \int_incr:N \l__zrefclever_tmpa_int
5626     }
5627     \int_compare:nNnT { \l__zrefclever_tmpa_int } > { 5 }
5628     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5629 }
5630 }

```

## 9.9 subcaption

```

5631 \__zrefclever_compat_module:nn { subcaption }
5632 {
5633     \__zrefclever_if_package_loaded:nT { subcaption }
5634     {
5635         \__zrefclever_zcsetup:n
5636         {
5637             countertype =
5638             {
5639                 subfigure = figure ,
5640                 subtable = table ,
5641             } ,
5642             counterresetby =
5643             {
5644                 subfigure = figure ,
5645                 subtable = table ,
5646             } ,
5647         }

```

Support for subref reference.

```

5648     \zref@newprop { subref }
5649     { \cs_if_exist_use:c { thesub \@capttype } }
5650     \tl_put_right:Nn \caption@subtypehook
5651     { \zref@localaddprop \ZREF@mainlist { subref } }
5652 }
5653 }

```

## 9.10 subfig

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

```

5654 \_zrefclever_compat_module:nn { subfig }
5655 {
5656   \_zrefclever_if_package_loaded:nT { subfig }
5657   {
5658     \_zrefclever_zcsetup:n
5659     {
5660       countertype =
5661       {
5662         subfigure = figure ,
5663         subtable = table ,
5664       } ,
5665       counterresetby =
5666       {
5667         subfigure = figure ,
5668         subtable = table ,
5669       } ,
5670     }
5671   }
5672 }
5673 </package>

```

## 10 Language files

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

### 10.1 Localization guidelines

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

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

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

**Sectioning:** A number of reference types are provided to support referencing to document sectioning commands. Obviously, `part`, `chapter`, `section`, and `paragraph` are meant to refer to the sectioning commands of the standard classes and elsewhere, which anyone reading this is certainly acquainted with. Note that `zref-clever` uses – by default at least, which is what the language files cater for – the `section` reference type to refer to `\subsections` and `\subsubsections` as well, similarly, `paragraph` also refers to `\subparagraph`. The `appendix` reference type is meant to refer to any sectioning



command – be them chapters, sections, or paragraphs – issued after `\appendix`, which corresponds to how the standard classes, the KOMA Script classes, and `memoir` deal with appendices. The `book` reference type deserves some explanation. The word “book” has a good number of meanings, and the most common one is not the one which is intended here. The Webster dictionary gives us a couple of definitions of interest: “1. A collection of sheets of paper, or similar material, blank, written, or printed, bound together; commonly, many folded and bound sheets containing continuous printing or writing.” and “3. A part or subdivision of a treatise or literary work; as, the tenth book of ‘Paradise Lost.’” It is this third meaning which the `book` reference type is meant to support: a major subdivision of a work, much like `\part`. Even if it does not exist in the standard classes, it may exist elsewhere, in particular, it is provided by `memoir`.

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

**Notes:** `zref-clever` provides three reference types in this area: `footnote`, `endnote`, and `note`. The first two refer to footnotes and end notes, respectively. The third is meant as a convenience for a general “note” object, either the other two, or something else. By experience, here is one place where that initial observation of not simply translating the reference types names is particularly relevant. There’s a natural temptation, because three different types exist and are somewhat close to each other, to distinguish them clearly. Duty would compel us to do so. But that may lead to less than ideal results. Different terms work well for some languages, like English and German, which have compound words for the purpose. But less so for other languages, like Portuguese, French, or Italian. For example, in a document in French which only contains footnotes, arguably a very common use case, would it be better to refer to a footnote as just “note”, or be very precise with “note infrapaginale”? Of course, in a document which contains both footnotes and end notes, we may need the distinction. But is it really the better default? True, possibly the inclusion of the `note` reference type, with no clear object to refer to, creates more noise than convenience here. If I recall correctly, my intention was to provide an easy way out for users from possible contentious localizations for `footnote` and `endnote`, but I’m not sure if it’s been working like this in practice, and I should probably have refrained from adding it in the first place.

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

**Code:** A couple of reference types are provided for code related environments: `algorithm` and `listing`. By experience, the `listing` type has already proven to be a particularly challenging one. Formally, it should be a good default term to encompass anything which may regularly be included in a `lstlisting` environment as provided by

the `listings` package. However, it seems that in different languages it is quite difficult to find a satisfying term for it. Though my English is decent, I'm not a native speaker, still I'm not even sure how common the term is used for the purpose even in English. It seems to be traditional enough in the L<sup>A</sup>T<sub>E</sub>X community at least. In doubt, pend to the jargon side, anglicism if need be. Since we are bound to displease mostly everyone anyway, at least we do so in a consistent manner.

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

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

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

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

were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

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

## 10.2 English

English language file has been initially provided by the author.

```
5674 ⟨*package⟩
5675 \zcDeclareLanguage { english }
5676 \zcDeclareLanguageAlias { american } { english }
5677 \zcDeclareLanguageAlias { australian } { english }
5678 \zcDeclareLanguageAlias { british } { english }
5679 \zcDeclareLanguageAlias { canadian } { english }
5680 \zcDeclareLanguageAlias { newzealand } { english }
5681 \zcDeclareLanguageAlias { UKenglish } { english }
5682 \zcDeclareLanguageAlias { USenglish } { english }
5683 ⟨/package⟩

5684 ⟨*lang-english⟩

5685 namesep = {\nobreakspace} ,
5686 pairsep = {~and\nobreakspace} ,
5687 listsep = {,~} ,
5688 lastsep = {~and\nobreakspace} ,
5689 tpairsep = {~and\nobreakspace} ,
5690 tlistsep = {,~} ,
5691 tlastsep = {,~and\nobreakspace} ,
5692 notesep = {~} ,
5693 rangesep = {~to\nobreakspace} ,
5694
5695 type = book ,
5696 Name-sg = Book ,
5697 name-sg = book ,
5698 Name-pl = Books ,
5699 name-pl = books ,
5700
5701 type = part ,
5702 Name-sg = Part ,
5703 name-sg = part ,
5704 Name-pl = Parts ,
5705 name-pl = parts ,
5706
5707 type = chapter ,
5708 Name-sg = Chapter ,
5709 name-sg = chapter ,
5710 Name-pl = Chapters ,
5711 name-pl = chapters ,
5712
5713 type = section ,
5714 Name-sg = Section ,
5715 name-sg = section ,
```

```

5716 Name-pl = Sections ,
5717 name-pl = sections ,
5718
5719 type = paragraph ,
5720 Name-sg = Paragraph ,
5721 name-sg = paragraph ,
5722 Name-pl = Paragraphs ,
5723 name-pl = paragraphs ,
5724 Name-sg-ab = Par. ,
5725 name-sg-ab = par. ,
5726 Name-pl-ab = Par. ,
5727 name-pl-ab = par. ,
5728
5729 type = appendix ,
5730 Name-sg = Appendix ,
5731 name-sg = appendix ,
5732 Name-pl = Appendices ,
5733 name-pl = appendices ,
5734
5735 type = page ,
5736 Name-sg = Page ,
5737 name-sg = page ,
5738 Name-pl = Pages ,
5739 name-pl = pages ,
5740 rangesep = {\textendash} ,
5741 rangetopair = false ,
5742
5743 type = line ,
5744 Name-sg = Line ,
5745 name-sg = line ,
5746 Name-pl = Lines ,
5747 name-pl = lines ,
5748
5749 type = figure ,
5750 Name-sg = Figure ,
5751 name-sg = figure ,
5752 Name-pl = Figures ,
5753 name-pl = figures ,
5754 Name-sg-ab = Fig. ,
5755 name-sg-ab = fig. ,
5756 Name-pl-ab = Figs. ,
5757 name-pl-ab = figs. ,
5758
5759 type = table ,
5760 Name-sg = Table ,
5761 name-sg = table ,
5762 Name-pl = Tables ,
5763 name-pl = tables ,
5764
5765 type = item ,
5766 Name-sg = Item ,
5767 name-sg = item ,
5768 Name-pl = Items ,
5769 name-pl = items ,

```

```

5770
5771 type = footnote ,
5772   Name-sg = Footnote ,
5773   name-sg = footnote ,
5774   Name-pl = Footnotes ,
5775   name-pl = footnotes ,
5776
5777 type = endnote ,
5778   Name-sg = Note ,
5779   name-sg = note ,
5780   Name-pl = Notes ,
5781   name-pl = notes ,
5782
5783 type = note ,
5784   Name-sg = Note ,
5785   name-sg = note ,
5786   Name-pl = Notes ,
5787   name-pl = notes ,
5788
5789 type = equation ,
5790   Name-sg = Equation ,
5791   name-sg = equation ,
5792   Name-pl = Equations ,
5793   name-pl = equations ,
5794   Name-sg-ab = Eq. ,
5795   name-sg-ab = eq. ,
5796   Name-pl-ab = Eqs. ,
5797   name-pl-ab = eqs. ,
5798   refbounds-first-sg = {,(,)}, ,
5799   refbounds = {(,,)} ,
5800
5801 type = theorem ,
5802   Name-sg = Theorem ,
5803   name-sg = theorem ,
5804   Name-pl = Theorems ,
5805   name-pl = theorems ,
5806
5807 type = lemma ,
5808   Name-sg = Lemma ,
5809   name-sg = lemma ,
5810   Name-pl = Lemmas ,
5811   name-pl = lemmas ,
5812
5813 type = corollary ,
5814   Name-sg = Corollary ,
5815   name-sg = corollary ,
5816   Name-pl = Corollaries ,
5817   name-pl = corollaries ,
5818
5819 type = proposition ,
5820   Name-sg = Proposition ,
5821   name-sg = proposition ,
5822   Name-pl = Propositions ,
5823   name-pl = propositions ,

```

```

5824
5825 type = definition ,
5826     Name-sg = Definition ,
5827     name-sg = definition ,
5828     Name-pl = Definitions ,
5829     name-pl = definitions ,
5830
5831 type = proof ,
5832     Name-sg = Proof ,
5833     name-sg = proof ,
5834     Name-pl = Proofs ,
5835     name-pl = proofs ,
5836
5837 type = result ,
5838     Name-sg = Result ,
5839     name-sg = result ,
5840     Name-pl = Results ,
5841     name-pl = results ,
5842
5843 type = remark ,
5844     Name-sg = Remark ,
5845     name-sg = remark ,
5846     Name-pl = Remarks ,
5847     name-pl = remarks ,
5848
5849 type = example ,
5850     Name-sg = Example ,
5851     name-sg = example ,
5852     Name-pl = Examples ,
5853     name-pl = examples ,
5854
5855 type = algorithm ,
5856     Name-sg = Algorithm ,
5857     name-sg = algorithm ,
5858     Name-pl = Algorithms ,
5859     name-pl = algorithms ,
5860
5861 type = listing ,
5862     Name-sg = Listing ,
5863     name-sg = listing ,
5864     Name-pl = Listings ,
5865     name-pl = listings ,
5866
5867 type = exercise ,
5868     Name-sg = Exercise ,
5869     name-sg = exercise ,
5870     Name-pl = Exercises ,
5871     name-pl = exercises ,
5872
5873 type = solution ,
5874     Name-sg = Solution ,
5875     name-sg = solution ,
5876     Name-pl = Solutions ,
5877     name-pl = solutions ,

```

5878  $\langle$ /lang-english)

### 10.3 German

German language file has been initially provided by the author.

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

```
5879  $\langle$ *package)
5880 \zcDeclareLanguage
5881 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5882 { german }
5883 \zcDeclareLanguageAlias { ngerman      } { german }
5884 \zcDeclareLanguageAlias { austrian     } { german }
5885 \zcDeclareLanguageAlias { naustrian    } { german }
5886 \zcDeclareLanguageAlias { swissgerman  } { german }
5887 \zcDeclareLanguageAlias { nswissgerman } { german }
5888  $\langle$ /package)

5889  $\langle$ *lang-german)

5890 namesep = {\nobreakspace} ,
5891 pairsep  = {\~und\nobreakspace} ,
5892 listsep  = { , ~ } ,
5893 lastsep  = {\~und\nobreakspace} ,
5894 tpairsep = {\~und\nobreakspace} ,
5895 tlistsep = { , ~ } ,
5896 tlastsep = {\~und\nobreakspace} ,
5897 notesep  = { ~ } ,
5898 rangesep = {\~bis\nobreakspace} ,
5899
5900 type = book ,
5901 gender = n ,
5902 case = N ,
5903   Name-sg = Buch ,
5904   Name-pl = Bücher ,
5905 case = A ,
5906   Name-sg = Buch ,
5907   Name-pl = Bücher ,
5908 case = D ,
5909   Name-sg = Buch ,
5910   Name-pl = Büchern ,
5911 case = G ,
5912   Name-sg = Buches ,
5913   Name-pl = Bücher ,
5914
5915 type = part ,
5916 gender = m ,
5917 case = N ,
5918   Name-sg = Teil ,
5919   Name-pl = Teile ,
5920 case = A ,
5921   Name-sg = Teil ,
5922   Name-pl = Teile ,
5923 case = D ,
```

```

5924     Name-sg = Teil ,
5925     Name-pl = Teilen ,
5926     case = G ,
5927     Name-sg = Teiles ,
5928     Name-pl = Teile ,
5929
5930 type = chapter ,
5931     gender = n ,
5932     case = N ,
5933     Name-sg = Kapitel ,
5934     Name-pl = Kapitel ,
5935     case = A ,
5936     Name-sg = Kapitel ,
5937     Name-pl = Kapitel ,
5938     case = D ,
5939     Name-sg = Kapitel ,
5940     Name-pl = Kapiteln ,
5941     case = G ,
5942     Name-sg = Kapitels ,
5943     Name-pl = Kapitel ,
5944
5945 type = section ,
5946     gender = m ,
5947     case = N ,
5948     Name-sg = Abschnitt ,
5949     Name-pl = Abschnitte ,
5950     case = A ,
5951     Name-sg = Abschnitt ,
5952     Name-pl = Abschnitte ,
5953     case = D ,
5954     Name-sg = Abschnitt ,
5955     Name-pl = Abschnitten ,
5956     case = G ,
5957     Name-sg = Abschnitts ,
5958     Name-pl = Abschnitte ,
5959
5960 type = paragraph ,
5961     gender = m ,
5962     case = N ,
5963     Name-sg = Absatz ,
5964     Name-pl = Absätze ,
5965     case = A ,
5966     Name-sg = Absatz ,
5967     Name-pl = Absätze ,
5968     case = D ,
5969     Name-sg = Absatz ,
5970     Name-pl = Absätzen ,
5971     case = G ,
5972     Name-sg = Absatzes ,
5973     Name-pl = Absätze ,
5974
5975 type = appendix ,
5976     gender = m ,
5977     case = N ,

```



```

5978     Name-sg = Anhang ,
5979     Name-pl = Anhänge ,
5980     case = A ,
5981     Name-sg = Anhang ,
5982     Name-pl = Anhänge ,
5983     case = D ,
5984     Name-sg = Anhang ,
5985     Name-pl = Anhängen ,
5986     case = G ,
5987     Name-sg = Anhangs ,
5988     Name-pl = Anhänge ,
5989
5990     type = page ,
5991     gender = f ,
5992     case = N ,
5993     Name-sg = Seite ,
5994     Name-pl = Seiten ,
5995     case = A ,
5996     Name-sg = Seite ,
5997     Name-pl = Seiten ,
5998     case = D ,
5999     Name-sg = Seite ,
6000     Name-pl = Seiten ,
6001     case = G ,
6002     Name-sg = Seite ,
6003     Name-pl = Seiten ,
6004     rangeseq = {\textendash} ,
6005     rangetopair = false ,
6006
6007     type = line ,
6008     gender = f ,
6009     case = N ,
6010     Name-sg = Zeile ,
6011     Name-pl = Zeilen ,
6012     case = A ,
6013     Name-sg = Zeile ,
6014     Name-pl = Zeilen ,
6015     case = D ,
6016     Name-sg = Zeile ,
6017     Name-pl = Zeilen ,
6018     case = G ,
6019     Name-sg = Zeile ,
6020     Name-pl = Zeilen ,
6021
6022     type = figure ,
6023     gender = f ,
6024     case = N ,
6025     Name-sg = Abbildung ,
6026     Name-pl = Abbildungen ,
6027     Name-sg-ab = Abb. ,
6028     Name-pl-ab = Abb. ,
6029     case = A ,
6030     Name-sg = Abbildung ,
6031     Name-pl = Abbildungen ,

```

