

# The zref-clever package

## Code documentation

gusbrs

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

Version v0.4.9 – 2024-11-16

**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 . . . . .	9
	4.3 Data extraction . . . . .	11
	4.4 Option infra . . . . .	12
	4.5 Reference format . . . . .	21
	4.6 Languages . . . . .	25
	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 . . . . .	69
	5.3 \zcLanguageSetup . . . . .	74
<b>6</b>	<b>User interface</b>	<b>84</b>
	6.1 \zcref . . . . .	84
	6.2 \zcpageref . . . . .	86
<b>7</b>	<b>Sorting</b>	<b>86</b>
<b>8</b>	<b>Typesetting</b>	<b>93</b>

<b>9</b>	<b>Compatibility</b>	<b>127</b>
9.1	appendix	127
9.2	appendices	129
9.3	memoir	130
9.4	amsmath	131
9.5	mathtools	133
9.6	breqn	134
9.7	listings	135
9.8	enumitem	135
9.9	subcaption	136
9.10	subfig	137
9.11	beamer	137
<b>10</b>	<b>Language files</b>	<b>138</b>
10.1	Localization guidelines	138
10.2	English	141
10.3	German	144
10.4	French	153
10.5	Portuguese	157
10.6	Spanish	162
10.7	Dutch	166
10.8	Italian	170
10.9	Russian	175
	<b>Index</b>	<b>198</b>

## 1 Initial setup

Start the DocStrip guards.

```
1 \*package)
```

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

```
2 \@@=zrefclever)
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`). 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 (`ltxcmdhooks`), with implications to the hook we add to `\appendix` (by Phe-lype 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 `zref-clever`, 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.

```

3 \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} {2024-11-16} {0.4.9}
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 `\cl@<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 `\cl@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresettters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by

other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\cl@⟨counter⟩` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettors_seq`, and should be handled with care, since there is no possible verification mechanism for this.

Recursively generate a *sequence* of “enclosing counters” and values, for a given `⟨counter⟩` and leave it in the input stream. These functions must be expandable, since they get called from `\zref@newprop` and are the ones responsible for generating the desired information when the label is being set. Note that the order in which we are getting this information is reversed, since we are navigating the counter reset chain bottom-up. But it is very hard to do otherwise here where we need expandable functions, and easy to handle at the reading side.

```

    \__zrefclever_get_enclosing_counters:n {⟨counter⟩}
    \__zrefclever_get_enclosing_counters_value:n {⟨counter⟩}

64 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
65   {
66     \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
67     {
68       { \__zrefclever_counter_reset_by:n {#1} }
69       \__zrefclever_get_enclosing_counters:e
70       { \__zrefclever_counter_reset_by:n {#1} }
71     }
72   }
73 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
74   {
75     \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
76     {
77       { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
78       \__zrefclever_get_enclosing_counters_value:e
79       { \__zrefclever_counter_reset_by:n {#1} }
80     }
81   }

82 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { e }
83 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

```

(End of definition for `\__zrefclever_get_enclosing_counters:n` and `\__zrefclever_get_enclosing_counters_value:n`.)

`\__zrefclever_counter_reset_by:n` Auxiliary function for `\__zrefclever_get_enclosing_counters:n` and `\__zrefclever_get_enclosing_counters_value:n`, and useful on its own standing. It is broken in parts to be able to use the expandable mapping functions. `\__zrefclever_counter_reset_by:n` leaves in the stream the “enclosing counter” which resets `⟨counter⟩`.

```

\__zrefclever_counter_reset_by:n {⟨counter⟩}

```

```

84 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
85   {
86     \bool_if:nTF
87     { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
88     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
89     {
90       \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
91       { \__zrefclever_counter_reset_by_aux:nn {#1} }
92     }
93   }
94 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
95   {
96     \cs_if_exist:cT { c@ #2 }
97     {
98       \tl_if_empty:cF { cl@ #2 }
99       {
100         \tl_map_tokens:cn { cl@ #2 }
101         { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
102       }
103     }
104   }
105 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
106   {
107     \str_if_eq:nnT {#2} {#3}
108     { \tl_map_break:n { \seq_map_break:n {#1} } }
109   }

```

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

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

```

110 \zref@newprop { zc@enclval }
111   {
112     \__zrefclever_get_enclosing_counters_value:e
113     { \l__zrefclever_current_counter_tl }
114   }
115 \zref@addprop \ZREF@mainlist { zc@enclval }

```

The `zc@enclcnt` property is provided for the purpose of easing the debugging of counter reset chains, thus it is not added main property list by default.

```

116 \zref@newprop { zc@enclcnt }
117   { \__zrefclever_get_enclosing_counters:e \l__zrefclever_current_counter_tl }

```

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, 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.

```

118 \int_new:N \g__zrefclever_page_format_int
119 \tl_new:N \g__zrefclever_prev_page_format_tl
120 \AddToHook { shipout / before }
121 {
122   \tl_if_eq:NNF \g__zrefclever_prev_page_format_tl \thepage
123   {
124     \int_gincr:N \g__zrefclever_page_format_int
125     \tl_gset_eq:NN \g__zrefclever_prev_page_format_tl \thepage
126   }
127 }
128 \zref@newprop* { zc@pgfmt } { \int_use:N \g__zrefclever_page_format_int }
129 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

Still some other properties which we don't need to handle at the data provision side, but need to cater for at the retrieval side, are the ones from the `zref-xr` module, which are added to the labels imported from external documents, and needed to construct hyperlinks to them and to distinguish them from the current document ones at sorting and compressing: `urluse`, `url` and `externaldocument`.

## 4 Plumbing

### 4.1 Auxiliary

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

```

130 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
131 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
132 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
133 { \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` Temporary scratch variables.

```

\l__zrefclever_tmpb_tl
\l__zrefclever_tmpa_seq
\g__zrefclever_tmpa_seq
\l__zrefclever_tmpa_bool
\l__zrefclever_tmpa_int
134 \tl_new:N \l__zrefclever_tmpa_tl
135 \tl_new:N \l__zrefclever_tmpb_tl
136 \seq_new:N \l__zrefclever_tmpa_seq
137 \seq_new:N \g__zrefclever_tmpa_seq
138 \bool_new:N \l__zrefclever_tmpa_bool
139 \int_new:N \l__zrefclever_tmpa_int

```



(End of definition for \l\_zrefclever\_tpa\_t1 and others.)