```

6032     Name-sg-ab = Abb. ,
6033     Name-pl-ab = Abb. ,
6034     case = D ,
6035     Name-sg = Abbildung ,
6036     Name-pl = Abbildungen ,
6037     Name-sg-ab = Abb. ,
6038     Name-pl-ab = Abb. ,
6039     case = G ,
6040     Name-sg = Abbildung ,
6041     Name-pl = Abbildungen ,
6042     Name-sg-ab = Abb. ,
6043     Name-pl-ab = Abb. ,
6044
6045     type = table ,
6046     gender = f ,
6047     case = N ,
6048     Name-sg = Tabelle ,
6049     Name-pl = Tabellen ,
6050     case = A ,
6051     Name-sg = Tabelle ,
6052     Name-pl = Tabellen ,
6053     case = D ,
6054     Name-sg = Tabelle ,
6055     Name-pl = Tabellen ,
6056     case = G ,
6057     Name-sg = Tabelle ,
6058     Name-pl = Tabellen ,
6059
6060     type = item ,
6061     gender = m ,
6062     case = N ,
6063     Name-sg = Punkt ,
6064     Name-pl = Punkte ,
6065     case = A ,
6066     Name-sg = Punkt ,
6067     Name-pl = Punkte ,
6068     case = D ,
6069     Name-sg = Punkt ,
6070     Name-pl = Punkten ,
6071     case = G ,
6072     Name-sg = Punktes ,
6073     Name-pl = Punkte ,
6074
6075     type = footnote ,
6076     gender = f ,
6077     case = N ,
6078     Name-sg = Fußnote ,
6079     Name-pl = Fußnoten ,
6080     case = A ,
6081     Name-sg = Fußnote ,
6082     Name-pl = Fußnoten ,
6083     case = D ,
6084     Name-sg = Fußnote ,
6085     Name-pl = Fußnoten ,

```

```

6086 case = G ,
6087     Name-sg = Fußnote ,
6088     Name-pl = Fußnoten ,
6089
6090 type = endnote ,
6091     gender = f ,
6092     case = N ,
6093         Name-sg = Endnote ,
6094         Name-pl = Endnoten ,
6095     case = A ,
6096         Name-sg = Endnote ,
6097         Name-pl = Endnoten ,
6098     case = D ,
6099         Name-sg = Endnote ,
6100         Name-pl = Endnoten ,
6101     case = G ,
6102         Name-sg = Endnote ,
6103         Name-pl = Endnoten ,
6104
6105 type = note ,
6106     gender = f ,
6107     case = N ,
6108         Name-sg = Anmerkung ,
6109         Name-pl = Anmerkungen ,
6110     case = A ,
6111         Name-sg = Anmerkung ,
6112         Name-pl = Anmerkungen ,
6113     case = D ,
6114         Name-sg = Anmerkung ,
6115         Name-pl = Anmerkungen ,
6116     case = G ,
6117         Name-sg = Anmerkung ,
6118         Name-pl = Anmerkungen ,
6119
6120 type = equation ,
6121     gender = f ,
6122     case = N ,
6123         Name-sg = Gleichung ,
6124         Name-pl = Gleichungen ,
6125     case = A ,
6126         Name-sg = Gleichung ,
6127         Name-pl = Gleichungen ,
6128     case = D ,
6129         Name-sg = Gleichung ,
6130         Name-pl = Gleichungen ,
6131     case = G ,
6132         Name-sg = Gleichung ,
6133         Name-pl = Gleichungen ,
6134     refbounds-first-sg = {,(,)},
6135     refbounds = {(,,)} ,
6136
6137 type = theorem ,
6138     gender = n ,
6139     case = N ,

```

```

6140     Name-sg = Theorem ,
6141     Name-pl = Theoreme ,
6142     case = A ,
6143     Name-sg = Theorem ,
6144     Name-pl = Theoreme ,
6145     case = D ,
6146     Name-sg = Theorem ,
6147     Name-pl = Theoremen ,
6148     case = G ,
6149     Name-sg = Theorems ,
6150     Name-pl = Theoreme ,
6151
6152     type = lemma ,
6153     gender = n ,
6154     case = N ,
6155     Name-sg = Lemma ,
6156     Name-pl = Lemmata ,
6157     case = A ,
6158     Name-sg = Lemma ,
6159     Name-pl = Lemmata ,
6160     case = D ,
6161     Name-sg = Lemma ,
6162     Name-pl = Lemmata ,
6163     case = G ,
6164     Name-sg = Lemmas ,
6165     Name-pl = Lemmata ,
6166
6167     type = corollary ,
6168     gender = n ,
6169     case = N ,
6170     Name-sg = Korollar ,
6171     Name-pl = Korollare ,
6172     case = A ,
6173     Name-sg = Korollar ,
6174     Name-pl = Korollare ,
6175     case = D ,
6176     Name-sg = Korollar ,
6177     Name-pl = Korollaren ,
6178     case = G ,
6179     Name-sg = Korollars ,
6180     Name-pl = Korollare ,
6181
6182     type = proposition ,
6183     gender = m ,
6184     case = N ,
6185     Name-sg = Satz ,
6186     Name-pl = Sätze ,
6187     case = A ,
6188     Name-sg = Satz ,
6189     Name-pl = Sätze ,
6190     case = D ,
6191     Name-sg = Satz ,
6192     Name-pl = Sätzen ,
6193     case = G ,

```

```

6194     Name-sg = Satzes ,
6195     Name-pl = Sätze ,
6196
6197 type = definition ,
6198     gender = f ,
6199     case = N ,
6200     Name-sg = Definition ,
6201     Name-pl = Definitionen ,
6202     case = A ,
6203     Name-sg = Definition ,
6204     Name-pl = Definitionen ,
6205     case = D ,
6206     Name-sg = Definition ,
6207     Name-pl = Definitionen ,
6208     case = G ,
6209     Name-sg = Definition ,
6210     Name-pl = Definitionen ,
6211
6212 type = proof ,
6213     gender = m ,
6214     case = N ,
6215     Name-sg = Beweis ,
6216     Name-pl = Beweise ,
6217     case = A ,
6218     Name-sg = Beweis ,
6219     Name-pl = Beweise ,
6220     case = D ,
6221     Name-sg = Beweis ,
6222     Name-pl = Beweisen ,
6223     case = G ,
6224     Name-sg = Beweises ,
6225     Name-pl = Beweise ,
6226
6227 type = result ,
6228     gender = n ,
6229     case = N ,
6230     Name-sg = Ergebnis ,
6231     Name-pl = Ergebnisse ,
6232     case = A ,
6233     Name-sg = Ergebnis ,
6234     Name-pl = Ergebnisse ,
6235     case = D ,
6236     Name-sg = Ergebnis ,
6237     Name-pl = Ergebnissen ,
6238     case = G ,
6239     Name-sg = Ergebnisses ,
6240     Name-pl = Ergebnisse ,
6241
6242 type = remark ,
6243     gender = f ,
6244     case = N ,
6245     Name-sg = Bemerkung ,
6246     Name-pl = Bemerkungen ,
6247     case = A ,

```

```

6248     Name-sg = Bemerkung ,
6249     Name-pl = Bemerkungen ,
6250     case = D ,
6251     Name-sg = Bemerkung ,
6252     Name-pl = Bemerkungen ,
6253     case = G ,
6254     Name-sg = Bemerkung ,
6255     Name-pl = Bemerkungen ,
6256
6257 type = example ,
6258     gender = n ,
6259     case = N ,
6260     Name-sg = Beispiel ,
6261     Name-pl = Beispiele ,
6262     case = A ,
6263     Name-sg = Beispiel ,
6264     Name-pl = Beispiele ,
6265     case = D ,
6266     Name-sg = Beispiel ,
6267     Name-pl = Beispielen ,
6268     case = G ,
6269     Name-sg = Beispiels ,
6270     Name-pl = Beispiele ,
6271
6272 type = algorithm ,
6273     gender = m ,
6274     case = N ,
6275     Name-sg = Algorithmus ,
6276     Name-pl = Algorithmen ,
6277     case = A ,
6278     Name-sg = Algorithmus ,
6279     Name-pl = Algorithmen ,
6280     case = D ,
6281     Name-sg = Algorithmus ,
6282     Name-pl = Algorithmen ,
6283     case = G ,
6284     Name-sg = Algorithmus ,
6285     Name-pl = Algorithmen ,
6286
6287 type = listing ,
6288     gender = n ,
6289     case = N ,
6290     Name-sg = Listing ,
6291     Name-pl = Listings ,
6292     case = A ,
6293     Name-sg = Listing ,
6294     Name-pl = Listings ,
6295     case = D ,
6296     Name-sg = Listing ,
6297     Name-pl = Listings ,
6298     case = G ,
6299     Name-sg = Listings ,
6300     Name-pl = Listings ,
6301

```

```

6302 type = exercise ,
6303   gender = f ,
6304   case = N ,
6305     Name-sg = Übungsaufgabe ,
6306     Name-pl = Übungsaufgaben ,
6307   case = A ,
6308     Name-sg = Übungsaufgabe ,
6309     Name-pl = Übungsaufgaben ,
6310   case = D ,
6311     Name-sg = Übungsaufgabe ,
6312     Name-pl = Übungsaufgaben ,
6313   case = G ,
6314     Name-sg = Übungsaufgabe ,
6315     Name-pl = Übungsaufgaben ,
6316
6317 type = solution ,
6318   gender = f ,
6319   case = N ,
6320     Name-sg = Lösung ,
6321     Name-pl = Lösungen ,
6322   case = A ,
6323     Name-sg = Lösung ,
6324     Name-pl = Lösungen ,
6325   case = D ,
6326     Name-sg = Lösung ,
6327     Name-pl = Lösungen ,
6328   case = G ,
6329     Name-sg = Lösung ,
6330     Name-pl = Lösungen ,
6331 </lang-german>

```

## 10.4 French

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

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

```

6332 <*package>
6333 \zcDeclareLanguage [ gender = { f , m } ] { french }
6334 \zcDeclareLanguageAlias { acadian } { french }
6335 </package>
6336 <*lang-french>
6337 namesep = {\nobreakspace} ,
6338 pairsep = {\~et\nobreakspace} ,
6339 listsep = { , ~ } ,
6340 lastsep = {\~et\nobreakspace} ,
6341 tpairsep = {\~et\nobreakspace} ,
6342 tlistsep = { , ~ } ,
6343 tlastsep = {\~et\nobreakspace} ,

```

```

6344 notesep = {-} ,
6345 rangeseq = {-\nbreakspace} ,
6346
6347 type = book ,
6348   gender = m ,
6349   Name-sg = Livre ,
6350   name-sg = livre ,
6351   Name-pl = Livres ,
6352   name-pl = livres ,
6353
6354 type = part ,
6355   gender = f ,
6356   Name-sg = Partie ,
6357   name-sg = partie ,
6358   Name-pl = Parties ,
6359   name-pl = parties ,
6360
6361 type = chapter ,
6362   gender = m ,
6363   Name-sg = Chapitre ,
6364   name-sg = chapitre ,
6365   Name-pl = Chapitres ,
6366   name-pl = chapitres ,
6367
6368 type = section ,
6369   gender = f ,
6370   Name-sg = Section ,
6371   name-sg = section ,
6372   Name-pl = Sections ,
6373   name-pl = sections ,
6374
6375 type = paragraph ,
6376   gender = m ,
6377   Name-sg = Paragraphe ,
6378   name-sg = paragraphe ,
6379   Name-pl = Paragraphes ,
6380   name-pl = paragraphes ,
6381
6382 type = appendix ,
6383   gender = f ,
6384   Name-sg = Annexe ,
6385   name-sg = annexe ,
6386   Name-pl = Annexes ,
6387   name-pl = annexes ,
6388
6389 type = page ,
6390   gender = f ,
6391   Name-sg = Page ,
6392   name-sg = page ,
6393   Name-pl = Pages ,
6394   name-pl = pages ,
6395   rangeseq = {-} ,
6396   rangetopair = false ,
6397

```



```

6398 type = line ,
6399   gender = f ,
6400   Name-sg = Ligne ,
6401   name-sg = ligne ,
6402   Name-pl = Lignes ,
6403   name-pl = lignes ,
6404
6405 type = figure ,
6406   gender = f ,
6407   Name-sg = Figure ,
6408   name-sg = figure ,
6409   Name-pl = Figures ,
6410   name-pl = figures ,
6411
6412 type = table ,
6413   gender = f ,
6414   Name-sg = Table ,
6415   name-sg = table ,
6416   Name-pl = Tables ,
6417   name-pl = tables ,
6418
6419 type = item ,
6420   gender = m ,
6421   Name-sg = Point ,
6422   name-sg = point ,
6423   Name-pl = Points ,
6424   name-pl = points ,
6425
6426 type = footnote ,
6427   gender = f ,
6428   Name-sg = Note ,
6429   name-sg = note ,
6430   Name-pl = Notes ,
6431   name-pl = notes ,
6432
6433 type = endnote ,
6434   gender = f ,
6435   Name-sg = Note ,
6436   name-sg = note ,
6437   Name-pl = Notes ,
6438   name-pl = notes ,
6439
6440 type = note ,
6441   gender = f ,
6442   Name-sg = Note ,
6443   name-sg = note ,
6444   Name-pl = Notes ,
6445   name-pl = notes ,
6446
6447 type = equation ,
6448   gender = f ,
6449   Name-sg = Équation ,
6450   name-sg = équation ,
6451   Name-pl = Équations ,

```

```

6452 name-pl = équations ,
6453 refbounds-first-sg = {,(,)},
6454 refbounds = {(,,)} ,
6455
6456 type = theorem ,
6457 gender = m ,
6458 Name-sg = Théorème ,
6459 name-sg = théorème ,
6460 Name-pl = Théorèmes ,
6461 name-pl = théorèmes ,
6462
6463 type = lemma ,
6464 gender = m ,
6465 Name-sg = Lemme ,
6466 name-sg = lemme ,
6467 Name-pl = Lemmes ,
6468 name-pl = lemmes ,
6469
6470 type = corollary ,
6471 gender = m ,
6472 Name-sg = Corollaire ,
6473 name-sg = corollaire ,
6474 Name-pl = Corollaires ,
6475 name-pl = corollaires ,
6476
6477 type = proposition ,
6478 gender = f ,
6479 Name-sg = Proposition ,
6480 name-sg = proposition ,
6481 Name-pl = Propositions ,
6482 name-pl = propositions ,
6483
6484 type = definition ,
6485 gender = f ,
6486 Name-sg = Définition ,
6487 name-sg = définition ,
6488 Name-pl = Définitions ,
6489 name-pl = définitions ,
6490
6491 type = proof ,
6492 gender = f ,
6493 Name-sg = Démonstration ,
6494 name-sg = démonstration ,
6495 Name-pl = Démonstrations ,
6496 name-pl = démonstrations ,
6497
6498 type = result ,
6499 gender = m ,
6500 Name-sg = Résultat ,
6501 name-sg = résultat ,
6502 Name-pl = Résultats ,
6503 name-pl = résultats ,
6504
6505 type = remark ,

```

```

6506   gender = f ,
6507   Name-sg = Remarque ,
6508   name-sg = remarque ,
6509   Name-pl = Remarques ,
6510   name-pl = remarques ,
6511
6512   type = example ,
6513   gender = m ,
6514   Name-sg = Exemple ,
6515   name-sg = exemple ,
6516   Name-pl = Exemples ,
6517   name-pl = exemples ,
6518
6519   type = algorithm ,
6520   gender = m ,
6521   Name-sg = Algorithme ,
6522   name-sg = algorithme ,
6523   Name-pl = Algorithmes ,
6524   name-pl = algorithmes ,
6525
6526   type = listing ,
6527   gender = m ,
6528   Name-sg = Listing ,
6529   name-sg = listing ,
6530   Name-pl = Listings ,
6531   name-pl = listings ,
6532
6533   type = exercise ,
6534   gender = m ,
6535   Name-sg = Exercice ,
6536   name-sg = exercice ,
6537   Name-pl = Exercices ,
6538   name-pl = exercices ,
6539
6540   type = solution ,
6541   gender = f ,
6542   Name-sg = Solution ,
6543   name-sg = solution ,
6544   Name-pl = Solutions ,
6545   name-pl = solutions ,
6546 </lang-french>

```

## 10.5 Portuguese

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

```

6547 <*package>
6548 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6549 \zcDeclareLanguageAlias { brazilian } { portuguese }
6550 \zcDeclareLanguageAlias { brazil   } { portuguese }
6551 \zcDeclareLanguageAlias { portuges } { portuguese }
6552 </package>

```

```

6553 <*lang-portuguese>
6554 namesep = {\nobreakspace} ,
6555 pairsep = {\nobreakspace} ,
6556 listsep = {,~} ,
6557 lastsep = {\nobreakspace} ,
6558 tpairsep = {\nobreakspace} ,
6559 tlistsep = {,~} ,
6560 tlastsep = {\nobreakspace} ,
6561 notesep = {~} ,
6562 rangesep = {\nobreakspace} ,
6563
6564 type = book ,
6565     gender = m ,
6566     Name-sg = Livro ,
6567     name-sg = livro ,
6568     Name-pl = Livros ,
6569     name-pl = livros ,
6570
6571 type = part ,
6572     gender = f ,
6573     Name-sg = Parte ,
6574     name-sg = parte ,
6575     Name-pl = Partes ,
6576     name-pl = partes ,
6577
6578 type = chapter ,
6579     gender = m ,
6580     Name-sg = Capítulo ,
6581     name-sg = capítulo ,
6582     Name-pl = Capítulos ,
6583     name-pl = capítulos ,
6584
6585 type = section ,
6586     gender = f ,
6587     Name-sg = Seção ,
6588     name-sg = seção ,
6589     Name-pl = Seções ,
6590     name-pl = seções ,
6591
6592 type = paragraph ,
6593     gender = m ,
6594     Name-sg = Parágrafo ,
6595     name-sg = parágrafo ,
6596     Name-pl = Parágrafos ,
6597     name-pl = parágrafos ,
6598     Name-sg-ab = Par. ,
6599     name-sg-ab = par. ,
6600     Name-pl-ab = Par. ,
6601     name-pl-ab = par. ,
6602
6603 type = appendix ,
6604     gender = m ,
6605     Name-sg = Apêndice ,
6606     name-sg = apêndice ,

```

```

6607 Name-pl = Apêndices ,
6608 name-pl = apêndices ,
6609
6610 type = page ,
6611 gender = f ,
6612 Name-sg = Página ,
6613 name-sg = página ,
6614 Name-pl = Páginas ,
6615 name-pl = páginas ,
6616 rangesep = {\textendash} ,
6617 rangetopair = false ,
6618
6619 type = line ,
6620 gender = f ,
6621 Name-sg = Linha ,
6622 name-sg = linha ,
6623 Name-pl = Linhas ,
6624 name-pl = linhas ,
6625
6626 type = figure ,
6627 gender = f ,
6628 Name-sg = Figura ,
6629 name-sg = figura ,
6630 Name-pl = Figuras ,
6631 name-pl = figuras ,
6632 Name-sg-ab = Fig. ,
6633 name-sg-ab = fig. ,
6634 Name-pl-ab = Figs. ,
6635 name-pl-ab = figs. ,
6636
6637 type = table ,
6638 gender = f ,
6639 Name-sg = Tabela ,
6640 name-sg = tabela ,
6641 Name-pl = Tabelas ,
6642 name-pl = tabelas ,
6643
6644 type = item ,
6645 gender = m ,
6646 Name-sg = Item ,
6647 name-sg = item ,
6648 Name-pl = Itens ,
6649 name-pl = itens ,
6650
6651 type = footnote ,
6652 gender = f ,
6653 Name-sg = Nota ,
6654 name-sg = nota ,
6655 Name-pl = Notas ,
6656 name-pl = notas ,
6657
6658 type = endnote ,
6659 gender = f ,
6660 Name-sg = Nota ,

```

```

6661 name-sg = nota ,
6662 Name-pl = Notas ,
6663 name-pl = notas ,
6664
6665 type = note ,
6666 gender = f ,
6667 Name-sg = Nota ,
6668 name-sg = nota ,
6669 Name-pl = Notas ,
6670 name-pl = notas ,
6671
6672 type = equation ,
6673 gender = f ,
6674 Name-sg = Equação ,
6675 name-sg = equação ,
6676 Name-pl = Equações ,
6677 name-pl = equações ,
6678 Name-sg-ab = Eq. ,
6679 name-sg-ab = eq. ,
6680 Name-pl-ab = Eqs. ,
6681 name-pl-ab = eqs. ,
6682 refbounds-first-sg = {,(,)}, ,
6683 refbounds = {(,,)} ,
6684
6685 type = theorem ,
6686 gender = m ,
6687 Name-sg = Teorema ,
6688 name-sg = teorema ,
6689 Name-pl = Teoremas ,
6690 name-pl = teoremas ,
6691
6692 type = lemma ,
6693 gender = m ,
6694 Name-sg = Lema ,
6695 name-sg = lema ,
6696 Name-pl = Lemas ,
6697 name-pl = lemas ,
6698
6699 type = corollary ,
6700 gender = m ,
6701 Name-sg = Corolário ,
6702 name-sg = corolário ,
6703 Name-pl = Corolários ,
6704 name-pl = corolários ,
6705
6706 type = proposition ,
6707 gender = f ,
6708 Name-sg = Proposição ,
6709 name-sg = proposição ,
6710 Name-pl = Proposições ,
6711 name-pl = proposições ,
6712
6713 type = definition ,
6714 gender = f ,

```

6715 Name-sg = Definição ,  
6716 name-sg = definição ,  
6717 Name-pl = Definições ,  
6718 name-pl = definições ,  
6719  
6720 type = proof ,  
6721 gender = f ,  
6722 Name-sg = Demonstração ,  
6723 name-sg = demonstração ,  
6724 Name-pl = Demonstrações ,  
6725 name-pl = demonstrações ,  
6726  
6727 type = result ,  
6728 gender = m ,  
6729 Name-sg = Resultado ,  
6730 name-sg = resultado ,  
6731 Name-pl = Resultados ,  
6732 name-pl = resultados ,  
6733  
6734 type = remark ,  
6735 gender = f ,  
6736 Name-sg = Observação ,  
6737 name-sg = observação ,  
6738 Name-pl = Observações ,  
6739 name-pl = observações ,  
6740  
6741 type = example ,  
6742 gender = m ,  
6743 Name-sg = Exemplo ,  
6744 name-sg = exemplo ,  
6745 Name-pl = Exemplos ,  
6746 name-pl = exemplos ,  
6747  
6748 type = algorithm ,  
6749 gender = m ,  
6750 Name-sg = Algoritmo ,  
6751 name-sg = algoritmo ,  
6752 Name-pl = Algoritmos ,  
6753 name-pl = algoritmos ,  
6754  
6755 type = listing ,  
6756 gender = f ,  
6757 Name-sg = Listagem ,  
6758 name-sg = listagem ,  
6759 Name-pl = Listagens ,  
6760 name-pl = listagens ,  
6761  
6762 type = exercise ,  
6763 gender = m ,  
6764 Name-sg = Exercício ,  
6765 name-sg = exercício ,  
6766 Name-pl = Exercícios ,  
6767 name-pl = exercícios ,  
6768

```

6769 type = solution ,
6770   gender = f ,
6771   Name-sg = Solução ,
6772   name-sg = solução ,
6773   Name-pl = Soluções ,
6774   name-pl = soluções ,
6775 </lang-portuguese>

```

## 10.6 Spanish

Spanish language file has been initially provided by the author.

```

6776 <*package>
6777 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6778 </package>
6779 <*lang-spanish>
6780 namesep = {\nobreakspace} ,
6781 pairsep = {\~y\nobreakspace} ,
6782 listsep = { ,~ } ,
6783 lastsep = {\~y\nobreakspace} ,
6784 tpairsep = {\~y\nobreakspace} ,
6785 tlistsep = { ,~ } ,
6786 tlastsep = {\~y\nobreakspace} ,
6787 notesep = {~} ,
6788 rangesep = {\~a\nobreakspace} ,
6789
6790 type = book ,
6791   gender = m ,
6792   Name-sg = Libro ,
6793   name-sg = libro ,
6794   Name-pl = Libros ,
6795   name-pl = libros ,
6796
6797 type = part ,
6798   gender = f ,
6799   Name-sg = Parte ,
6800   name-sg = parte ,
6801   Name-pl = Partes ,
6802   name-pl = partes ,
6803
6804 type = chapter ,
6805   gender = m ,
6806   Name-sg = Capítulo ,
6807   name-sg = capítulo ,
6808   Name-pl = Capítulos ,
6809   name-pl = capítulos ,
6810
6811 type = section ,
6812   gender = f ,
6813   Name-sg = Sección ,
6814   name-sg = sección ,
6815   Name-pl = Secciones ,
6816   name-pl = secciones ,

```



```

6817
6818 type = paragraph ,
6819     gender = m ,
6820     Name-sg = Párrafo ,
6821     name-sg = párrafo ,
6822     Name-pl = Párrafos ,
6823     name-pl = párrafos ,
6824
6825 type = appendix ,
6826     gender = m ,
6827     Name-sg = Apéndice ,
6828     name-sg = apéndice ,
6829     Name-pl = Apéndices ,
6830     name-pl = apéndices ,
6831
6832 type = page ,
6833     gender = f ,
6834     Name-sg = Página ,
6835     name-sg = página ,
6836     Name-pl = Páginas ,
6837     name-pl = páginas ,
6838     rangeseq = {\textendash} ,
6839     rangetopair = false ,
6840
6841 type = line ,
6842     gender = f ,
6843     Name-sg = Línea ,
6844     name-sg = línea ,
6845     Name-pl = Líneas ,
6846     name-pl = líneas ,
6847
6848 type = figure ,
6849     gender = f ,
6850     Name-sg = Figura ,
6851     name-sg = figura ,
6852     Name-pl = Figuras ,
6853     name-pl = figuras ,
6854
6855 type = table ,
6856     gender = m ,
6857     Name-sg = Cuadro ,
6858     name-sg = cuadro ,
6859     Name-pl = Cuadros ,
6860     name-pl = cuadros ,
6861
6862 type = item ,
6863     gender = m ,
6864     Name-sg = Punto ,
6865     name-sg = punto ,
6866     Name-pl = Puntos ,
6867     name-pl = puntos ,
6868
6869 type = footnote ,
6870     gender = f ,

```

```

6871 Name-sg = Nota ,
6872 name-sg = nota ,
6873 Name-pl = Notas ,
6874 name-pl = notas ,
6875
6876 type = endnote ,
6877 gender = f ,
6878 Name-sg = Nota ,
6879 name-sg = nota ,
6880 Name-pl = Notas ,
6881 name-pl = notas ,
6882
6883 type = note ,
6884 gender = f ,
6885 Name-sg = Nota ,
6886 name-sg = nota ,
6887 Name-pl = Notas ,
6888 name-pl = notas ,
6889
6890 type = equation ,
6891 gender = f ,
6892 Name-sg = Ecuación ,
6893 name-sg = ecuación ,
6894 Name-pl = Ecuaciones ,
6895 name-pl = ecuaciones ,
6896 refbounds-first-sg = {,(,)},
6897 refbounds = {(,,)},
6898
6899 type = theorem ,
6900 gender = m ,
6901 Name-sg = Teorema ,
6902 name-sg = teorema ,
6903 Name-pl = Teoremas ,
6904 name-pl = teoremas ,
6905
6906 type = lemma ,
6907 gender = m ,
6908 Name-sg = Lema ,
6909 name-sg = lema ,
6910 Name-pl = Lemas ,
6911 name-pl = lemas ,
6912
6913 type = corollary ,
6914 gender = m ,
6915 Name-sg = Corolario ,
6916 name-sg = corolario ,
6917 Name-pl = Corolarios ,
6918 name-pl = corolarios ,
6919
6920 type = proposition ,
6921 gender = f ,
6922 Name-sg = Proposición ,
6923 name-sg = proposición ,
6924 Name-pl = Proposiciones ,

```

```

6925 name-pl = proposiciones ,
6926
6927 type = definition ,
6928   gender = f ,
6929   Name-sg = Definición ,
6930   name-sg = definición ,
6931   Name-pl = Definiciones ,
6932   name-pl = definiciones ,
6933
6934 type = proof ,
6935   gender = f ,
6936   Name-sg = Demostración ,
6937   name-sg = demostración ,
6938   Name-pl = Demostraciones ,
6939   name-pl = demostraciones ,
6940
6941 type = result ,
6942   gender = m ,
6943   Name-sg = Resultado ,
6944   name-sg = resultado ,
6945   Name-pl = Resultados ,
6946   name-pl = resultados ,
6947
6948 type = remark ,
6949   gender = f ,
6950   Name-sg = Observación ,
6951   name-sg = observación ,
6952   Name-pl = Observaciones ,
6953   name-pl = observaciones ,
6954
6955 type = example ,
6956   gender = m ,
6957   Name-sg = Ejemplo ,
6958   name-sg = ejemplo ,
6959   Name-pl = Ejemplos ,
6960   name-pl = ejemplos ,
6961
6962 type = algorithm ,
6963   gender = m ,
6964   Name-sg = Algoritmo ,
6965   name-sg = algoritmo ,
6966   Name-pl = Algoritmos ,
6967   name-pl = algoritmos ,
6968
6969 type = listing ,
6970   gender = m ,
6971   Name-sg = Listado ,
6972   name-sg = listado ,
6973   Name-pl = Listados ,
6974   name-pl = listados ,
6975
6976 type = exercise ,
6977   gender = m ,
6978   Name-sg = Ejercicio ,

```

```

6979 name-sg = ejercicio ,
6980 Name-pl = Ejercicios ,
6981 name-pl = ejercicios ,
6982
6983 type = solution ,
6984 gender = f ,
6985 Name-sg = Solución ,
6986 name-sg = solución ,
6987 Name-pl = Soluciones ,
6988 name-pl = soluciones ,
6989 </lang-spanish>

```

## 10.7 Dutch

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

```

6990 <*package>
6991 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6992 </package>
6993 <*lang-dutch>
6994 namesep = {\nobreakspace} ,
6995 pairsep = {-en\nobreakspace} ,
6996 listsep = {,~} ,
6997 lastsep = {-en\nobreakspace} ,
6998 tpairsep = {-en\nobreakspace} ,
6999 tlistsep = {,~} ,
7000 tlastsep = {,~en\nobreakspace} ,
7001 notesep = {-} ,
7002 rangesep = {-t/m\nobreakspace} ,
7003
7004 type = book ,
7005 gender = n ,
7006 Name-sg = Boek ,
7007 name-sg = boek ,
7008 Name-pl = Boeken ,
7009 name-pl = boeken ,
7010
7011 type = part ,
7012 gender = n ,
7013 Name-sg = Deel ,
7014 name-sg = deel ,
7015 Name-pl = Delen ,
7016 name-pl = delen ,
7017
7018 type = chapter ,
7019 gender = n ,
7020 Name-sg = Hoofdstuk ,
7021 name-sg = hoofdstuk ,
7022 Name-pl = Hoofdstukken ,
7023 name-pl = hoofdstukken ,
7024
7025 type = section ,

```