## 4.2 Messages

```
140 \msg_new:nnn { zref-clever } { option-not-type-specific }
141 {
142   Option~'#1'~is~not~type-specific~\msg_line_context:~
143   Set~it~in~'\iow_char:N\zcLanguageSetup'~before~first~'type'~
144   switch~or~as~package~option.
145 }
146 \msg_new:nnn { zref-clever } { option-only-type-specific }
147 {
148   No~type~specified~for~option~'#1'~\msg_line_context:~
149   Set~it~after~'type'~switch.
150 }
151 \msg_new:nnn { zref-clever } { key-requires-value }
152 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:. }
153 \msg_new:nnn { zref-clever } { language-declared }
154 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing~to~do. }
155 \msg_new:nnn { zref-clever } { unknown-language-alias }
156 {
157   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
158   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
159   '\iow_char:N\zcDeclareLanguageAlias'.
160 }
161 \msg_new:nnn { zref-clever } { unknown-language-setup }
162 {
163   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
164   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
165   '\iow_char:N\zcDeclareLanguageAlias'.
166 }
167 \msg_new:nnn { zref-clever } { unknown-language-opt }
168 {
169   Language~'#1'~is~unknown~\msg_line_context:~
170   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
171   '\iow_char:N\zcDeclareLanguageAlias'.
172 }
173 \msg_new:nnn { zref-clever } { unknown-language-decl }
174 {
175   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
176   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
177   '\iow_char:N\zcDeclareLanguageAlias'.
178 }
179 \msg_new:nnn { zref-clever } { language-no-decl-ref }
180 {
181   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
182   Nothing~to~do~with~option~'d=#2'.
183 }
184 \msg_new:nnn { zref-clever } { language-no-gender }
185 {
186   Language~'#1'~has~no~declared~gender~\msg_line_context:~
187   Nothing~to~do~with~option~'#2=#3'.
188 }
189 \msg_new:nnn { zref-clever } { language-no-decl-setup }
```

```

190 {
191   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
192   Nothing~to~do~with~option~'case=#2'.
193 }
194 \msg_new:nnn { zref-clever } { unknown-decl-case }
195 {
196   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:..~
197   Using~default~declension~case.
198 }
199 \msg_new:nnn { zref-clever } { nudge-multitype }
200 {
201   Reference~with~multiple~types~\msg_line_context:..~
202   You~may~wish~to~separate~them~or~review~language~around~it.
203 }
204 \msg_new:nnn { zref-clever } { nudge-comptosing }
205 {
206   Multiple~labels~have~been~compressed~into~singular~type~name~
207   for~type~'#1'~\msg_line_context:..
208 }
209 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
210 {
211   Option~'sg'~signals~that~a~singular~type~name~was~expected~
212   \msg_line_context:..~But~type~'#1'~has~plural~type~name.
213 }
214 \msg_new:nnn { zref-clever } { gender-not-declared }
215 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:.. }
216 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
217 {
218   Gender~mismatch~for~type~'#1'~\msg_line_context:..~
219   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
220 }
221 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
222 {
223   You've~specified~'g=#1'~\msg_line_context:..~
224   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
225 }
226 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
227 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:.. }
228 \msg_new:nnn { zref-clever } { option-document-only }
229 { Option~'#1'~is~only~available~after~\iow_char:N\begin\{document\}. }
230 \msg_new:nnn { zref-clever } { langfile-loaded }
231 { Loaded~'#1'~language~file. }
232 \msg_new:nnn { zref-clever } { zref-property-undefined }
233 {
234   Option~'ref=#1'~requested~\msg_line_context:..~
235   But~the~property~'#1'~is~not~declared,~falling-back~to~'default'.
236 }
237 \msg_new:nnn { zref-clever } { endrange-property-undefined }
238 {
239   Option~'endrange=#1'~requested~\msg_line_context:..~
240   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
241 }
242 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
243 {

```

```

244 Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:~
245 To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
246 '\iow_char:N\zcref'.
247 }
248 \msg_new:nnn { zref-clever } { missing-hyperref }
249 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
250 \msg_new:nnn { zref-clever } { option-preamble-only }
251 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
252 \msg_new:nnn { zref-clever } { unknown-compat-module }
253 {
254   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
255   Nothing~to~do.
256 }
257 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
258 {
259   The~value~of~option~'#1'~must~be~a~comma~separated~list~
260   of~four~items.~We~received~'#2'~items~\msg_line_context:~
261   Option~not~set.
262 }
263 \msg_new:nnn { zref-clever } { missing-zref-check }
264 {
265   Option~'check'~requested~\msg_line_context:~
266   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
267 }
268 \msg_new:nnn { zref-clever } { zref-check-too-old }
269 {
270   Option~'check'~requested~\msg_line_context:~
271   But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
272 }
273 \msg_new:nnn { zref-clever } { missing-type }
274 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
275 \msg_new:nnn { zref-clever } { missing-property }
276 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
277 \msg_new:nnn { zref-clever } { missing-name }
278 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
279 \msg_new:nnn { zref-clever } { single-element-range }
280 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
281 \msg_new:nnn { zref-clever } { compat-package }
282 { Loaded~support~for~'#1'~package. }
283 \msg_new:nnn { zref-clever } { compat-class }
284 { Loaded~support~for~'#1'~documentclass. }
285 \msg_new:nnn { zref-clever } { option-deprecated }
286 {
287   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
288   Use~'#2'~instead.
289 }
290 \msg_new:nnn { zref-clever } { load-time-options }
291 {
292   'zref-clever'~does~not~accept~load-time~options.~
293   To~configure~package~options,~use~'\iow_char:N\zcsetup'.
294 }

```

### 4.3 Data extraction

`\_zrefclever_extract_default:Nnn` Extract property  $\langle prop \rangle$  from  $\langle label \rangle$  and sets variable  $\langle t1 var \rangle$  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  $\langle t1 var \rangle$  with  $\langle default \rangle$ .

```

    \_zrefclever_extract_default:Nnnn {\langle t1 var \rangle}
      {\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

295 \cs_new_protected:Npn \_zrefclever_extract_default:Nnnn #1#2#3#4
296   {
297     \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
298       { \zref@extractdefault {#2} {#3} {#4} }
299   }
300 \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 e expansion, `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. Thus, this is meant to be use in an e expansion context, not in other situations. In case the property is not found, leave  $\langle default \rangle$  in the stream.

```

    \_zrefclever_extract_unexp:nnn{\langle label \rangle}{\langle prop \rangle}{\langle default \rangle}

301 \cs_new:Npn \_zrefclever_extract_unexp:nnn #1#2#3
302   {
303     \exp_args:NNo \exp_args:No
304       \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
305   }
306 \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{\langle label \rangle}{\langle prop \rangle}{\langle default \rangle}

307 \cs_new:Npn \_zrefclever_extract:nnn #1#2#3
308   { \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 `\option`. The data type of the variable must be specified (`tl`, `seq`, `bool`, etc.).

```

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

309 \cs_new:Npn \__zrefclever_opt_varname_general:nn #1#2
310 { 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}

311 \cs_new:Npn \__zrefclever_opt_varname_type:nnn #1#2#3
312 { l__zrefclever_opt_type_ #1 _ #2 _ #3 }
313 \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}

314 \cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
315 {
316   \__zrefclever_language_if_declared:nTF {#1}
317   {
318     g__zrefclever_opt_language_
319     \tl_use:c { \__zrefclever_language_varname:n {#1} }
320     _ #2 _ #3
321   }
322   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
323 }
324 \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>}
325 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
326 {
327   \__zrefclever_language_if_declared:nTF {#1}
328   {
329     g__zrefclever_opt_lang_
330     \tl_use:c { \__zrefclever_language_varname:n {#1} }
331     _default_ #2 _ #3
332   }
333   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
334 }
335 \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:nnnn {<lang>} {<ref type>}
    {<option>} {<data type>}
336 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
337 {
338   \__zrefclever_language_if_declared:nTF {#1}
339   {
340     g__zrefclever_opt_lang_
341     \tl_use:c { \__zrefclever_language_varname:n {#1} }
342     _type_ #2 _ #3 _ #4
343   }
344   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
345 }
346 \cs_generate_variant:Nn
347   \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }

```

(End of definition for `\__zrefclever_opt_varname_lang_type:nnnn`.)