```

7026 gender = m ,
7027 Name-sg = Paragraaf ,
7028 name-sg = paragraaf ,
7029 Name-pl = Paragrafen ,
7030 name-pl = paragrafen ,

```

```

7031
7032 type = paragraph ,
7033 gender = f ,
7034 Name-sg = Alinea ,
7035 name-sg = alinea ,
7036 Name-pl = Alinea's ,
7037 name-pl = alinea's ,
7038

```

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

```

7039 type = appendix ,
7040 gender = { f , m } ,
7041 Name-sg = Blage ,
7042 name-sg = blage ,
7043 Name-pl = Blagen ,
7044 name-pl = blagen ,
7045

```

```

7046 type = page ,
7047 gender = { f , m } ,
7048 Name-sg = Pagina ,
7049 name-sg = pagina ,
7050 Name-pl = Pagina's ,
7051 name-pl = pagina's ,
7052 rangesep = {\textendash} ,
7053 rangetopair = false ,
7054

```

```

7055 type = line ,
7056 gender = m ,
7057 Name-sg = Regel ,
7058 name-sg = regel ,
7059 Name-pl = Regels ,
7060 name-pl = regels ,
7061

```

```

7062 type = figure ,
7063 gender = { n , f , m } ,
7064 Name-sg = Figuur ,
7065 name-sg = figuur ,
7066 Name-pl = Figuren ,
7067 name-pl = figuren ,
7068

```

```

7069 type = table ,
7070 gender = { f , m } ,
7071 Name-sg = Tabel ,
7072 name-sg = tabel ,
7073 Name-pl = Tabellen ,
7074 name-pl = tabellen ,
7075

```

```

7076 type = item ,
7077   gender = n ,
7078   Name-sg = Punt ,
7079   name-sg = punt ,
7080   Name-pl = Punten ,
7081   name-pl = punten ,
7082
7083 type = footnote ,
7084   gender = { f , m } ,
7085   Name-sg = Voetnoot ,
7086   name-sg = voetnoot ,
7087   Name-pl = Voetnoten ,
7088   name-pl = voetnoten ,
7089
7090 type = endnote ,
7091   gender = { f , m } ,
7092   Name-sg = Eindnoot ,
7093   name-sg = eindnoot ,
7094   Name-pl = Eindnoten ,
7095   name-pl = eindnoten ,
7096
7097 type = note ,
7098   gender = f ,
7099   Name-sg = Opmerking ,
7100   name-sg = opmerking ,
7101   Name-pl = Opmerkingen ,
7102   name-pl = opmerkingen ,
7103
7104 type = equation ,
7105   gender = f ,
7106   Name-sg = Vergelking ,
7107   name-sg = vergelking ,
7108   Name-pl = Vergelkingen ,
7109   name-pl = vergelkingen ,
7110   Name-sg-ab = Vgl. ,
7111   name-sg-ab = vgl. ,
7112   Name-pl-ab = Vgl.'s ,
7113   name-pl-ab = vgl.'s ,
7114   refbounds-first-sg = { ,(,) } ,
7115   refbounds = { (,,) } ,
7116
7117 type = theorem ,
7118   gender = f ,
7119   Name-sg = Stelling ,
7120   name-sg = stelling ,
7121   Name-pl = Stellingen ,
7122   name-pl = stellingen ,
7123
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".
7124 type = lemma ,
7125   gender = n ,

```

7126 Name-sg = Lemma ,  
7127 name-sg = lemma ,  
7128 Name-pl = Lemma's ,  
7129 name-pl = lemma's ,  
7130  
7131 type = corollary ,  
7132 gender = n ,  
7133 Name-sg = Gevolg ,  
7134 name-sg = gevolg ,  
7135 Name-pl = Gevolgen ,  
7136 name-pl = gevolgen ,  
7137  
7138 type = proposition ,  
7139 gender = f ,  
7140 Name-sg = Propositie ,  
7141 name-sg = propositie ,  
7142 Name-pl = Proposities ,  
7143 name-pl = proposities ,  
7144  
7145 type = definition ,  
7146 gender = f ,  
7147 Name-sg = Definitie ,  
7148 name-sg = definitie ,  
7149 Name-pl = Definities ,  
7150 name-pl = definities ,  
7151  
7152 type = proof ,  
7153 gender = n ,  
7154 Name-sg = Bews ,  
7155 name-sg = bews ,  
7156 Name-pl = Bewzen ,  
7157 name-pl = bewzen ,  
7158  
7159 type = result ,  
7160 gender = n ,  
7161 Name-sg = Resultaat ,  
7162 name-sg = resultaat ,  
7163 Name-pl = Resultaten ,  
7164 name-pl = resultaten ,  
7165  
7166 type = remark ,  
7167 gender = f ,  
7168 Name-sg = Opmerking ,  
7169 name-sg = opmerking ,  
7170 Name-pl = Opmerkingen ,  
7171 name-pl = opmerkingen ,  
7172  
7173 type = example ,  
7174 gender = n ,  
7175 Name-sg = Voorbeeld ,  
7176 name-sg = voorbeeld ,  
7177 Name-pl = Voorbeelden ,  
7178 name-pl = voorbeelden ,  
7179

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

```
7180 type = algorithm ,
7181   gender = { n , f , m } ,
7182   Name-sg = Algoritme ,
7183   name-sg = algoritme ,
7184   Name-pl = Algoritmen ,
7185   name-pl = algoritmen ,
7186
```

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

```
7187 type = listing ,
7188   gender = m ,
7189   Name-sg = Listing ,
7190   name-sg = listing ,
7191   Name-pl = Listings ,
7192   name-pl = listings ,
7193
7194 type = exercise ,
7195   gender = { f , m } ,
7196   Name-sg = Opgave ,
7197   name-sg = opgave ,
7198   Name-pl = Opgaven ,
7199   name-pl = opgaven ,
7200
7201 type = solution ,
7202   gender = f ,
7203   Name-sg = Oplossing ,
7204   name-sg = oplossing ,
7205   Name-pl = Oplossingen ,
7206   name-pl = oplossingen ,
7207 </lang-dutch>
```

## 10.8 Italian

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

```
7208 <*package>
7209 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7210 </package>
7211 <*lang-italian>
7212 namesep = {\nobreakspace} ,
7213 pairsep = {\~e\nobreakspace} ,
7214 listsep = { , ~ } ,
7215 lastsep = {\~e\nobreakspace} ,
7216 tpairsep = {\~e\nobreakspace} ,
7217 tlistsep = { , ~ } ,
7218 tlastsep = { , ~e\nobreakspace} ,
7219 notesep = { ~ } ,
7220 rangesep = {\~a\nobreakspace} ,
```



```

7221 +refbounds-rb = {da\nobreakspace,,} ,
7222
7223 type = book ,
7224     gender = m ,
7225     Name-sg = Libro ,
7226     name-sg = libro ,
7227     Name-pl = Libri ,
7228     name-pl = libri ,
7229
7230 type = part ,
7231     gender = f ,
7232     Name-sg = Parte ,
7233     name-sg = parte ,
7234     Name-pl = Parti ,
7235     name-pl = parti ,
7236
7237 type = chapter ,
7238     gender = m ,
7239     Name-sg = Capitolo ,
7240     name-sg = capitolo ,
7241     Name-pl = Capitoli ,
7242     name-pl = capitoli ,
7243
7244 type = section ,
7245     gender = m ,
7246     Name-sg = Paragrafo ,
7247     name-sg = paragrafo ,
7248     Name-pl = Paragrafi ,
7249     name-pl = paragrafi ,
7250
7251 type = paragraph ,
7252     gender = m ,
7253     Name-sg = Capoverso ,
7254     name-sg = capoverso ,
7255     Name-pl = Capoversi ,
7256     name-pl = capoversi ,
7257
7258 type = appendix ,
7259     gender = f ,
7260     Name-sg = Appendice ,
7261     name-sg = appendice ,
7262     Name-pl = Appendici ,
7263     name-pl = appendici ,
7264
7265 type = page ,
7266     gender = f ,
7267     Name-sg = Pagina ,
7268     name-sg = pagina ,
7269     Name-pl = Pagine ,
7270     name-pl = pagine ,
7271     Name-sg-ab = Pag. ,
7272     name-sg-ab = pag. ,
7273     Name-pl-ab = Pag. ,
7274     name-pl-ab = pag. ,

```

```

7275 rangeseq = {\textendash} ,
7276 rangetopair = false ,
7277 +refbounds-rb = {,,} ,
7278
7279 type = line ,
7280 gender = f ,
7281 Name-sg = Riga ,
7282 name-sg = riga ,
7283 Name-pl = Righe ,
7284 name-pl = righe ,
7285
7286 type = figure ,
7287 gender = f ,
7288 Name-sg = Figura ,
7289 name-sg = figura ,
7290 Name-pl = Figure ,
7291 name-pl = figure ,
7292 Name-sg-ab = Fig. ,
7293 name-sg-ab = fig. ,
7294 Name-pl-ab = Fig. ,
7295 name-pl-ab = fig. ,
7296
7297 type = table ,
7298 gender = f ,
7299 Name-sg = Tabella ,
7300 name-sg = tabella ,
7301 Name-pl = Tabelle ,
7302 name-pl = tabelle ,
7303 Name-sg-ab = Tab. ,
7304 name-sg-ab = tab. ,
7305 Name-pl-ab = Tab. ,
7306 name-pl-ab = tab. ,
7307
7308 type = item ,
7309 gender = m ,
7310 Name-sg = Punto ,
7311 name-sg = punto ,
7312 Name-pl = Punti ,
7313 name-pl = punti ,
7314
7315 type = footnote ,
7316 gender = f ,
7317 Name-sg = Nota ,
7318 name-sg = nota ,
7319 Name-pl = Note ,
7320 name-pl = note ,
7321
7322 type = endnote ,
7323 gender = f ,
7324 Name-sg = Nota ,
7325 name-sg = nota ,
7326 Name-pl = Note ,
7327 name-pl = note ,
7328

```

```

7329 type = note ,
7330   gender = f ,
7331   Name-sg = Nota ,
7332   name-sg = nota ,
7333   Name-pl = Note ,
7334   name-pl = note ,
7335
7336 type = equation ,
7337   gender = f ,
7338   Name-sg = Equazione ,
7339   name-sg = equazione ,
7340   Name-pl = Equazioni ,
7341   name-pl = equazioni ,
7342   Name-sg-ab = Eq. ,
7343   name-sg-ab = eq. ,
7344   Name-pl-ab = Eq. ,
7345   name-pl-ab = eq. ,
7346   +refbounds-rb = {da\nobreakspace(,,)} ,
7347   refbounds-first-sg = {(,)}, ,
7348   refbounds = {(,,)} ,
7349
7350 type = theorem ,
7351   gender = m ,
7352   Name-sg = Teorema ,
7353   name-sg = teorema ,
7354   Name-pl = Teoremi ,
7355   name-pl = teoremi ,
7356
7357 type = lemma ,
7358   gender = m ,
7359   Name-sg = Lemma ,
7360   name-sg = lemma ,
7361   Name-pl = Lemmi ,
7362   name-pl = lemmi ,
7363
7364 type = corollary ,
7365   gender = m ,
7366   Name-sg = Corollario ,
7367   name-sg = corollario ,
7368   Name-pl = Corollari ,
7369   name-pl = corollari ,
7370
7371 type = proposition ,
7372   gender = f ,
7373   Name-sg = Proposizione ,
7374   name-sg = proposizione ,
7375   Name-pl = Proposizioni ,
7376   name-pl = proposizioni ,
7377
7378 type = definition ,
7379   gender = f ,
7380   Name-sg = Definizione ,
7381   name-sg = definizione ,
7382   Name-pl = Definizioni ,

```