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

```

    \__zrefclever_opt_varname_fallback:nn {<option>} {<data type>}
348 \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
349 { 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>}
```

```
350 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
```

```
351   { \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>}
```

```
352 \cs_new_protected:Npn \_zrefclever_opt_tl_set:Nn #1#2
```

```
353   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_set:Nn #1 {#2}
```

```
357    \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
```

```
358      { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
359      { \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
360   }
```

```
361 \cs_generate_variant:Nn \_zrefclever_opt_tl_set:Nn { cn }
```

```
362 \cs_new_protected:Npn \_zrefclever_opt_tl_clear:N #1
```

```
363   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_clear:N #1
```

```
367    \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
```

```
368      { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
369      { \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
370   }
```

```
371 \cs_generate_variant:Nn \_zrefclever_opt_tl_clear:N { c }
```

```
372 \cs_new_protected:Npn \_zrefclever_opt_tl_gset:Nn #1#2
```

```
373   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_gset:Nn #1 {#2}
```

```
377   }
```

```
378 \cs_generate_variant:Nn \_zrefclever_opt_tl_gset:Nn { cn }
```

```
379 \cs_new_protected:Npn \_zrefclever_opt_tl_gclear:N #1
```

```
380   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_gclear:N #1
```

```
384   }
```

```
385 \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 <option tl>.
```

```
    \_zrefclever_opt_tl_unset:N {<option tl>}
```

```
386 \cs_new_protected:Npn \_zrefclever_opt_tl_unset:N #1
```

```
387   {
```

```
388     \tl_if_exist:NT #1
```

```

389     {
390       \tl_clear:N #1 % ?
391       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
392         { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
393         { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
394     }
395 }
396 \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) {<option tl>} {<true>} {<false>}
397 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
398 {
399   \tl_if_exist:NTF #1
400   {
401     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
402     {
403       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
404       { \prg_return_true: }
405       { \prg_return_false: }
406     }
407     { \prg_return_true: }
408   }
409   { \prg_return_false: }
410 }

```

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

```

\__zrefclever_opt_tl_gset_if_new:Nn \__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\_zrefclever_opt_tl_gclear_if_new:N \__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
411 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
412 {
413   \__zrefclever_opt_tl_if_set:NF #1
414   {
415     \tl_if_exist:NF #1
416     { \tl_new:N #1 }
417     \tl_gset:Nn #1 {#2}
418   }
419 }
420 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
421 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
422 {
423   \__zrefclever_opt_tl_if_set:NF #1
424   {
425     \tl_if_exist:NF #1
426     { \tl_new:N #1 }
427     \tl_gclear:N #1
428   }

```



```

429 }
430 \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>}
431 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
432 {
433   \__zrefclever_opt_tl_if_set:NTF #1
434   {
435     \tl_set_eq:NN #2 #1
436     \prg_return_true:
437   }
438   { \prg_return_false: }
439 }
440 \prg_generate_conditional_variant:Nnn
441 \__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_set_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_gset_clist_split:Nn \__zrefclever_opt_seq_gset_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_set_eq:NN \__zrefclever_opt_seq_set_eq:NN {<option seq>} {<seq var>}
\__zrefclever_opt_seq_gset_eq:NN \__zrefclever_opt_seq_gset_eq:NN {<option seq>} {<seq var>}
442 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
443 { \seq_set_split:Nnn #1 { , } {#2} }
444 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
445 { \seq_gset_split:Nnn #1 { , } {#2} }
446 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
447 {
448   \seq_if_exist:NF #1
449   { \seq_new:N #1 }
450   \seq_set_eq:NN #1 #2
451   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
452   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
453   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
454 }
455 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
456 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
457 {
458   \seq_if_exist:NF #1
459   { \seq_new:N #1 }
460   \seq_gset_eq:NN #1 #2
461 }
462 \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 <option seq>.

```

\__zrefclever_opt_seq_unset:N {<option seq>}

```

```

463 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
464 {
465   \seq_if_exist:NT #1
466   {
467     \seq_clear:N #1 % ?
468     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
469     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
470     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
471   }
472 }
473 \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) {<option seq>} {<true>} {<false>}
474 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
475 {
476   \seq_if_exist:NTF #1
477   {
478     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
479     {
480       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
481       { \prg_return_true: }
482       { \prg_return_false: }
483     }
484     { \prg_return_true: }
485   }
486   { \prg_return_false: }
487 }
488 \prg_generate_conditional_variant:Nnn
489 \__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>}
490 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
491 {
492   \__zrefclever_opt_seq_if_set:NTF #1
493   {
494     \seq_set_eq:NN #2 #1
495     \prg_return_true:
496   }
497   { \prg_return_false: }
498 }
499 \prg_generate_conditional_variant:Nnn
500 \__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>}

```

```

501 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
502 {
503   \bool_if_exist:NT #1
504   {
505     % \bool_set_false:N #1 % ?
506     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
507     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
508     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
509   }
510 }
511 \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>}
512 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
513 {
514   \bool_if_exist:NTF #1
515   {
516     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
517     {
518       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
519       { \prg_return_true: }
520       { \prg_return_false: }
521     }
522     { \prg_return_true: }
523   }
524   { \prg_return_false: }
525 }
526 \prg_generate_conditional_variant:Nnn
527 \__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 {<option bool>}
\__zrefclever_opt_bool_set_false:N {<option bool>}
\__zrefclever_opt_bool_gset_true:N {<option bool>}
\__zrefclever_opt_bool_gset_false:N {<option bool>}
528 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
529 {
530   \bool_if_exist:NF #1
531   { \bool_new:N #1 }
532   \bool_set_true:N #1
533   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
534   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
535   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
536 }
537 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
538 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
539 {
540   \bool_if_exist:NF #1
541   { \bool_new:N #1 }

```

```

542     \bool_set_false:N #1
543     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
544     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
545     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
546   }
547 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
548 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
549   {
550     \bool_if_exist:NF #1
551     { \bool_new:N #1 }
552     \bool_gset_true:N #1
553   }
554 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
555 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
556   {
557     \bool_if_exist:NF #1
558     { \bool_new:N #1 }
559     \bool_gset_false:N #1
560   }
561 \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:NNTF      \__zrefclever_opt_bool_get:NN(TF) {(option bool to get)} {(bool var to set)}
                                     {(true)} {(false)}

```

```

562 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
563   {
564     \__zrefclever_opt_bool_if_set:NTF #1
565     {
566       \bool_set_eq:NN #2 #1
567       \prg_return_true:
568     }
569     { \prg_return_false: }
570   }
571 \prg_generate_conditional_variant:Nnn
572   \__zrefclever_opt_bool_get:NN { cN } { F }

```

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

```

\__zrefclever_opt_bool_if:NTF      \__zrefclever_opt_bool_if:N(TF) {(option bool)} {(true)} {(false)}
573 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
574   {
575     \__zrefclever_opt_bool_if_set:NTF #1
576     { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
577     { \prg_return_false: }
578   }
579 \prg_generate_conditional_variant:Nnn
580   \__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-
  \l_zrefclever_setup_language_tl specific and language-specific options handling, notably in \_zrefclever_provide_
  \l_zrefclever_lang_decl_case_tl langfile:n, \zcRefTypeSetup, and \zcLanguageSetup, but also for language specific
\l_zrefclever_lang_declension_seq options retrieval.
  \l_zrefclever_lang_gender_seq
581 \tl_new:N \l__zrefclever_setup_type_tl
582 \tl_new:N \l__zrefclever_setup_language_tl
583 \tl_new:N \l__zrefclever_lang_decl_case_tl
584 \seq_new:N \l__zrefclever_lang_declension_seq
585 \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
efclever_rf_opts_tl_maybe_type_specific_seq
\g_zrefclever_rf_opts_seq_refbounds_seq
clever_rf_opts_bool_maybe_type_specific_seq
\g_zrefclever_rf_opts_tl_type_names_seq
\g_zrefclever_rf_opts_tl_typesetup_seq
\g_zrefclever_rf_opts_tl_reference_seq

```

Lists of reference format options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent. These variables are *constants*, but I don’t seem to be able to find a way to concatenate two constants into a third one without triggering L<sup>A</sup>T<sub>E</sub>X3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```

586 \seq_new:N \g_zrefclever_rf_opts_tl_not_type_specific_seq
587 \seq_gset_from_clist:Nn
588   \g_zrefclever_rf_opts_tl_not_type_specific_seq
589   {
590     tpairsep ,
591     tlistsep ,
592     tlastsep ,
593     notesep ,
594   }
595 \seq_new:N \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
596 \seq_gset_from_clist:Nn
597   \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
598   {
599     namesep ,
600     pairsep ,
601     listsep ,
602     lastsep ,
603     rangesep ,
604     namefont ,
605     reffont ,
606   }
607 \seq_new:N \g_zrefclever_rf_opts_seq_refbounds_seq
608 \seq_gset_from_clist:Nn
609   \g_zrefclever_rf_opts_seq_refbounds_seq
610   {
611     refbounds-first ,
612     refbounds-first-sg ,
613     refbounds-first-pb ,
614     refbounds-first-rb ,
615     refbounds-mid ,
616     refbounds-mid-rb ,
617     refbounds-mid-re ,
618     refbounds-last ,
619     refbounds-last-pe ,
620     refbounds-last-re ,
621   }
622 \seq_new:N \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
623 \seq_gset_from_clist:Nn
624   \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
625   {
626     cap ,
627     abbrev ,
628     rangetopair ,
629   }

```

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:.

```
630 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
631 \seq_gset_from_clist:Nn
632 \g__zrefclever_rf_opts_tl_type_names_seq
633 {
634   Name-sg ,
635   name-sg ,
636   Name-pl ,
637   name-pl ,
638   Name-sg-ab ,
639   name-sg-ab ,
640   Name-pl-ab ,
641   name-pl-ab ,
642 }
```

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

```
643 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
644 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
645 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
646 \g__zrefclever_rf_opts_tl_type_names_seq
647 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
648 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
649 \g__zrefclever_rf_opts_tl_not_type_specific_seq
650 \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.

```
651 \clist_map_inline:nn
652 {
653   reference ,
654   typesetup ,
655   langsetup ,
656   langfile ,
657 }
658 {
659   \keys_define:nn { zref-clever/ #1 }
660   {
661     +refbounds-first .meta:n =
662     {
663       refbounds-first = {##1} ,
664       refbounds-first-sg = {##1} ,
665       refbounds-first-pb = {##1} ,
666       refbounds-first-rb = {##1} ,
667     } ,
668     +refbounds-mid .meta:n =
669     {
670       refbounds-mid = {##1} ,
671       refbounds-mid-rb = {##1} ,
672       refbounds-mid-re = {##1} ,
673     } ,
674     +refbounds-last .meta:n =
675     {
676       refbounds-last = {##1} ,
```

```

677         refbounds-last-pe = {##1} ,
678         refbounds-last-re = {##1} ,
679     } ,
680 +refbounds-rb .meta:n =
681     {
682         refbounds-first-rb = {##1} ,
683         refbounds-mid-rb = {##1} ,
684     } ,
685 +refbounds-re .meta:n =
686     {
687         refbounds-mid-re = {##1} ,
688         refbounds-last-re = {##1} ,
689     } ,
690 +refbounds .meta:n =
691     {
692         +refbounds-first = {##1} ,
693         +refbounds-mid = {##1} ,
694         +refbounds-last = {##1} ,
695     } ,
696     refbounds .meta:n = { +refbounds = {##1} } ,
697 }
698 }
699 \clist_map_inline:nn
700 {
701     reference ,
702     typesetup ,
703 }
704 {
705     \keys_define:nn { zref-clever/ #1 }
706     {
707         +refbounds-first .default:o = \c_novalue_tl ,
708         +refbounds-mid .default:o = \c_novalue_tl ,
709         +refbounds-last .default:o = \c_novalue_tl ,
710         +refbounds-rb .default:o = \c_novalue_tl ,
711         +refbounds-re .default:o = \c_novalue_tl ,
712         +refbounds .default:o = \c_novalue_tl ,
713         refbounds .default:o = \c_novalue_tl ,
714     }
715 }
716 \clist_map_inline:nn
717 {
718     langsetup ,
719     langfile ,
720 }
721 {
722     \keys_define:nn { zref-clever/ #1 }
723     {
724         +refbounds-first .value_required:n = true ,
725         +refbounds-mid .value_required:n = true ,
726         +refbounds-last .value_required:n = true ,
727         +refbounds-rb .value_required:n = true ,
728         +refbounds-re .value_required:n = true ,
729         +refbounds .value_required:n = true ,
730         refbounds .value_required:n = true ,

```



```

731     }
732 }

```

## 4.6 Languages

`\l_zrefclever_current_language_tl` is an internal alias for babel's `\language` or polyglossia's `\mainbabelname` and, if none of them is loaded, we set it to `english`. `\l_zrefclever_main_language_tl` is an internal alias for babel's `\bbl@main@language` or for polyglossia's `\mainbabelname`, as the case may be. Note that for polyglossia we get babel's language names, so that we only need to handle those internally. `\l_zrefclever_ref_language_tl` is the internal variable which stores the language in which the reference is to be made.

```

733 \tl_new:N \l_zrefclever_ref_language_tl
734 \tl_new:N \l_zrefclever_current_language_tl
735 \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`.

```

736 \tl_new:N \l_zrefclever_ref_language_tl
737 \tl_set:Nn \l_zrefclever_ref_language_tl { \l_zrefclever_ref_language_tl }

```

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

`\_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>}

```

```

738 \cs_new:Npn \_zrefclever_language_varname:n #1
739 { 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`.

```

740 \cs_set_eq:NN \zrefclever_language_varname:n
741 \_zrefclever_language_varname:n

```

*(End of definition for `\zrefclever_language_varname:n`.)*

`\_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>}

```

```

742 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
743 {
744   \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
745     { \prg_return_true: }
746     { \prg_return_false: }
747 }
748 \prg_generate_conditional_variant:Nnn
749 \_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:n` for use in `zref-vario`.

```
750 \prg_set_eq_conditional:NNn \zrefclever_language_if_declared:n
751 \__zrefclever_language_if_declared:n { TF }
```

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

`\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>}

752 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
753 {
754   \group_begin:
755     \tl_if_empty:nF {#2}
756     {
757       \__zrefclever_language_if_declared:nTF {#2}
758       { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
759       {
760         \tl_new:c { \__zrefclever_language_varname:n {#2} }
761         \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
762         \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
763         \keys_set:nn { zref-clever/declarelang } {#1}
764       }
765     }
766   \group_end:
767 }
768 \@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>}

769 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
770 {
771   \tl_if_empty:nF {#1}
772   {
```

```

773     \_zrefclever_language_if_declared:nTF {#2}
774     {
775         \tl_new:c { \_zrefclever_language_varname:n {#1} }
776         \tl_gset:ce { \_zrefclever_language_varname:n {#1} }
777             { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
778     }
779     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
780 }
781 }
782 \@onlypreamble \zcDeclareLanguageAlias

```

(End of definition for \zcDeclareLanguageAlias.)