```

7383 name-pl = definizioni ,
7384
7385 type = proof ,
7386 gender = f ,
7387 Name-sg = Dimostrazione ,
7388 name-sg = dimostrazione ,
7389 Name-pl = Dimostrazioni ,
7390 name-pl = dimostrazioni ,
7391
7392 type = result ,
7393 gender = m ,
7394 Name-sg = Risultato ,
7395 name-sg = risultato ,
7396 Name-pl = Risultati ,
7397 name-pl = risultati ,
7398
7399 type = remark ,
7400 gender = f ,
7401 Name-sg = Osservazione ,
7402 name-sg = osservazione ,
7403 Name-pl = Osservazioni ,
7404 name-pl = osservazioni ,
7405
7406 type = example ,
7407 gender = m ,
7408 Name-sg = Esempio ,
7409 name-sg = esempio ,
7410 Name-pl = Esempi ,
7411 name-pl = esempi ,
7412
7413 type = algorithm ,
7414 gender = m ,
7415 Name-sg = Algoritmo ,
7416 name-sg = algoritmo ,
7417 Name-pl = Algoritmi ,
7418 name-pl = algoritmi ,
7419
7420 type = listing ,
7421 gender = m ,
7422 Name-sg = Listato ,
7423 name-sg = listato ,
7424 Name-pl = Listati ,
7425 name-pl = listati ,
7426
7427 type = exercise ,
7428 gender = m ,
7429 Name-sg = Esercizio ,
7430 name-sg = esercizio ,
7431 Name-pl = Esercizi ,
7432 name-pl = esercizi ,
7433
7434 type = solution ,
7435 gender = f ,
7436 Name-sg = Soluzione ,

```

7437 name-sg = soluzione ,  
 7438 Name-pl = Soluzioni ,  
 7439 name-pl = soluzioni ,  
 7440 </lang-italian>

## Index

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

**A**

\A ..... 1894  
 \AddToHook .... 108, 2049, 2093, 2116,  
     2147, 2149, 2187, 2260, 2302, 2465,  
     2478, 2486, 5351, 5387, 5404, 5406,  
     5411, 5448, 5466, 5500, 5551, 5576  
 \AddToHookWithArguments ..... 2440  
 \alph ..... 5481, 5567  
 \appendix ..... 2, 127–129, 137  
 \appendixname ..... 127, 138  
 \AtEndOfPackage ..... 2476

**B**

\babelname ..... 2103  
 \babelprovide ..... 30, 55  
 bool commands:  
 \bool\_gset\_false:N ..... 547  
 \bool\_gset\_true:N ..... 540, 2452  
 \bool\_if:NTF ... 391, 468, 506, 564,  
     1767, 1818, 2053, 2057, 2442, 2488,  
     3468, 3871, 4012, 4139, 4175, 4209,  
     4276, 4289, 4301, 4352, 4369, 4379,  
     4384, 4430, 4434, 4490, 4515, 4522,  
     4528, 4539, 4545, 4573, 4618, 4640,  
     4669, 4818, 4971, 4973, 5503, 5579  
 \bool\_if:nTF ..... 76,  
     3582, 3592, 3616, 3633, 3648, 3713,  
     3721, 4362, 4739, 4780, 4861, 4878  
 \bool\_if\_exist:NTF ..... 345, 355,  
     379, 389, 439, 456, 466, 491, 494,  
     502, 504, 518, 521, 528, 531, 538, 545  
 \bool\_lazy\_all:nTF ..... 4458, 4983  
 \bool\_lazy\_and:nnTF .....  
     49, 54, 2612, 3439,  
     3460, 4243, 4314, 4692, 4956, 4998  
 \bool\_lazy\_any:nTF ..... 5135, 5144  
 \bool\_lazy\_or:nnTF 1384, 1417, 1974,  
     2558, 2825, 3327, 3352, 3443, 4944  
 \bool\_new:N .....  
     126, 346, 356, 381, 440, 458,  
     496, 519, 522, 529, 532, 539, 546,  
     806, 1581, 1582, 1609, 1633, 1997,  
     2004, 2011, 2024, 2025, 2195, 2196,  
     2197, 2198, 2199, 2295, 2296, 2433,  
     2445, 3476, 3491, 3753, 3754, 3765,  
     3766, 3773, 3775, 3776, 3789, 3790,  
     3791, 3803, 3804, 5465, 5510, 5550  
 \bool\_set:Nn ..... 3436  
 \bool\_set\_eq:NN ..... 554  
 \bool\_set\_false:N .. 380, 457, 493,  
     495, 530, 1594, 1598, 1770, 1868,  
     1916, 1961, 2032, 2041, 2042, 2059,  
     2066, 2207, 2211, 2218, 2226, 2227,  
     2228, 2320, 2332, 3481, 3574, 3820,  
     3821, 3838, 3877, 3888, 4288, 4480,  
     4481, 4488, 4489, 4722, 5142, 5159  
 \bool\_set\_true:N .....  
     347, 357, 441, 520, 523,  
     533, 1588, 1589, 1593, 1599, 1792,  
     1814, 1880, 1882, 1920, 1922, 1934,  
     1936, 1965, 1967, 1986, 1988, 2031,  
     2036, 2037, 2205, 2212, 2217, 2234,  
     2236, 2238, 2241, 2242, 2243, 2308,  
     2313, 3588, 3598, 3602, 3624, 3639,  
     3654, 3678, 3846, 3872, 3878, 3882,  
     3889, 4035, 4046, 4057, 4107, 4143,  
     4179, 4213, 4230, 4311, 4345, 4517,  
     4577, 4623, 4645, 4674, 4932, 4939,  
     5061, 5120, 5158, 5195, 5202, 5203,  
     5221, 5238, 5240, 5479, 5517, 5564  
 \bool\_until\_do:Nn .....  
     1869, 1923, 1968, 3614, 3839  
 \bool\_while\_do:nn ..... 5609

**C**

\chaptername ..... 138  
 clist commands:  
 \clist\_map\_inline:nn .....  
     639, 687, 704, 1023,  
     2229, 2381, 2454, 2964, 5482, 5569  
 \counterwithin ..... 5  
 \crefstripprefix ..... 45  
 cs commands:  
 \cs\_generate\_variant:Nn ..... 73,

288, 294, 301, 312, 323, 334, 349, 359, 366, 373, 384, 408, 418, 443, 450, 461, 499, 525, 535, 542, 549, 977, 1851, 1890, 1944, 1996, 2620, 4774, 4812, 5178, 5282, 5315, 5348	
<code>\cs_if_exist:N</code> TF .....	24, 27, 66, 86, 5375, 5381
<code>\cs_if_exist_p:N</code> .....	51, 56, 4245, 4316, 4694, 5611
<code>\cs_if_exist_use:N</code> .....	5447, 5649
<code>\cs_new:Npn</code> .....	64, 74, 84, 95, 289, 295, 297, 299, 302, 313, 324, 336, 338, 726, 4735, 4775, 4813, 5168
<code>\cs_new_protected:Npn</code> .....	283, 340, 350, 360, 367, 374, 399, 409, 430, 432, 434, 444, 451, 489, 516, 526, 536, 543, 819, 921, 1530, 1549, 1753, 1852, 1900, 1945, 2484, 2618, 3431, 3495, 3537, 3548, 3561, 3691, 3743, 3805, 4019, 4484, 4731, 4733, 4927, 5162, 5179, 5250, 5283, 5316, 5518
<code>\cs_set_eq:NN</code> .....	728
<code>\cs_to_str:N</code> .....	339
<b>D</b>	
<code>\d</code> .....	1894
<code>\def</code> .....	3
<b>E</b>	
exp commands:	
<code>\exp_args:Ne</code> .....	931, 5523, 5531
<code>\exp_args:Nee</code> .....	1771, 1783, 1795, 1805, 1871, 1925, 1970, 5181, 5187, 5209, 5213, 5228
<code>\exp_args:NNe</code> .....	36, 39
<code>\exp_args:NNNo</code> .....	285
<code>\exp_args:NNNV</code> .....	1832, 1884, 1938, 1990
<code>\exp_args:NNo</code> .....	285, 291
<code>\exp_args:No</code> .....	291
<code>\exp_not:N</code> .....	114, 4386, 4399, 4402, 4405, 4751, 4755, 4763, 4767, 4792, 4795, 4803, 4806, 4834, 4837, 4841, 4846, 4852, 4855, 4867, 4870, 4898, 4903, 4913, 4918
<code>\exp_not:n</code> .....	292, 4051, 4074, 4082, 4100, 4113, 4117, 4152, 4161, 4183, 4191, 4198, 4222, 4236, 4253, 4263, 4305, 4328, 4338, 4387, 4398, 4403, 4404, 4561, 4584, 4596, 4630, 4652, 4661, 4681, 4702, 4712, 4752, 4764, 4793, 4794, 4804, 4805, 4835, 4836, 4838, 4842, 4853, 4854, 4856, 4864, 4868, 4869, 4872, 4899, 4914
<code>\ExplSyntaxOn</code> .....	30, 937
<b>F</b>	
<code>\figurename</code> .....	138
file commands:	
<code>\file_get:n</code> NTF .....	931
<code>\fmtversion</code> .....	5
<code>\footnote</code> .....	2
<b>G</b>	
group commands:	
<code>\group_begin:</code> .....	742, 923, 1769, 1856, 1904, 1949, 2888, 3433, 3447, 3479, 4386, 4402, 4751, 4763, 4792, 4803, 4834, 4841, 4852, 4867, 4898, 4913
<code>\group_end:</code> .....	754, 975, 1833, 1885, 1939, 1991, 2916, 3450, 3473, 3483, 4399, 4405, 4755, 4767, 4795, 4806, 4837, 4846, 4855, 4870, 4903, 4918
<b>H</b>	
<code>\hyperlink</code> .....	5165
<b>I</b>	
<code>\IfBooleanT</code> .....	3480
<code>\IfClassLoadedTF</code> .....	121
<code>\ifdraft</code> .....	2210
<code>\IfFormatAtLeastTF</code> .....	5, 6
<code>\ifoptionfinal</code> .....	2216
<code>\IfPackageAtLeastTF</code> .....	2306
<code>\IfPackageLoadedTF</code> .....	119
<code>\input</code> .....	30
int commands:	
<code>\int_case:n</code> NTF .....	4022, 4064, 4126, 4437, 4552, 4609
<code>\int_compare:n</code> NnTF .....	3625, 3640, 3655, 3667, 3679, 3699, 3701, 3745, 3919, 4042, 4070, 4092, 4147, 4217, 4422, 4424, 4501, 4531, 4590, 5191, 5197, 5217, 5223, 5627
<code>\int_compare_p:n</code> Nn .....	1387, 1420, 1975, 1977, 2561, 2828, 3330, 3355, 3715, 3723, 4462, 4948, 4959, 4988, 5155
<code>\int_gincr:N</code> .....	112
<code>\int_incr:N</code> .....	4477, 4509, 4521, 4523, 4538, 4540, 4544, 4546, 4558, 4581, 4593, 4627, 4649, 4658, 4678, 4729, 5625
<code>\int_new:N</code> .....	106, 127, 3492, 3493, 3755, 3756, 3757, 3770, 3771
<code>\int_rand:n</code> .....	310, 321, 332
<code>\int_set:Nn</code> .....	3700, 3702, 3706, 3709, 5608
<code>\int_to_roman:n</code> .....	5612, 5619, 5620, 5623
<code>\int_use:N</code> .....	52, 57, 62, 68, 116
<code>\int_zero:N</code> .....	3693, 3694, 3815, 3816, 3817, 3818, 3819, 4475, 4476, 4478, 4479, 4724, 4725



<code>\prg_return_false:</code> .....	4796, 4798, 4801, 4802, 4807, 4808, 119, 121, 393, 397, 426, 470, 474, 485, 508, 512, 557, 564, 565, 734, 1896
<code>\prg_return_true:</code> .....	472, 483, 507, 510, 555, 564, 733, 1895
<code>\prg_set_eq_conditional:NNn</code> ...	738
prop commands:	
<code>\prop_if_in:NnTF</code> .....	36
<code>\prop_if_in_p:Nn</code> .....	77
<code>\prop_item:Nn</code> .....	39, 78
<code>\prop_new:N</code> .....	2347, 2402
<code>\prop_put:Nnn</code> .....	1553
<code>\prop_remove:Nn</code> .....	1552
<code>\providecommand</code> .....	5
<code>\ProvidesExplPackage</code> .....	14
<code>\ProvidesFile</code> .....	30
<b>R</b>	
<code>\refstepcounter</code> .	3, 4, 130, 131, 133, 134
regex commands:	
<code>\regex_match:nnTF</code> .....	1894
<code>\renewlist</code> .....	134
<code>\RequirePackage</code> .....	16, 17, 18, 19, 2054
<code>\roman</code> .....	7
<b>S</b>	
<code>\scantokens</code> .....	128
seq commands:	
<code>\seq_clear:N</code> .....	455, 830, 867, 948, 961, 1022, 1630, 2555, 2822, 2899, 2912, 2963, 3497, 5309
<code>\seq_const_from_clist:Nn</code> .....	21
<code>\seq_count:N</code> .....	1388, 1402, 1421, 1436, 2562, 2572, 2829, 2842, 3331, 3345, 3356, 3370
<code>\seq_gclear:N</code> .	1381, 1414, 3324, 3349
<code>\seq_gconcat:NNN</code> .....	632, 636
<code>\seq_get_left:NN</code> .....	845, 856, 952, 1000, 2903, 2941, 3849
<code>\seq_gput_right:Nn</code> ...	963, 969, 2458
<code>\seq_gremove_all:Nn</code> .....	2490
<code>\seq_gset_eq:NN</code> .....	448, 1050
<code>\seq_gset_from_clist:Nn</code> .....	575, 584, 596, 611, 619, 780, 795
<code>\seq_gset_split:Nnn</code> .....	433
<code>\seq_if_empty:NTF</code> ..	831, 868, 949, 989, 1010, 2900, 2930, 2951, 3843, 5010
<code>\seq_if_exist:NTF</code> ..	436, 446, 453, 464
<code>\seq_if_in:NnTF</code> .....	849, 883, 927, 995, 1025, 2383, 2456, 2489, 2936, 2966, 3541, 5006
<code>\seq_item:Nn</code> 4745, 4750, 4756, 4758, 4761, 4762, 4768, 4769, 4786, 4791,	4796, 4798, 4801, 4802, 4807, 4808, 4839, 4840, 4847, 4849, 4884, 4896, 4904, 4907, 4911, 4912, 4919, 4920
<code>\seq_map_break:n</code> .....	98, 3734, 3737
<code>\seq_map_function:NN</code> .....	3500
<code>\seq_map_indexed_inline:Nn</code> .	44, 3695
<code>\seq_map_inline:Nn</code> ...	1064, 1089, 1326, 1365, 1443, 2480, 2493, 2541, 2578, 2627, 2639, 2805, 2848, 2987, 3011, 3274, 3314, 3376, 3731, 5521
<code>\seq_map_tokens:Nn</code> .....	80
<code>\seq_new:N</code> .....	124, 125, 437, 447, 572, 573, 574, 583, 595, 610, 618, 631, 635, 775, 790, 920, 1042, 1618, 2376, 2446, 3475, 3494, 3752, 3769, 3792, 3793, 3794, 3795, 3796, 3797, 3798, 3799, 3800, 3801, 3802
<code>\seq_pop_left:NN</code> .....	3841
<code>\seq_put_right:Nn</code> .....	1026, 2385, 2967, 3544
<code>\seq_reverse:N</code> .....	1624
<code>\seq_set_eq:NN</code> .....	438, 482, 3807, 4033, 4044, 4055, 4105, 4141, 4177, 4211, 4227, 4309, 4343, 4354, 4575, 4620, 4642, 4671
<code>\seq_set_from_clist:Nn</code> ...	1623, 3435
<code>\seq_set_split:Nnn</code> .....	431
<code>\seq_sort:Nn</code> .....	88, 3503
<code>\seq_use:Nn</code> .....	5024
<code>\setcounter</code> ..	5373, 5374, 5390, 5405, 5409
<code>\sidefootnote</code> .....	130
sort commands:	
<code>\sort_return_same:</code> .....	88, 92, 3510, 3515, 3589, 3609, 3630, 3645, 3659, 3684, 3719, 3734, 3750
<code>\sort_return_swapped:</code> .....	88, 92, 3523, 3599, 3608, 3629, 3644, 3660, 3683, 3727, 3737, 3749
<code>\space</code> .....	11
<code>\stepcounter</code> .....	133, 5389, 5408
str commands:	
<code>\str_case:nn</code> .....	45
<code>\str_case:nnTF</code> .....	1125, 1647, 2118, 2155, 2231, 2670, 3046
<code>\str_compare:nNnTF</code> .....	3605
<code>\str_if_eq:nnTF</code> .....	97, 4777
<code>\str_if_eq_p:nn</code>	5140, 5146, 5148, 5152
<code>\str_new:N</code> .....	2069
<code>\str_set:Nn</code> ...	2074, 2076, 2078, 2080
<code>\string</code> .....	5528, 5536
<code>\subparagraph</code> .....	136
<code>\subref</code> .....	135
<code>\subsections</code> .....	136
<code>\subsubsection</code> .....	61, 62



<code>\subsubsections</code>	136	<code>\textendash</code>	1547, 5740, 6004, 6616, 6838, 7052, 7275
<code>\subsubsection</code>	62	<code>\thechapter</code>	127
<b>T</b>			
<code>\tablename</code>	138	<code>\thelstnumber</code>	134
<code>\tag</code>	124, 131, 133	<code>\thepage</code>	7, 110, 113
TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:		<code>\thesection</code>	127
<code>\@Alph</code>	127	tl commands:	
<code>\@addtoreset</code>	5	<code>\c_novalue_tl</code>	695, 696, 697, 698, 699, 700, 701, 2498, 2546, 2644, 2810
<code>\@auxout</code>	5527, 5535	<code>\tl_clear:N</code>	354, 378, 839, 877, 890, 907, 916, 941, 950, 983, 2625, 2891, 2901, 2924, 3809, 3810, 3811, 3812, 3813, 3814, 3845, 4470, 4471, 4472, 4473, 4474, 4520, 4537, 4931, 4938, 4968, 5060, 5119, 5276
<code>\@bsphack</code>	924, 5520	<code>\tl_const:Nn</code>	1532
<code>\@capttype</code>	5447, 5649	<code>\tl_gclear:N</code>	371, 415
<code>\@chapapp</code>	127	<code>\tl_gset:Nn</code>	364, 405, 749, 764
<code>\@currentcounter</code>	2–5, 63, 130, 131, 134, 27, 28, 55, 56, 57, 2430, 2431	<code>\tl_gset_eq:NN</code>	113
<code>\@currentlabel</code>	3, 124, 134	<code>\tl_head:N</code>	3643, 3656, 3668, 3670, 3680, 3682
<code>\@elt</code>	5	<code>\tl_head:n</code>	1872, 1873, 1926, 1927, 1971, 1972, 1978
<code>\@esphack</code>	974, 5540	<code>\tl_if_empty:NTF</code>	34, 88, 833, 843, 870, 881, 902, 909, 1016, 1072, 1097, 1129, 1164, 1201, 1238, 1286, 1334, 1340, 1373, 1451, 1489, 1755, 1879, 1933, 2957, 2995, 3019, 3050, 3085, 3122, 3159, 3207, 3282, 3288, 3322, 3384, 3405, 3451, 4278, 4871, 4953, 4975, 5033, 5044, 5087
<code>\@ifl@t@r</code>	5	<code>\tl_if_empty:nTF</code>	743, 759, 982, 1274, 1551, 1560, 1723, 2451, 2746, 2923, 3195, 5164
<code>\@onlypreamble</code>	756, 770, 2918	<code>\tl_if_empty_p:N</code>	50, 55, 2614, 4244, 4315, 4693, 4986, 5000, 5139, 5149, 5153
<code>\bbl@loaded</code>	55	<code>\tl_if_empty_p:n</code>	1385, 1418, 2559, 2826, 3328, 3353, 3584, 3585, 3594, 3595, 3620, 3621, 3636, 3651