```

783 \keys_define:nn { zref-clever/declarelang }
784 {
785     declension .code:n =
786     {
787         \seq_new:c
788         {
789             \_zrefclever_opt_varname_language:enn
790             { \l__zrefclever_setup_language_tl } { declension } { seq }
791         }
792         \seq_gset_from_clist:cn
793         {
794             \_zrefclever_opt_varname_language:enn
795             { \l__zrefclever_setup_language_tl } { declension } { seq }
796         }
797         {#1}
798     } ,
799     declension .value_required:n = true ,
800     gender .code:n =
801     {
802         \seq_new:c
803         {
804             \_zrefclever_opt_varname_language:enn
805             { \l__zrefclever_setup_language_tl } { gender } { seq }
806         }
807         \seq_gset_from_clist:cn
808         {
809             \_zrefclever_opt_varname_language:enn
810             { \l__zrefclever_setup_language_tl } { gender } { seq }
811         }
812         {#1}
813     } ,
814     gender .value_required:n = true ,
815     allcaps .choices:nn =
816     { true , false }
817     {
818         \bool_new:c
819         {
820             \_zrefclever_opt_varname_language:enn
821             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
822         }
823         \use:c { bool_gset_ \l_keys_choice_tl :c }
824         {

```

```

825         \_zrefclever_opt_varname_language:enn
826         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
827     }
828 },
829 allcaps .default:n = true ,
830 }

```

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

```

831 \cs_new_protected:Npn \_zrefclever_process_language_settings:
832 {
833     \_zrefclever_language_if_declared:eTF
834     { \l__zrefclever_ref_language_tl }
835     {

```

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.

```

836     \_zrefclever_opt_seq_get:cNF
837     {
838         \_zrefclever_opt_varname_language:enn
839         { \l__zrefclever_ref_language_tl } { declension } { seq }
840     }
841     \l__zrefclever_lang_declension_seq
842     { \seq_clear:N \l__zrefclever_lang_declension_seq }
843     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
844     {
845         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
846         {
847             \msg_warning:nnee { zref-clever }
848             { language-no-decl-ref }
849             { \l__zrefclever_ref_language_tl }
850             { \l__zrefclever_ref_decl_case_tl }
851             \tl_clear:N \l__zrefclever_ref_decl_case_tl
852         }
853     }
854     {
855         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
856         {
857             \seq_get_left:NN \l__zrefclever_lang_declension_seq
858             \l__zrefclever_ref_decl_case_tl
859         }
860         {
861             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

862         \l__zrefclever_ref_decl_case_tl
863         {
864             \msg_warning:nnee { zref-clever }
865             { unknown-decl-case }
866             { \l__zrefclever_ref_decl_case_tl }
867             { \l__zrefclever_ref_language_tl }
868             \seq_get_left:NN \l__zrefclever_lang_declension_seq
869             \l__zrefclever_ref_decl_case_tl
870         }
871     }
872 }

```

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.

```

873     \__zrefclever_opt_seq_get:cNF
874     {
875         \__zrefclever_opt_varname_language:enn
876         { \l__zrefclever_ref_language_tl } { gender } { seq }
877     }
878     \l__zrefclever_lang_gender_seq
879     { \seq_clear:N \l__zrefclever_lang_gender_seq }
880     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
881     {
882         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
883         {
884             \msg_warning:nneee { zref-clever }
885             { language-no-gender }
886             { \l__zrefclever_ref_language_tl }
887             { g }
888             { \l__zrefclever_ref_gender_tl }
889             \tl_clear:N \l__zrefclever_ref_gender_tl
890         }
891     }
892     {
893         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
894         {
895             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
896             \l__zrefclever_ref_gender_tl
897             {
898                 \msg_warning:nnee { zref-clever }
899                 { gender-not-declared }
900                 { \l__zrefclever_ref_language_tl }
901                 { \l__zrefclever_ref_gender_tl }
902                 \tl_clear:N \l__zrefclever_ref_gender_tl
903             }
904         }
905     }

```

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

```

906     \__zrefclever_opt_bool_if:cT
907     {
908         \__zrefclever_opt_varname_language:enn
909         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

910     }
911     { \keys_set:nn { zref-clever/reference } { cap = true } }
912   }
913   {

```

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

```

914     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
915     {
916       \msg_warning:nnee { zref-clever } { unknown-language-decl }
917       { \l__zrefclever_ref_decl_case_tl }
918       { \l__zrefclever_ref_language_tl }
919       \tl_clear:N \l__zrefclever_ref_decl_case_tl
920     }
921     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
922     {
923       \msg_warning:nnee { zref-clever }
924       { language-no-gender }
925       { \l__zrefclever_ref_language_tl }
926       { g }
927       { \l__zrefclever_ref_gender_tl }
928       \tl_clear:N \l__zrefclever_ref_gender_tl
929     }
930   }
931 }

```

(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 `.lbf` 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.

```
932 \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>}
```

```
933 \cs_new_protected:Npn \__zrefclever__provide_langfile:n #1
934 {
935   \group_begin:
936   \@bsphack
937   \__zrefclever_language_if_declared:nT {#1}
938   {
939     \seq_if_in:NcF
940     \g__zrefclever_loaded_langfiles_seq
941     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
942     {
943       \exp_args:Ne \file_get:nnNTF
944       {
945         zref-clever-
946         \tl_use:c { \__zrefclever_language_varname:n {#1} }
947         .lang
948       }
949       { \ExplSyntaxOn }
950       \l__zrefclever_tmpa_tl
951       {
952         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
953         \tl_clear:N \l__zrefclever_setup_type_tl
954         \__zrefclever_opt_seq_get:cNF
955         {
956           \__zrefclever_opt_varname_language:nmn
957           {#1} { declension } { seq }
958         }
959         \l__zrefclever_lang_declension_seq
960         { \seq_clear:N \l__zrefclever_lang_declension_seq }
961         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
962         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
```

```

963         {
964             \seq_get_left:NN \l__zrefclever_lang_declension_seq
965             \l__zrefclever_lang_decl_case_tl
966         }
967     \__zrefclever_opt_seq_get:cNF
968     {
969         \__zrefclever_opt_varname_language:nnn
970         {#1} { gender } { seq }
971     }
972     \l__zrefclever_lang_gender_seq
973     { \seq_clear:N \l__zrefclever_lang_gender_seq }
974     \keys_set:nV { zref-clever/langfile } \l__zrefclever_tmpa_tl
975     \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
976     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
977     \msg_info:nne { zref-clever } { langfile-loaded }
978     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
979 }
980 {

```

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.

```

981         \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
982         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
983     }
984 }
985 }
986 \@esphack
987 \group_end:
988 }
989 \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.

```

990 \keys_define:nn { zref-clever/langfile }
991 {
992     type .code:n =
993     {
994         \tl_if_empty:nTF {#1}
995         { \tl_clear:N \l__zrefclever_setup_type_tl }
996         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
997     } ,
998     case .code:n =
999     {
1000         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1001         {
1002             \msg_info:nnee { zref-clever } { language-no-decl-setup }
1003             { \l__zrefclever_setup_language_tl } {#1}
1004         }
1005     }

```



```

1006         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
1007         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
1008         {
1009             \msg_info:nnee { zref-clever } { unknown-decl-case }
1010             {#1} { \l__zrefclever_setup_language_tl }
1011             \seq_get_left:NN \l__zrefclever_lang_declension_seq
1012             \l__zrefclever_lang_decl_case_tl
1013         }
1014     }
1015 } ,
1016 case .value_required:n = true ,
1017 gender .value_required:n = true ,
1018 gender .code:n =
1019 {
1020     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1021     {
1022         \msg_info:nneee { zref-clever } { language-no-gender }
1023         { \l__zrefclever_setup_language_tl } { gender } {#1}
1024     }
1025     {
1026         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1027         {
1028             \msg_info:nnn { zref-clever }
1029             { option-only-type-specific } { gender }
1030         }
1031         {
1032             \seq_clear:N \l__zrefclever_tmpa_seq
1033             \clist_map_inline:nn {#1}
1034             {
1035                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1036                 { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
1037                 {
1038                     \msg_info:nnee { zref-clever }
1039                     { gender-not-declared }
1040                     { \l__zrefclever_setup_language_tl } {##1}
1041                 }
1042             }
1043             \__zrefclever_opt_seq_if_set:cF
1044             {
1045                 \__zrefclever_opt_varname_lang_type:eenn
1046                 { \l__zrefclever_setup_language_tl }
1047                 { \l__zrefclever_setup_type_tl }
1048                 { gender }
1049                 { seq }
1050             }
1051             {
1052                 \seq_new:c
1053                 {
1054                     \__zrefclever_opt_varname_lang_type:eenn
1055                     { \l__zrefclever_setup_language_tl }
1056                     { \l__zrefclever_setup_type_tl }
1057                     { gender }
1058                     { seq }
1059                 }

```

```

1060         \seq_gset_eq:cn
1061         {
1062             \__zrefclever_opt_varname_lang_type:enn
1063             { \l__zrefclever_setup_language_tl }
1064             { \l__zrefclever_setup_type_tl }
1065             { gender }
1066             { seq }
1067         }
1068         \l__zrefclever_tmpa_seq
1069     }
1070 }
1071 }
1072 },
1073 }
1074 \seq_map_inline:Nn
1075 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1076 {
1077     \keys_define:nn { zref-clever/langfile }
1078     {
1079         #1 .value_required:n = true ,
1080         #1 .code:n =
1081         {
1082             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1083             {
1084                 \__zrefclever_opt_tl_gset_if_new:cn
1085                 {
1086                     \__zrefclever_opt_varname_lang_default:enn
1087                     { \l__zrefclever_setup_language_tl }
1088                     {#1} { tl }
1089                 }
1090                 {##1}
1091             }
1092             {
1093                 \msg_info:nnn { zref-clever }
1094                 { option-not-type-specific } {#1}
1095             }
1096         },
1097     }
1098 }
1099 \seq_map_inline:Nn
1100 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1101 {
1102     \keys_define:nn { zref-clever/langfile }
1103     {
1104         #1 .value_required:n = true ,
1105         #1 .code:n =
1106         {
1107             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1108             {
1109                 \__zrefclever_opt_tl_gset_if_new:cn
1110                 {
1111                     \__zrefclever_opt_varname_lang_default:enn
1112                     { \l__zrefclever_setup_language_tl }
1113                     {#1} { tl }

```

```

1114     }
1115     {##1}
1116   }
1117   {
1118     \__zrefclever_opt_tl_gset_if_new:cn
1119     {
1120       \__zrefclever_opt_varname_lang_type:eenn
1121       { \l__zrefclever_setup_language_tl }
1122       { \l__zrefclever_setup_type_tl }
1123       {#1} { tl }
1124     }
1125     {##1}
1126   }
1127 },
1128 }
1129 }
1130 \keys_define:nn { zref-clever/langfile }
1131 {
1132   endrange .value_required:n = true ,
1133   endrange .code:n =
1134   {
1135     \str_case:nnF {#1}
1136     {
1137       { ref }
1138       {
1139         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1140         {
1141           \__zrefclever_opt_tl_gclear_if_new:c
1142           {
1143             \__zrefclever_opt_varname_lang_default:enn
1144             { \l__zrefclever_setup_language_tl }
1145             { endrangefunc } { tl }
1146           }
1147           \__zrefclever_opt_tl_gclear_if_new:c
1148           {
1149             \__zrefclever_opt_varname_lang_default:enn
1150             { \l__zrefclever_setup_language_tl }
1151             { endrangeprop } { tl }
1152           }
1153         }
1154       }
1155     }
1156     \__zrefclever_opt_tl_gclear_if_new:c
1157     {
1158       \__zrefclever_opt_varname_lang_type:eenn
1159       { \l__zrefclever_setup_language_tl }
1160       { \l__zrefclever_setup_type_tl }
1161       { endrangefunc } { tl }
1162     }
1163     \__zrefclever_opt_tl_gclear_if_new:c
1164     {
1165       \__zrefclever_opt_varname_lang_type:eenn
1166       { \l__zrefclever_setup_language_tl }
1167       { \l__zrefclever_setup_type_tl }
1168       { endrangeprop } { tl }

```

```

1168     }
1169   }
1170 }
1171 { stripprefix }
1172 {
1173   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1174   {
1175     \__zrefclever_opt_tl_gset_if_new:cn
1176     {
1177       \__zrefclever_opt_varname_lang_default:enn
1178       { \l__zrefclever_setup_language_tl }
1179       { endrangefunc } { tl }
1180     }
1181     { __zrefclever_get_endrange_striprefix }
1182     \__zrefclever_opt_tl_gclear_if_new:c
1183     {
1184       \__zrefclever_opt_varname_lang_default:enn
1185       { \l__zrefclever_setup_language_tl }
1186       { endrangeprop } { tl }
1187     }
1188   }
1189   {
1190     \__zrefclever_opt_tl_gset_if_new:cn
1191     {
1192       \__zrefclever_opt_varname_lang_type:eenn
1193       { \l__zrefclever_setup_language_tl }
1194       { \l__zrefclever_setup_type_tl }
1195       { endrangefunc } { tl }
1196     }
1197     { __zrefclever_get_endrange_striprefix }
1198     \__zrefclever_opt_tl_gclear_if_new:c
1199     {
1200       \__zrefclever_opt_varname_lang_type:eenn
1201       { \l__zrefclever_setup_language_tl }
1202       { \l__zrefclever_setup_type_tl }
1203       { endrangeprop } { tl }
1204     }
1205   }
1206 }
1207 { pagecomp }
1208 {
1209   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1210   {
1211     \__zrefclever_opt_tl_gset_if_new:cn
1212     {
1213       \__zrefclever_opt_varname_lang_default:enn
1214       { \l__zrefclever_setup_language_tl }
1215       { endrangefunc } { tl }
1216     }
1217     { __zrefclever_get_endrange_pagecomp }
1218     \__zrefclever_opt_tl_gclear_if_new:c
1219     {
1220       \__zrefclever_opt_varname_lang_default:enn
1221       { \l__zrefclever_setup_language_tl }