<code>\bbl@main@language</code>	24, 2098	<code>\tl_if_eq:NNTF</code>	110, 3555, 3578, 3885
<code>\c@lstnumber</code>	134	<code>\tl_if_eq:NnTF</code>	1781, 3498, 3530, 3705, 3708, 3733, 3736, 3853, 3901, 4935, 5185
<code>\c@page</code>	7	<code>\tl_if_eq:nnTF</code>	1771, 1783, 1795, 1805, 1871, 1925, 1970, 3697, 5181, 5187, 5209, 5213, 5228, 5523, 5531
<code>\caption@subtypehook</code>	5650	<code>\tl_if_exist:NTF</code>	342, 352, 362, 369, 376, 387, 403, 413, 732
<code>\hyper@@link</code>	113, 122	<code>\tl_if_exist_p:N</code>	2613
<code>\hyper@linkfile</code>	5166	<code>\tl_if_novalue:nTF</code>	2501, 2549, 2647, 2813
<code>\lst@AddToHook</code>	5599	<code>\tl_map_break:n</code>	98
<code>\ltx@gobble</code>	64	<code>\tl_map_tokens:Nn</code>	90
<code>\MT@newlabel</code>	5528, 5536		
<code>\protected@write</code>	5527, 5535		
<code>\zref@addprop</code>	21, 31, 46, 61, 63, 105, 117		
<code>\zref@default</code>	113, 114, 4732, 4734		
<code>\zref@extractdefault</code>	11, 12, 123, 286, 292, 296		
<code>\zref@ifpropundefined</code>	43, 1280, 1567, 1729, 2752, 3201, 5170, 5566		
<code>\zref@ifrefcontainsprop</code>	43, 45, 1757, 1765, 1824, 1842, 1854, 1902, 1947, 3906, 4737, 4820, 4874, 5173		
<code>\zref@ifrefundefined</code>	3505, 3507, 3519, 3874, 3876, 3881, 3898, 4280, 4291, 4492, 4815, 4929		
<code>\zref@label</code>	64, 2443		
<code>\zref@localaddprop</code>	5449, 5504, 5580, 5651		
<code>\ZREF@mainlist</code>	21, 31, 46, 61, 63, 105, 117, 5449, 5504, 5580, 5651		
<code>\zref@newprop</code>	6, 7, 20, 22, 32, 47, 62, 100, 116, 5446, 5481, 5567, 5648		
<code>\zref@refused</code>	3896		
<code>\zref@wrapper@babel</code>	64, 84, 2443, 3430		
<code>\zrefclever@required@kernel</code>	3, 4, 6, 11		

<code>\tl_new:N</code> . . . . .	107, 122, 123, 343, 353, 363, 370, 404, 414, 569, 570, 571, 721, 722, 723, 724, 748, 763, 1555, 2181, 2200, 2268, 2289, 2340, 2426, 3485, 3486, 3487, 3488, 3489, 3490, 3758, 3759, 3760, 3761, 3762, 3763, 3764, 3767, 3768, 3772, 3774, 3777, 3778, 3779, 3780, 3781, 3782, 3783, 3784, 3785, 3786, 3787, 3788	<code>\zcpageref</code> . . . . .	85, <a href="#">3477</a>
<code>\tl_put_left:Nn</code> .	4365, 4372, 4415, 5046, 5047, 5089, 5091, 5093, 5095	<code>\zcref</code> . . . . .	21, 65, 67, 84–88, 94, 96, 132, <a href="#">3429</a> , 3482
<code>\tl_put_right:Nn</code> .	4049, 4072, 4080, 4098, 4111, 4150, 4159, 4181, 4189, 4196, 4220, 4234, 4251, 4261, 4559, 4582, 4594, 4628, 4650, 4659, 4679, 4700, 4710, 4954, 4955, 4966, 5650	<code>\zcRefTypeSetup</code> . . . . .	21, 68, <a href="#">2621</a>
<code>\tl_reverse:N</code> . . . . .	3565, 3568	<code>\zcsetup</code> . . . . .	21, 55, 65, 67, 68, <a href="#">2616</a>
<code>\tl_set:Nn</code> .	285, 344, 725, 750, 940, 984, 996, 1564, 1571, 1573, 1762, 1830, 1834, 1847, 1858, 1863, 1875, 1877, 1886, 1888, 1906, 1911, 1929, 1931, 1940, 1942, 1951, 1956, 1981, 1983, 1992, 1994, 2097, 2098, 2103, 2104, 2107, 2108, 2122, 2128, 2133, 2159, 2165, 2170, 2623, 2892, 2925, 2937, 3672, 3674, 3855, 3856, 4029, 4031, 4303, 4326, 4336, 4382, 4505, 4507, 4518, 4535, 4950, 4951, 4964	<code>\zlabel</code> . . . . .	2, 64, <a href="#">129</a> , <a href="#">131</a> , <a href="#">133</a>
<code>\tl_set_eq:NN</code> . . . . .	423, 4468	zrefcheck commands:	
<code>\tl_show:N</code> . . . . .	4431	<code>\zrefcheck_zcref_beg_label:</code> . .	3442
<code>\tl_tail:N</code> . . . . .	3673, 3675	<code>\zrefcheck_zcref_end_label_-</code> maybe: . . . . .	3464
<code>\tl_tail:n</code> . . . . .	.. 1876, 1878, 1930, 1932, 1982, 1984	<code>\zrefcheck_zcref_run_checks_on_-</code> labels:n . . . . .	3465
<code>\tl_use:N</code> . . . . .	307, 318, 329, 765, 929, 934, 964, 966, 970	zrefclever commands:	
<b>U</b>		<code>\zrefclever_language_if_declared:n</code> . . . . .	738
use commands:	<code>\use:N</code> 25, 28, 811, 4247, 4322, 4696, 5342	<code>\zrefclever_language_if_declared:nTF</code> . . . . .	<a href="#">738</a>
<code>\UseHook</code> . . . . .	1857, 1905, 1950	<code>\zrefclever_language_varname:n</code> . . . . . . .	<a href="#">728</a> , <a href="#">728</a>
<b>V</b>		<code>\l_zrefclever_ref_language_tl</code> . .	<a href="#">724</a>
<code>\value</code> . . . . .	5390, 5409	zrefclever internal commands:	
<b>Z</b>		<code>\l_zrefclever_abbrev_bool</code> . . . . .	3777, 3964, 4957
<code>\Z</code> . . . . .	1894	<code>\l_zrefclever_amsmath_subequations_-</code> bool . . . . .	5465, 5479, 5503
<code>\zcDeclareLanguage</code> 13, 25, 26, <a href="#">740</a> , 5675, 5880, 6333, 6548, 6777, 6991, 7209		<code>\l_zrefclever_breqn_dgroup_bool</code> . . . . .	5550, 5564, 5579
<code>\zcDeclareLanguageAlias</code> . . . . .	.. 26, <a href="#">757</a> , 5676, 5677, 5678, 5679, 5680, 5681, 5682, 5883, 5884, 5885, 5886, 5887, 6334, 6549, 6550, 6551	<code>\l_zrefclever_cap_bool</code> . . . . .	3777, 3960, 4945
<code>\zcLanguageSetup</code> . . . . .	.. 21, 30, 68, 73, 74, 138, <a href="#">2886</a>	<code>\l_zrefclever_capfirst_bool</code> . . . . . . . .	2004, 2007, 4947
		<code>\_zrefclever_compat_module:nn</code> . . . . . . .	65, <a href="#">2484</a> , <a href="#">2484</a> , 5349, 5367, 5428, 5461, 5511, 5546, 5586, 5604, 5631, 5654
		<code>\_zrefclever_counter_reset_by:n</code> . . . . .	6, 62, 66, 68, 70, <a href="#">74</a> , 74, 5473, 5558
		<code>\_zrefclever_counter_reset_by_-</code> aux:nn . . . . .	81, 84
		<code>\_zrefclever_counter_reset_by_-</code> auxi:nnn . . . . .	91, 95
		<code>\l_zrefclever_counter_resetby_-</code> prop . . . . .	6, 62, 77, 78, 2402, 2414
		<code>\l_zrefclever_counter_resetters_-</code> seq . . . . .	5, 6, 62, 80, 2376, 2383, 2386
		<code>\l_zrefclever_counter_type_prop</code> . . . . .	4, 61, 36, 39, 2347, 2359
		<code>\l_zrefclever_current_counter_-</code> tl . . . . .	3, 5, 63, 20, 24, 25, 37, 40, 42, 50, 51, 52, 103, 2426, 2429
		<code>\l_zrefclever_current_language_-</code> tl . . . . .	24, 55, 722, 2097, 2103, 2107, 2123, 2160

\l\_\_zrefclever\_endrangefunc\_tl . . . . . 3977, 3981, 3985, 3989, 3993, 3997,  
. . . . . 3777, 3952, 4244, 4245, 4247,  
4315, 4316, 4322, 4693, 4694, 4696  
\l\_\_zrefclever\_endrangeprop\_tl . . . . .  
. . . . . 48, 1755, 1765, 3777, 3956  
\\_\_zrefclever\_extract:nnn . . . . .  
. . . . . 12, 295, 295, 1860, 1865,  
1908, 1913, 1953, 1958, 3626, 3628,  
3641, 3658, 3746, 3748, 5192, 5194,  
5198, 5200, 5218, 5220, 5224, 5226  
\\_\_zrefclever\_extract\_default:Nnnn . . . . .  
. . . . . 11, 283, 283, 288, 1759, 1820,  
1827, 1837, 1844, 3539, 3550, 3552,  
3563, 3566, 3569, 3571, 3859, 3862  
\\_\_zrefclever\_extract\_unexp:nnn . . . . .  
. . . . . 12, 123,  
289, 289, 294, 1773, 1777, 1785,  
1789, 1797, 1801, 1807, 1811, 4394,  
4748, 4753, 4765, 4789, 4830, 4843,  
4892, 4900, 4915, 5171, 5174, 5175,  
5182, 5183, 5188, 5189, 5210, 5211,  
5214, 5215, 5230, 5234, 5524, 5532  
\\_\_zrefclever\_extract\_url\_-  
unexp:n . . . . . 4390, 4747,  
4788, 4826, 4888, 5168, 5168, 5178  
\\_\_zrefclever\_get\_enclosing\_-  
counters\_value:n . . . . .  
. . . . . 6, 64, 64, 69, 73, 102  
\\_\_zrefclever\_get\_endrange\_-  
pagecomp:nnN . . . . . 1900, 1944  
\\_\_zrefclever\_get\_endrange\_-  
pagecomptwo:nnN . . . . . 1945, 1996  
\\_\_zrefclever\_get\_endrange\_-  
property:nnN . . . . . 45, 1753, 1851  
\\_\_zrefclever\_get\_endrange\_-  
stripprefix:nnN . . . . . 1852, 1890  
\\_\_zrefclever\_get\_ref:nN . . . . .  
. . . . . 113, 114, 4052, 4075, 4083, 4101,  
4114, 4118, 4153, 4162, 4184, 4192,  
4199, 4223, 4237, 4264, 4306, 4339,  
4374, 4562, 4585, 4597, 4631, 4653,  
4662, 4682, 4713, 4735, 4735, 4774  
\\_\_zrefclever\_get\_ref\_endrange:nnN . . . . .  
. . . . . 45, 114,  
115, 4254, 4329, 4703, 4775, 4775, 4812  
\\_\_zrefclever\_get\_ref\_first: . . . . .  
. . . . . 113, 114, 118, 4366, 4416, 4813, 4813  
\\_\_zrefclever\_get\_rf\_opt\_bool:nN 126  
\\_\_zrefclever\_get\_rf\_opt\_-  
bool:nnnnN . . . . . 20,  
3957, 3961, 3965, 5316, 5316, 5348  
\\_\_zrefclever\_get\_rf\_opt\_-  
seq:nnnN . . . . . 20, 125, 3969, 3973,  
3977, 3981, 3985, 3989, 3993, 3997,  
4001, 4005, 5002, 5283, 5283, 5315  
\\_\_zrefclever\_get\_rf\_opt\_tl:nnnN . . . . .  
. . . . . 20,  
22, 45, 125, 3453, 3824, 3828, 3832,  
3921, 3925, 3929, 3933, 3937, 3941,  
3945, 3949, 3953, 5250, 5250, 5282  
\\_\_zrefclever\_hyperlink:nnn . . . . .  
. . . . . 122, 4388,  
4746, 4787, 4824, 4886, 5162, 5162  
\l\_\_zrefclever\_hyperlink\_bool . . . . .  
. . . . . 2024, 2031, 2036, 2041, 2053, 2059,  
2066, 3481, 4741, 4782, 4880, 5137  
\l\_\_zrefclever\_hyperref\_warn\_-  
bool . . . . . 2025, 2032, 2037, 2042, 2057  
\\_\_zrefclever\_if\_class\_loaded:n . . . . .  
. . . . . 118, 120  
\\_\_zrefclever\_if\_class\_loaded:nTF . . . . .  
. . . . . 5430  
\\_\_zrefclever\_if\_package\_-  
loaded:n . . . . . 118, 118  
\\_\_zrefclever\_if\_package\_-  
loaded:nTF . . . . . 2051,  
2095, 2101, 2304, 5369, 5463,  
5513, 5548, 5588, 5606, 5633, 5656  
\\_\_zrefclever\_is\_integer\_rgx:n . . . . .  
. . . . . 1891, 1892, 1899  
\\_\_zrefclever\_is\_integer\_rgx:nTF . . . . .  
. . . . . 1917, 1919, 1962, 1964  
\l\_\_zrefclever\_label\_a\_tl . . . . .  
. . . . . 93, 3758, 3842, 3861, 3874, 3896,  
3898, 3904, 3907, 3913, 4030, 4052,  
4075, 4083, 4118, 4184, 4199, 4249,  
4255, 4264, 4296, 4306, 4324, 4330,  
4339, 4492, 4496, 4506, 4519, 4536,  
4562, 4598, 4662, 4698, 4704, 4713  
\l\_\_zrefclever\_label\_b\_tl . . . . .  
. . . . . 93, 3758,  
3845, 3850, 3864, 3876, 3881, 4496  
\l\_\_zrefclever\_label\_count\_int . . . . .  
. . . . . 93, 3755, 3815,  
3919, 4022, 4475, 4501, 4729, 4989  
\l\_\_zrefclever\_label\_enclval\_a\_-  
tl . . . . . 3485, 3563, 3565, 3620,  
3636, 3656, 3668, 3672, 3673, 3680  
\l\_\_zrefclever\_label\_enclval\_b\_-  
tl . . . . . 3485, 3566, 3568, 3621,  
3643, 3651, 3670, 3674, 3675, 3682  
\l\_\_zrefclever\_label\_extdoc\_a\_tl . . . . .  
. . . . . 3485, 3569, 3579, 3584, 3594, 3607  
\l\_\_zrefclever\_label\_extdoc\_b\_tl . . . . .  
. . . . . 3485, 3571, 3580, 3585, 3595, 3606  
\l\_\_zrefclever\_label\_type\_a\_tl . . . . .  
. . . . . 3454, 3485, 3540, 3542,

3545, 3551, 3556, 3705, 3733, 3825,  
3829, 3833, 3855, 3860, 3886, 3901,  
3922, 3926, 3930, 3934, 3938, 3942,  
3946, 3950, 3954, 3958, 3962, 3966,  
3970, 3974, 3978, 3982, 3986, 3990,  
3994, 3998, 4002, 4006, 4032, 4508  
\l\_\_zrefclever\_label\_type\_b\_tl . .  
. . . . . 3485, 3553,  
3557, 3708, 3736, 3856, 3863, 3887  
\\_zrefclever\_label\_type\_put\_-  
new\_right:n 86, 88, 3501, 3537, 3537  
\l\_\_zrefclever\_label\_types\_seq . .  
. . . . 88, 3494, 3497, 3541, 3544, 3731  
\l\_\_zrefclever\_labelhook\_bool . . . .  
. . . . . 2433, 2436, 2442  
\\_zrefclever\_labels\_in\_sequence:mn  
. . 48, 94, 123, 4294, 4495, 5179, 5179  
\l\_\_zrefclever\_lang\_decl\_case\_tl  
. . . . . 569,  
950, 953, 996, 1001, 1340, 1357,  
2901, 2904, 2937, 2942, 3288, 3305  
\l\_\_zrefclever\_lang\_declension\_-  
seq . . . . . 569,  
829, 830, 831, 845, 849, 856, 947,  
948, 949, 952, 989, 995, 1000, 2898,  
2899, 2900, 2903, 2930, 2936, 2941  
\l\_\_zrefclever\_lang\_gender\_seq . .  
. . . . 569, 866, 867, 868, 883, 960,  
961, 1010, 1025, 2911, 2912, 2951, 2966  
\\_zrefclever\_language\_if\_-  
declared:n . . . . 25, 730, 737, 739  
\\_zrefclever\_language\_if\_-  
declared:n(TF) . . . . . 25  
\\_zrefclever\_language\_if\_-  
declared:nTF 304, 315, 326, 730,  
745, 761, 821, 925, 2134, 2171, 2889  
\\_zrefclever\_language\_varname:n  
. . . . . 25, 307, 318,  
329, 726, 726, 729, 732, 748, 749,  
763, 764, 765, 929, 934, 964, 966, 970  
\l\_\_zrefclever\_last\_of\_type\_bool  
. . . . . 93, 3752, 3872,  
3877, 3878, 3882, 3888, 3889, 4012  
\l\_\_zrefclever\_lastsep\_tl . 3777,  
3936, 4082, 4117, 4161, 4198, 4236  
\l\_\_zrefclever\_link\_star\_bool . . . .  
. . 3436, 3475, 4742, 4783, 4881, 5138  
\l\_\_zrefclever\_listsep\_tl . . . . .  
. . . 3777, 3932, 4113, 4191, 4561,  
4584, 4596, 4630, 4652, 4661, 4681  
\g\_\_zrefclever\_loaded\_langfiles\_-  
seq . . . . . 920, 928, 963, 969  
\l\_\_zrefclever\_main\_language\_tl . .  
. . . . . 24,  
55, 723, 2098, 2104, 2108, 2129, 2166  
\\_zrefclever\_mathtools\_showonlyrefs:n  
. . . . . 3470, 5518  
\l\_\_zrefclever\_mathtools\_-  
showonlyrefs\_bool 3468, 5510, 5517  
\\_zrefclever\_name\_default: . . . . .  
. . . . . 4731, 4733, 4863  
\l\_\_zrefclever\_name\_format\_-  
fallback\_tl . . . . . 3764, 4964,  
4968, 5033, 5082, 5094, 5096, 5114  
\l\_\_zrefclever\_name\_format\_tl . . . .  
. . . 3764, 4950, 4951, 4954, 4955,  
4965, 4966, 5039, 5046, 5047, 5055,  
5063, 5073, 5090, 5091, 5104, 5124  
\l\_\_zrefclever\_name\_in\_link\_bool  
. . . . . 116,  
118, 3764, 4384, 4818, 5142, 5158, 5159  
\l\_\_zrefclever\_namefont\_tl 3777,  
3944, 4387, 4403, 4835, 4853, 4868  
\l\_\_zrefclever\_nameinlink\_str . . . .  
. . . . . 2069, 2074, 2076,  
2078, 2080, 5140, 5146, 5148, 5152  
\l\_\_zrefclever\_namesep\_tl . . . . .  
. . 3777, 3924, 4838, 4856, 4864, 4872  
\l\_\_zrefclever\_next\_is\_same\_bool  
. . . . . 94, 123, 3770,  
4489, 4522, 4539, 4545, 5203, 5241  
\l\_\_zrefclever\_next\_maybe\_range\_-  
bool . . . . .  
. 93, 123, 3770, 4288, 4301, 4488,  
4515, 4528, 5195, 5202, 5221, 5239  
\l\_\_zrefclever\_noabbrev\_first\_-  
bool . . . . . 2011, 2014, 4961  
\g\_\_zrefclever\_nocompat\_bool . . . .  
. . . . . 2445, 2452, 2488  
\l\_\_zrefclever\_nocompat\_bool . . . 64  
\g\_\_zrefclever\_nocompat\_modules\_-  
seq 2446, 2456, 2459, 2480, 2489, 2490  
\l\_\_zrefclever\_nocompat\_modules\_-  
seq . . . . . 65  
\l\_\_zrefclever\_nudge\_comptosing\_-  
bool . . . 2197, 2227, 2236, 2242, 4985  
\l\_\_zrefclever\_nudge\_enabled\_-  
bool . . . . . 2195, 2205, 2207,  
2211, 2212, 2217, 2218, 4460, 4971  
\l\_\_zrefclever\_nudge\_gender\_bool  
. . . . . 2199, 2228, 2238, 2243, 4999  
\l\_\_zrefclever\_nudge\_multitype\_-  
bool . . . 2196, 2226, 2234, 2241, 4461  
\l\_\_zrefclever\_nudge\_singular\_-  
bool . . . . . 2198, 2254, 4973  
\\_zrefclever\_opt\_bool\_get:NN . . . .  
. . . . . 550, 560