```

```

1222         { endrangeprop } { t1 }
1223     }
1224 }
1225 {
1226     \__zrefclever_opt_tl_gset_if_new:cn
1227     {
1228         \__zrefclever_opt_varname_lang_type:eenn
1229         { \l__zrefclever_setup_language_tl }
1230         { \l__zrefclever_setup_type_tl }
1231         { endrangefunc } { t1 }
1232     }
1233     { __zrefclever_get_endrange_pagecomp }
1234     \__zrefclever_opt_tl_gclear_if_new:c
1235     {
1236         \__zrefclever_opt_varname_lang_type:eenn
1237         { \l__zrefclever_setup_language_tl }
1238         { \l__zrefclever_setup_type_tl }
1239         { endrangeprop } { t1 }
1240     }
1241 }
1242 }
1243 { pagecomp2 }
1244 {
1245     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1246     {
1247         \__zrefclever_opt_tl_gset_if_new:cn
1248         {
1249             \__zrefclever_opt_varname_lang_default:enn
1250             { \l__zrefclever_setup_language_tl }
1251             { endrangefunc } { t1 }
1252         }
1253         { __zrefclever_get_endrange_pagecomptwo }
1254         \__zrefclever_opt_tl_gclear_if_new:c
1255         {
1256             \__zrefclever_opt_varname_lang_default:enn
1257             { \l__zrefclever_setup_language_tl }
1258             { endrangeprop } { t1 }
1259         }
1260     }
1261 }
1262     \__zrefclever_opt_tl_gset_if_new:cn
1263     {
1264         \__zrefclever_opt_varname_lang_type:eenn
1265         { \l__zrefclever_setup_language_tl }
1266         { \l__zrefclever_setup_type_tl }
1267         { endrangefunc } { t1 }
1268     }
1269     { __zrefclever_get_endrange_pagecomptwo }
1270     \__zrefclever_opt_tl_gclear_if_new:c
1271     {
1272         \__zrefclever_opt_varname_lang_type:eenn
1273         { \l__zrefclever_setup_language_tl }
1274         { \l__zrefclever_setup_type_tl }
1275         { endrangeprop } { t1 }

```

```

1276     }
1277   }
1278 }
1279 }
1280 {
1281   \tl_if_empty:nTF {#1}
1282   {
1283     \msg_info:nnn { zref-clever }
1284     { endrange-property-undefined } {#1}
1285   }
1286   {
1287     \zref@ifpropundefined {#1}
1288     {
1289       \msg_info:nnn { zref-clever }
1290       { endrange-property-undefined } {#1}
1291     }
1292     {
1293       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1294       {
1295         \__zrefclever_opt_tl_gset_if_new:cn
1296         {
1297           \__zrefclever_opt_varname_lang_default:enn
1298           { \l__zrefclever_setup_language_tl }
1299           { endrangefunc } { tl }
1300         }
1301         { __zrefclever_get_endrange_property }
1302         \__zrefclever_opt_tl_gset_if_new:cn
1303         {
1304           \__zrefclever_opt_varname_lang_default:enn
1305           { \l__zrefclever_setup_language_tl }
1306           { endrangeprop } { tl }
1307         }
1308         {#1}
1309       }
1310       {
1311         \__zrefclever_opt_tl_gset_if_new:cn
1312         {
1313           \__zrefclever_opt_varname_lang_type:eenn
1314           { \l__zrefclever_setup_language_tl }
1315           { \l__zrefclever_setup_type_tl }
1316           { endrangefunc } { tl }
1317         }
1318         { __zrefclever_get_endrange_property }
1319         \__zrefclever_opt_tl_gset_if_new:cn
1320         {
1321           \__zrefclever_opt_varname_lang_type:eenn
1322           { \l__zrefclever_setup_language_tl }
1323           { \l__zrefclever_setup_type_tl }
1324           { endrangeprop } { tl }
1325         }
1326         {#1}
1327       }
1328     }
1329   }

```

```

1330     }
1331   } ,
1332 }
1333 \seq_map_inline:Nn
1334 \g__zrefclever_rf_opts_tl_type_names_seq
1335 {
1336   \keys_define:nn { zref-clever/langfile }
1337   {
1338     #1 .value_required:n = true ,
1339     #1 .code:n =
1340     {
1341       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1342       {
1343         \msg_info:nnn { zref-clever }
1344         { option-only-type-specific } {#1}
1345       }
1346       {
1347         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1348         {
1349           \__zrefclever_opt_tl_gset_if_new:cn
1350           {
1351             \__zrefclever_opt_varname_lang_type:eenn
1352             { \l__zrefclever_setup_language_tl }
1353             { \l__zrefclever_setup_type_tl }
1354             {#1} { tl }
1355           }
1356           {##1}
1357         }
1358         {
1359           \__zrefclever_opt_tl_gset_if_new:cn
1360           {
1361             \__zrefclever_opt_varname_lang_type:eeen
1362             { \l__zrefclever_setup_language_tl }
1363             { \l__zrefclever_setup_type_tl }
1364             { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
1365           }
1366           {##1}
1367         }
1368       }
1369     } ,
1370   }
1371 }
1372 \seq_map_inline:Nn
1373 \g__zrefclever_rf_opts_seq_refbounds_seq
1374 {
1375   \keys_define:nn { zref-clever/langfile }
1376   {
1377     #1 .value_required:n = true ,
1378     #1 .code:n =
1379     {
1380       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1381       {
1382         \__zrefclever_opt_seq_if_set:cF
1383         {

```

```

1384     \__zrefclever_opt_varname_lang_default:enn
1385     { \l__zrefclever_setup_language_tl } {#1} { seq }
1386   }
1387   {
1388     \seq_gclear:N \g__zrefclever_tmpa_seq
1389     \__zrefclever_opt_seq_gset_clist_split:Nn
1390     \g__zrefclever_tmpa_seq {##1}
1391     \bool_lazy_or:nnTF
1392     { \tl_if_empty_p:n {##1} }
1393     {
1394       \int_compare_p:nNn
1395       { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1396     }
1397     {
1398       \__zrefclever_opt_seq_gset_eq:cN
1399       {
1400         \__zrefclever_opt_varname_lang_default:enn
1401         { \l__zrefclever_setup_language_tl }
1402         {#1} { seq }
1403       }
1404       \g__zrefclever_tmpa_seq
1405     }
1406     {
1407       \msg_info:nnee { zref-clever }
1408       { refbounds-must-be-four }
1409       {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
1410     }
1411   }
1412 }
1413 {
1414   \__zrefclever_opt_seq_if_set:cF
1415   {
1416     \__zrefclever_opt_varname_lang_type:eenn
1417     { \l__zrefclever_setup_language_tl }
1418     { \l__zrefclever_setup_type_tl } {#1} { seq }
1419   }
1420   {
1421     \seq_gclear:N \g__zrefclever_tmpa_seq
1422     \__zrefclever_opt_seq_gset_clist_split:Nn
1423     \g__zrefclever_tmpa_seq {##1}
1424     \bool_lazy_or:nnTF
1425     { \tl_if_empty_p:n {##1} }
1426     {
1427       \int_compare_p:nNn
1428       { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1429     }
1430     {
1431       \__zrefclever_opt_seq_gset_eq:cN
1432       {
1433         \__zrefclever_opt_varname_lang_type:eenn
1434         { \l__zrefclever_setup_language_tl }
1435         { \l__zrefclever_setup_type_tl }
1436         {#1} { seq }
1437       }