```

\__zrefclever_opt_bool_get:NN(TF)
    ..... 20
\__zrefclever_opt_bool_get:NNTF .
    ... 550, 5319, 5324, 5329, 5334, 5339
\__zrefclever_opt_bool_gset_-
    false:N ..... 19,
    516, 543, 549, 1498, 1515, 3407, 3415
\__zrefclever_opt_bool_gset_-
    true:N ..... 19,
    516, 536, 542, 1460, 1477, 3386, 3394
\__zrefclever_opt_bool_if:N 561, 568
\__zrefclever_opt_bool_if:N(TF) . 20
\__zrefclever_opt_bool_if:NNTF ...
    ..... 561, 894
\__zrefclever_opt_bool_if_set:N .
    ..... 500, 515
\__zrefclever_opt_bool_if_-
    set:N(TF) ..... 18
\__zrefclever_opt_bool_if_-
    set:NNTF ..... 500,
    552, 563, 1453, 1469, 1491, 1507
\__zrefclever_opt_bool_set_-
    false:N 19, 516, 526, 535, 2591, 2865
\__zrefclever_opt_bool_set_-
    true:N 19, 516, 516, 525, 2586, 2856
\__zrefclever_opt_bool_unset:N .
    ..... 18, 489, 489, 499, 2596, 2874
\__zrefclever_opt_seq_get:NN 478, 488
\__zrefclever_opt_seq_get:NN(TF) 18
\__zrefclever_opt_seq_get:NNTF .
    ... 478, 824, 861, 942, 955, 2893,
    2906, 5286, 5291, 5296, 5301, 5306
\__zrefclever_opt_seq_gset_-
    clist_split:Nn ..... 17,
    430, 432, 1382, 1415, 3325, 3350
\__zrefclever_opt_seq_gset_eq:NN
    ..... 17, 430,
    444, 450, 1391, 1424, 2974, 3334, 3359
\__zrefclever_opt_seq_if_set:N .
    ..... 462, 477
\__zrefclever_opt_seq_if_-
    set:N(TF) ..... 17
\__zrefclever_opt_seq_if_set:NNTF
    ..... 462, 480, 1033, 1375, 1407
\__zrefclever_opt_seq_set_clist_-
    split:Nn .. 17, 430, 430, 2556, 2823
\__zrefclever_opt_seq_set_eq:NN .
    ..... 17, 430, 434, 443, 2565, 2832
\__zrefclever_opt_seq_unset:N ...
    ..... 17, 451, 451, 461, 2551, 2815
\__zrefclever_opt_tl_clear:N ...
    ..... 14, 340,
    350, 359, 1651, 1656, 1671, 1686,
    1701, 2674, 2679, 2694, 2709, 2724
\__zrefclever_opt_tl_cset_-
    fallback:nn ..... 1530, 1537
\__zrefclever_opt_tl_gclear:N ...
    ..... 14, 340,
    367, 373, 3052, 3058, 3066, 3073,
    3094, 3110, 3131, 3147, 3168, 3184
\__zrefclever_opt_tl_gclear_if_-
    new:N ..... 16, 399,
    409, 418, 1131, 1137, 1145, 1152,
    1173, 1189, 1210, 1226, 1247, 1263
\__zrefclever_opt_tl_get:NN 419, 429
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF ...
    419, 5035, 5050, 5069, 5078, 5099,
    5109, 5253, 5258, 5263, 5268, 5273
\__zrefclever_opt_tl_gset:N .... 14
\__zrefclever_opt_tl_gset:Nn ...
    .. 340, 360, 366, 2997, 3021, 3029,
    3087, 3102, 3124, 3139, 3161, 3176,
    3209, 3216, 3225, 3233, 3290, 3300
\__zrefclever_opt_tl_gset_if_-
    new:Nn ..... 16,
    399, 399, 408, 1074, 1099, 1108,
    1166, 1181, 1203, 1218, 1240, 1255,
    1288, 1295, 1304, 1312, 1342, 1352
\__zrefclever_opt_tl_if_set:N .. 385
\__zrefclever_opt_tl_if_set:N(TF)
    ..... 15
\__zrefclever_opt_tl_if_set:NNTF .
    ..... 385, 401, 411, 421
\__zrefclever_opt_tl_set:N ..... 14
\__zrefclever_opt_tl_set:Nn ....
    ..... 340, 340, 349,
    1665, 1680, 1695, 1735, 1741, 2507,
    2656, 2688, 2703, 2718, 2758, 2765
\__zrefclever_opt_tl_unset:N ...
    ..... 15, 374, 374,
    384, 1710, 1715, 2503, 2649, 2733, 2738
\__zrefclever_opt_var_set_bool:n
    ..... 14, 338, 338, 345,
    346, 347, 355, 356, 357, 379, 380,
    381, 389, 391, 439, 440, 441, 456,
    457, 458, 466, 468, 494, 495, 496,
    504, 506, 521, 522, 523, 531, 532, 533
\__zrefclever_opt_varname_-
    fallback:nn ..... 14,
    336, 336, 1533, 5274, 5307, 5340
\__zrefclever_opt_varname_-
    general:nn ..... 12,
    297, 297, 1653, 1658, 1667, 1673,
    1682, 1688, 1697, 1703, 1712, 1717,
    1737, 1743, 2504, 2508, 2552, 2566,
    2587, 2592, 2597, 5254, 5287, 5320

```

`\__zrefclever_opt_varname_lang_-`  
`default:nnn` . . . [13](#), [313](#), [313](#), [323](#),  
[1076](#), [1101](#), [1133](#), [1139](#), [1168](#), [1175](#),  
[1205](#), [1212](#), [1242](#), [1249](#), [1290](#), [1297](#),  
[1377](#), [1393](#), [1455](#), [1462](#), [1493](#), [1500](#),  
[2999](#), [3023](#), [3054](#), [3060](#), [3089](#), [3096](#),  
[3126](#), [3133](#), [3163](#), [3170](#), [3211](#), [3218](#),  
[3336](#), [3388](#), [3409](#), [5269](#), [5302](#), [5335](#)  
`\__zrefclever_opt_varname_lang_-`  
`type:nnnn` . . . . . [14](#),  
[324](#), [324](#), [335](#), [1035](#), [1044](#), [1052](#),  
[1110](#), [1147](#), [1154](#), [1183](#), [1191](#), [1220](#),  
[1228](#), [1257](#), [1265](#), [1306](#), [1314](#), [1344](#),  
[1354](#), [1409](#), [1426](#), [1471](#), [1479](#), [1509](#),  
[1517](#), [2976](#), [3031](#), [3068](#), [3075](#), [3104](#),  
[3112](#), [3141](#), [3149](#), [3178](#), [3186](#), [3227](#),  
[3235](#), [3292](#), [3302](#), [3361](#), [3396](#), [3417](#),  
[5052](#), [5101](#), [5111](#), [5264](#), [5297](#), [5330](#)  
`\__zrefclever_opt_varname_-`  
`language:nnn` . . . . . [13](#), [302](#),  
[302](#), [312](#), [777](#), [782](#), [792](#), [797](#), [808](#),  
[813](#), [826](#), [863](#), [896](#), [944](#), [957](#), [2895](#), [2908](#)  
`\__zrefclever_opt_varname_-`  
`type:nnn` [13](#), [299](#), [299](#), [301](#), [2651](#),  
[2658](#), [2676](#), [2681](#), [2690](#), [2696](#), [2705](#),  
[2711](#), [2720](#), [2726](#), [2735](#), [2740](#), [2760](#),  
[2767](#), [2817](#), [2834](#), [2858](#), [2867](#), [2876](#),  
[5037](#), [5071](#), [5080](#), [5259](#), [5292](#), [5325](#)  
`\g_zrefclever_page_format_int` . .  
. . . . . [106](#), [112](#), [116](#)  
`\l_zrefclever_pairsep_tl` . . . . .  
. . . . . [3777](#), [3928](#), [4051](#),  
[4074](#), [4100](#), [4152](#), [4183](#), [4222](#), [4305](#)  
`\g_zrefclever_prev_page_format_-`  
`tl` . . . . . [7](#), [107](#), [110](#), [113](#)  
`\__zrefclever_process_language_-`  
`settings:` . . . [57](#), [59](#), [819](#), [819](#), [3438](#)  
`\__zrefclever_prop_put_non_-`  
`empty:Nnn` [43](#), [1549](#), [1549](#), [2358](#), [2413](#)  
`\__zrefclever_provide_langfile:n`  
[21](#), [30-32](#), [84](#), [921](#), [921](#), [977](#), [2140](#), [3437](#)  
`\l_zrefclever_range_beg_is_-`  
`first_bool` . . . . . [3770](#),  
[3820](#), [4139](#), [4175](#), [4209](#), [4480](#),  
[4517](#), [4573](#), [4618](#), [4640](#), [4669](#), [4722](#)  
`\l_zrefclever_range_beg_label_-`  
`tl` . . . . . [93](#),  
[3770](#), [3813](#), [4102](#), [4115](#), [4154](#), [4163](#),  
[4193](#), [4224](#), [4238](#), [4248](#), [4473](#), [4518](#),  
[4535](#), [4586](#), [4632](#), [4654](#), [4683](#), [4697](#)  
`\l_zrefclever_range_count_int` . .  
. . . . . [93](#),  
[3770](#), [3818](#), [4064](#), [4128](#), [4478](#), [4521](#),  
[4532](#), [4538](#), [4544](#), [4552](#), [4611](#), [4724](#)  
`\l_zrefclever_range_end_ref_tl` .  
. . . [3770](#), [3814](#), [4250](#), [4256](#), [4325](#),  
[4331](#), [4474](#), [4520](#), [4537](#), [4699](#), [4705](#)  
`\l_zrefclever_range_same_count_-`  
`int` . . . . . [93](#),  
[3770](#), [3819](#), [4042](#), [4093](#), [4129](#), [4479](#),  
[4523](#), [4540](#), [4546](#), [4591](#), [4612](#), [4725](#)  
`\l_zrefclever_rangeseq_tl` . . . . .  
. . . . . [3777](#), [3940](#),  
[4253](#), [4263](#), [4328](#), [4338](#), [4702](#), [4712](#)  
`\l_zrefclever_rangetopair_bool` .  
. . . . . [3777](#), [3968](#), [4289](#)  
`\l_zrefclever_ref_count_int` . . .  
. . . . . [3755](#), [3817](#),  
[4070](#), [4148](#), [4218](#), [4476](#), [4509](#), [4558](#),  
[4581](#), [4593](#), [4627](#), [4649](#), [4658](#), [4678](#)  
`\l_zrefclever_ref_decl_case_tl` .  
. . [27](#), [28](#), [833](#), [838](#), [839](#), [843](#), [846](#),  
[850](#), [854](#), [857](#), [902](#), [905](#), [907](#), [2181](#),  
[2191](#), [5044](#), [5048](#), [5087](#), [5092](#), [5097](#)  
`\__zrefclever_ref_default:` [4731](#),  
[4731](#), [4772](#), [4778](#), [4816](#), [4857](#), [4923](#)  
`\l_zrefclever_ref_gender_tl` . . .  
. . . . . [28](#), [870](#), [876](#),  
[877](#), [881](#), [884](#), [889](#), [890](#), [909](#), [915](#),  
[916](#), [2200](#), [2264](#), [5000](#), [5008](#), [5014](#), [5022](#)  
`\l_zrefclever_ref_language_tl` . .  
. . . . . [24](#), [27](#), [55](#), [721](#), [725](#), [822](#),  
[827](#), [837](#), [855](#), [864](#), [874](#), [888](#), [897](#),  
[906](#), [913](#), [2122](#), [2128](#), [2133](#), [2141](#),  
[2159](#), [2165](#), [2170](#), [3437](#), [3455](#), [3826](#),  
[3830](#), [3834](#), [3923](#), [3927](#), [3931](#), [3935](#),  
[3939](#), [3943](#), [3947](#), [3951](#), [3955](#), [3959](#),  
[3963](#), [3967](#), [3971](#), [3975](#), [3979](#), [3983](#),  
[3987](#), [3991](#), [3995](#), [3999](#), [4003](#), [4007](#),  
[5004](#), [5016](#), [5027](#), [5053](#), [5102](#), [5112](#)  
`\l_zrefclever_ref_property_tl` . .  
. . . . . [43](#), [48](#), [1555](#),  
[1564](#), [1571](#), [1573](#), [1757](#), [1781](#), [1825](#),  
[1842](#), [1854](#), [1861](#), [1866](#), [1902](#), [1909](#),  
[1914](#), [1947](#), [1954](#), [1959](#), [3530](#), [3853](#),  
[3908](#), [3912](#), [4737](#), [4822](#), [4876](#), [5185](#)  
`\l_zrefclever_ref_propserity_tl` [3498](#)  
`\l_zrefclever_ref_typeset_font_-`  
`tl` . . . . . [2268](#), [2270](#), [3448](#)  
`\l_zrefclever_refbounds_first_-`  
`pb_seq` . . . . . [3792](#),  
[3980](#), [4056](#), [4106](#), [4178](#), [4229](#), [4310](#)  
`\l_zrefclever_refbounds_first_-`  
`rb_seq` . [3792](#), [3984](#), [4212](#), [4344](#), [4673](#)  
`\l_zrefclever_refbounds_first_-`  
`seq` [3792](#), [3972](#), [4355](#), [4576](#), [4622](#), [4644](#)  
`\l_zrefclever_refbounds_first_-`  
`sg_seq` . [3792](#), [3976](#), [4034](#), [4045](#), [4142](#)

\l\_zrefclever\_refbounds\_last\_  
pe\_seq . . . . . 3792, 4004,  
4053, 4076, 4103, 4155, 4185, 4307  
\l\_zrefclever\_refbounds\_last\_  
re\_seq . . . . .  
. . . 3792, 4008, 4257, 4265, 4332, 4340  
\l\_zrefclever\_refbounds\_last\_  
seq 3792, 4000, 4084, 4119, 4164, 4200  
\l\_zrefclever\_refbounds\_mid\_rb\_  
seq . . . 3792, 3992, 4225, 4239, 4684  
\l\_zrefclever\_refbounds\_mid\_re\_  
seq . . . . . 3792, 3996, 4706, 4714  
\l\_zrefclever\_refbounds\_mid\_seq  
. . . . . 3792, 3988, 4116, 4194,  
4563, 4587, 4599, 4633, 4655, 4663  
\l\_zrefclever\_reffont\_tl . . . . .  
. . . . . 3777, 3948, 4752,  
4764, 4793, 4804, 4842, 4899, 4914  
\l\_zrefclever\_reftype\_override\_  
tl . . . . . 34, 44, 2340, 2343  
\g\_zrefclever\_rf\_opts\_bool\_  
maybe\_type\_specific\_seq . . . . .  
. . . 52, 53, 574, 1444, 2579, 2849, 3377  
\g\_zrefclever\_rf\_opts\_seq\_  
refbounds\_seq . . . . .  
. . . . . 574, 1366, 2542, 2806, 3315  
\g\_zrefclever\_rf\_opts\_tl\_maybe\_  
type\_specific\_seq 574, 1090, 3012  
\g\_zrefclever\_rf\_opts\_tl\_not\_  
type\_specific\_seq . . . . .  
. . . . . 574, 1065, 2628, 2988  
\g\_zrefclever\_rf\_opts\_tl\_  
reference\_seq . . . . . 574, 2494  
\g\_zrefclever\_rf\_opts\_tl\_type\_  
names\_seq . . . . . 574, 1327, 3275  
\g\_zrefclever\_rf\_opts\_tl\_  
typesetup\_seq . . . . . 574, 2640  
\l\_zrefclever\_setup\_language\_tl  
. . . . . 569, 750, 778, 783, 793,  
798, 809, 814, 940, 992, 999, 1013,  
1030, 1036, 1045, 1053, 1077, 1102,  
1111, 1134, 1140, 1148, 1155, 1169,  
1176, 1184, 1192, 1206, 1213, 1221,  
1229, 1243, 1250, 1258, 1266, 1291,  
1298, 1307, 1315, 1345, 1355, 1378,  
1394, 1410, 1427, 1456, 1463, 1472,  
1480, 1494, 1501, 1510, 1518, 2892,  
2933, 2940, 2954, 2971, 2977, 3000,  
3024, 3032, 3055, 3061, 3069, 3076,  
3090, 3097, 3105, 3113, 3127, 3134,  
3142, 3150, 3164, 3171, 3179, 3187,  
3212, 3219, 3228, 3236, 3293, 3303,  
3337, 3362, 3389, 3397, 3410, 3418  
\l\_zrefclever\_setup\_type\_tl 569,  
941, 983, 984, 1016, 1037, 1046,  
1054, 1072, 1097, 1112, 1129, 1149,  
1156, 1164, 1185, 1193, 1201, 1222,  
1230, 1238, 1259, 1267, 1286, 1308,  
1316, 1334, 1346, 1356, 1373, 1411,  
1428, 1451, 1473, 1481, 1489, 1511,  
1519, 2623, 2625, 2652, 2659, 2677,  
2682, 2691, 2697, 2706, 2712, 2721,  
2727, 2736, 2741, 2761, 2768, 2818,  
2835, 2859, 2868, 2877, 2891, 2924,  
2925, 2957, 2978, 2995, 3019, 3033,  
3050, 3070, 3077, 3085, 3106, 3114,  
3122, 3143, 3151, 3159, 3180, 3188,  
3207, 3229, 3237, 3282, 3294, 3304,  
3322, 3363, 3384, 3398, 3405, 3419  
\l\_zrefclever\_sort\_decided\_bool  
. . . . . 3491, 3574, 3588, 3598,  
3602, 3614, 3624, 3639, 3654, 3678  
\\_\_zrefclever\_sort\_default:nn . . . . .  
. . . . . 88, 3532, 3548, 3548  
\\_\_zrefclever\_sort\_default\_  
different\_types:nn . . . . .  
. . . . . 44, 86, 91, 3559, 3691, 3691  
\\_\_zrefclever\_sort\_default\_same\_  
type:nn . . . 86, 88, 3558, 3561, 3561  
\\_\_zrefclever\_sort\_labels: . . . . .  
. . . . . 86, 88, 92, 3446, 3495, 3495  
\\_\_zrefclever\_sort\_page:nn . . . . .  
. . . . . 92, 3531, 3743, 3743  
\l\_zrefclever\_sort\_prior\_a\_int . . . . .  
. . . . . 3492,  
3693, 3699, 3700, 3706, 3716, 3724  
\l\_zrefclever\_sort\_prior\_b\_int . . . . .  
. . . . . 3492,  
3694, 3701, 3702, 3709, 3717, 3725  
\l\_zrefclever\_tlastsep\_tl . . . . .  
. . . . . 3777, 3835, 4454  
\l\_zrefclever\_tlistsep\_tl . . . . .  
. . . . . 3777, 3831, 4425  
\l\_zrefclever\_tmpa\_bool . . . . .  
. . . . . 122, 1770, 1792,  
1814, 1818, 1868, 1869, 1880, 1882,  
1916, 1920, 1922, 1923, 1934, 1936,  
1961, 1965, 1967, 1968, 1986, 1988  
\l\_zrefclever\_tmpa\_int 122, 5608,  
5612, 5619, 5620, 5623, 5625, 5627  
\g\_zrefclever\_tmpa\_seq . . . . .  
. . . . . 122, 1381, 1383,  
1388, 1397, 1402, 1414, 1416, 1421,  
1431, 1436, 3324, 3326, 3331, 3340,  
3345, 3349, 3351, 3356, 3365, 3370  
\l\_zrefclever\_tmpa\_seq . . . . .  
. . . . . 122, 1022, 1026, 1058, 2555,

2557, 2562, 2567, 2572, 2822, 2824,  
 2829, 2837, 2842, 2963, 2967, 2982  
 \l\_zrefclever\_tmpa\_tl . . . . . 122,  
 938, 962, 1858, 1872, 1875, 1876,  
 1906, 1917, 1926, 1929, 1930, 1951,  
 1962, 1971, 1981, 1982, 3456, 3457  
 \l\_zrefclever\_tmpb\_tl . 122, 1820,  
 1827, 1830, 1834, 1863, 1873, 1877,  
 1878, 1879, 1886, 1911, 1919, 1927,  
 1931, 1932, 1933, 1940, 1956, 1964,  
 1972, 1975, 1978, 1983, 1984, 1992  
 \l\_zrefclever\_tpairsep\_tl . . . . .  
 . . . . . 3777, 3827, 4447  
 \l\_zrefclever\_type\_count\_int . . . .  
 . 93, 118, 3755, 3816, 4422, 4424,  
 4437, 4462, 4477, 4948, 4960, 5155  
 \l\_zrefclever\_type\_first\_label\_-  
 tl . . . . . 93,  
 116, 3758, 3811, 4029, 4280, 4291,  
 4295, 4323, 4374, 4391, 4395, 4471,  
 4505, 4815, 4821, 4827, 4831, 4844,  
 4875, 4889, 4893, 4901, 4916, 4929  
 \l\_zrefclever\_type\_first\_label\_-  
 type\_tl . . . 93, 118, 3758, 3812,  
 4031, 4284, 4472, 4507, 4936, 4979,  
 4995, 5003, 5015, 5021, 5038, 5054,  
 5064, 5072, 5081, 5103, 5113, 5125  
 \l\_zrefclever\_type\_first\_-  
 refbounds\_seq . . . . .  
 . . . 3792, 4033, 4044, 4055, 4105,  
 4141, 4177, 4211, 4228, 4309, 4343,  
 4354, 4375, 4575, 4621, 4643, 4672,  
 4839, 4840, 4847, 4849, 4885, 4897,  
 4905, 4908, 4911, 4912, 4919, 4920  
 \l\_zrefclever\_type\_first\_-  
 refbounds\_set\_bool . . . . . 3792,  
 3821, 4035, 4046, 4057, 4108, 4144,  
 4180, 4214, 4231, 4311, 4345,  
 4352, 4481, 4578, 4624, 4646, 4675  
 \l\_zrefclever\_type\_name\_gender\_-  
 seq . . . 3764, 5005, 5007, 5010, 5025  
 \l\_zrefclever\_type\_name\_-  
 missing\_bool . . . . .  
 . . 3764, 4861, 4932, 4939, 5061, 5121  
 \\_zrefclever\_type\_name\_setup: . .  
 . . . . . 20, 22, 116, 4361, 4927, 4927  
 \l\_zrefclever\_type\_name\_tl . . . . .  
 . . . . . 116, 118,  
 3764, 4398, 4404, 4836, 4854, 4869,  
 4871, 4931, 4938, 5042, 5058, 5060,  
 5076, 5085, 5107, 5117, 5119, 5139  
 \l\_zrefclever\_typeset\_compress\_-  
 bool . . . . . 1633, 1636, 4490  
 \l\_zrefclever\_typeset\_labels\_-  
 seq 93, 3752, 3807, 3841, 3843, 3849  
 \l\_zrefclever\_typeset\_last\_bool  
 . . . . . 93, 3752,  
 3838, 3839, 3846, 3871, 4434, 5154  
 \l\_zrefclever\_typeset\_name\_bool  
 . . 1582, 1589, 1594, 1599, 4363, 4379  
 \l\_zrefclever\_typeset\_queue\_-  
 curr\_tl . . . . . 93,  
 95, 113, 118, 3758, 3810, 4049,  
 4072, 4080, 4098, 4111, 4150,  
 4159, 4181, 4189, 4196, 4220, 4234,  
 4251, 4261, 4278, 4303, 4326, 4336,  
 4365, 4372, 4382, 4415, 4431, 4442,  
 4448, 4455, 4469, 4470, 4559, 4582,  
 4594, 4628, 4650, 4659, 4679, 4700,  
 4710, 4953, 4975, 4986, 5149, 5153  
 \l\_zrefclever\_typeset\_queue\_-  
 prev\_tl . 93, 3758, 3809, 4426, 4468  
 \l\_zrefclever\_typeset\_range\_-  
 bool . . . 1767, 1997, 2000, 3445, 4276  
 \l\_zrefclever\_typeset\_ref\_bool .  
 . . 1581, 1588, 1593, 1598, 4363, 4369  
 \\_zrefclever\_typeset\_refs: . . . . .  
 . . . . . 93-95, 3449, 3805, 3805  
 \\_zrefclever\_typeset\_refs\_last\_-  
 of\_type: . . . . .  
 . 100, 113, 115, 118, 4014, 4019, 4019  
 \\_zrefclever\_typeset\_refs\_not\_-  
 last\_of\_type: . . . . .  
 . . 94, 100, 113, 123, 4016, 4484, 4484  
 \l\_zrefclever\_typeset\_sort\_bool  
 . . . . . 1609, 1612, 3444  
 \l\_zrefclever\_typesort\_seq . . . . .  
 . 44, 91, 1618, 1623, 1624, 1630, 3695  
 \l\_zrefclever\_verbose\_testing\_-  
 bool . . . . . 3804, 4430  
 \\_zrefclever\_zcref:nnn . . . . .  
 . . . . . 27, 56, 3430, 3431  
 \\_zrefclever\_zcref:nnnn 84, 87, 3431  
 \l\_zrefclever\_zcref\_labels\_seq .  
 . . . . . 87, 88, 3435,  
 3466, 3471, 3475, 3500, 3503, 3808  
 \l\_zrefclever\_zcref\_note\_tl . . . . .  
 . . . . . 2289, 2292, 3451, 3458  
 \l\_zrefclever\_zcref\_with\_check\_-  
 bool . . . . . 2296, 2313, 3441, 3462  
 \\_zrefclever\_zcsetup:n . . . . .  
 . . . . . 68, 2617, 2618,  
 2618, 2620, 5353, 5377, 5383, 5391,  
 5413, 5432, 5450, 5468, 5502, 5553,  
 5578, 5590, 5600, 5615, 5635, 5658  
 \l\_zrefclever\_zrefcheck\_-  
 available\_bool . . . . .  
 . . 2295, 2308, 2320, 2332, 3440, 3461