```



```

1438         \g__zrefclever_tmpa_seq
1439     }
1440     {
1441     \msg_info:nnee { zref-clever }
1442     { refbounds-must-be-four }
1443     {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
1444     }
1445     }
1446     } ,
1447 } ,
1448 }
1449 }
1450 \seq_map_inline:Nn
1451 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1452 {
1453     \keys_define:nn { zref-clever/langfile }
1454     {
1455         #1 .choice: ,
1456         #1 / true .code:n =
1457         {
1458             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1459             {
1460                 \__zrefclever_opt_bool_if_set:cF
1461                 {
1462                     \__zrefclever_opt_varname_lang_default:enn
1463                     { \l__zrefclever_setup_language_tl }
1464                     {#1} { bool }
1465                 }
1466                 {
1467                     \__zrefclever_opt_bool_gset_true:c
1468                     {
1469                         \__zrefclever_opt_varname_lang_default:enn
1470                         { \l__zrefclever_setup_language_tl }
1471                         {#1} { bool }
1472                     }
1473                 }
1474             }
1475         }
1476         \__zrefclever_opt_bool_if_set:cF
1477         {
1478             \__zrefclever_opt_varname_lang_type:eenn
1479             { \l__zrefclever_setup_language_tl }
1480             { \l__zrefclever_setup_type_tl }
1481             {#1} { bool }
1482         }
1483         {
1484             \__zrefclever_opt_bool_gset_true:c
1485             {
1486                 \__zrefclever_opt_varname_lang_type:eenn
1487                 { \l__zrefclever_setup_language_tl }
1488                 { \l__zrefclever_setup_type_tl }
1489                 {#1} { bool }
1490             }
1491         }

```

```

1492     }
1493   } ,
1494   #1 / false .code:n =
1495   {
1496     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1497     {
1498       \__zrefclever_opt_bool_if_set:cF
1499       {
1500         \__zrefclever_opt_varname_lang_default:enn
1501         { \l__zrefclever_setup_language_tl }
1502         {#1} { bool }
1503       }
1504       {
1505         \__zrefclever_opt_bool_gset_false:c
1506         {
1507           \__zrefclever_opt_varname_lang_default:enn
1508           { \l__zrefclever_setup_language_tl }
1509           {#1} { bool }
1510         }
1511       }
1512     }
1513   {
1514     \__zrefclever_opt_bool_if_set:cF
1515     {
1516       \__zrefclever_opt_varname_lang_type:enn
1517       { \l__zrefclever_setup_language_tl }
1518       { \l__zrefclever_setup_type_tl }
1519       {#1} { bool }
1520     }
1521     {
1522       \__zrefclever_opt_bool_gset_false:c
1523       {
1524         \__zrefclever_opt_varname_lang_type:enn
1525         { \l__zrefclever_setup_language_tl }
1526         { \l__zrefclever_setup_type_tl }
1527         {#1} { bool }
1528       }
1529     }
1530   }
1531   } ,
1532   #1 .default:n = true ,
1533   no #1 .meta:n = { #1 = false } ,
1534   no #1 .value_forbidden:n = true ,
1535 }
1536 }

```

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.

```

1537 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1538 {

```

```

1539     \tl_const:cn
1540     { \__zrefclever_opt_varname_fallback:nn {#1} { t1 } } {#2}
1541   }
1542 \keyval_parse:nnn
1543 { }
1544 { \__zrefclever_opt_tl_cset_fallback:nn }
1545 {
1546   tpairsep = {,~} ,
1547   tlistsep = {,~} ,
1548   tlastsep = {,~} ,
1549   notesep  = {~} ,
1550   namesep  = {\nobreakspace} ,
1551   pairsep  = {,~} ,
1552   listsep  = {,~} ,
1553   lastsep  = {,~} ,
1554   rangeseq = {\textendash} ,
1555 }

```

## 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 <property list> {<key>} {<value>}

1556 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1557 {
1558   \tl_if_empty:nTF {#3}
1559     { \prop_remove:Nn #1 {#2} }
1560     { \prop_put:Nnn #1 {#2} {#3} }
1561 }

```

(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`.

```

1562 \tl_new:N \l__zrefclever_ref_property_tl
1563 \keys_define:nn { zref-clever/reference }
1564 {
1565   ref .code:n =
1566     {
1567       \tl_if_empty:nTF {#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     {
1574         \zref@ifpropundefined {#1}
1575         {
1576             \msg_warning:nnn { zref-clever }
1577             { zref-property-undefined } {#1}
1578             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1579         }
1580         { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1581     }
1582 },
1583 ref .initial:n = default ,
1584 ref .value_required:n = true ,
1585 page .meta:n = { ref = page },
1586 page .value_forbidden:n = true ,
1587 }

```

### typeset option

```

1588 \bool_new:N \l__zrefclever_typeset_ref_bool
1589 \bool_new:N \l__zrefclever_typeset_name_bool
1590 \keys_define:nn { zref-clever/reference }
1591 {
1592     typeset .choice: ,
1593     typeset / both .code:n =
1594     {
1595         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1596         \bool_set_true:N \l__zrefclever_typeset_name_bool
1597     } ,
1598     typeset / ref .code:n =
1599     {
1600         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1601         \bool_set_false:N \l__zrefclever_typeset_name_bool
1602     } ,
1603     typeset / name .code:n =
1604     {
1605         \bool_set_false:N \l__zrefclever_typeset_ref_bool
1606         \bool_set_true:N \l__zrefclever_typeset_name_bool
1607     } ,
1608     typeset .initial:n = both ,
1609     typeset .value_required:n = true ,
1610     noname .meta:n = { typeset = ref } ,
1611     noname .value_forbidden:n = true ,
1612     noref .meta:n = { typeset = name } ,
1613     noref .value_forbidden:n = true ,
1614 }

```

### sort option

```

1615 \bool_new:N \l__zrefclever_typeset_sort_bool
1616 \keys_define:nn { zref-clever/reference }
1617 {

```

```

1618     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1619     sort .initial:n = true ,
1620     sort .default:n = true ,
1621     nosort .meta:n = { sort = false },
1622     nosort .value_forbidden:n = true ,
1623 }

```

### 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`.

```

1624 \seq_new:N \l__zrefclever_typesort_seq
1625 \keys_define:nn { zref-clever/reference }
1626 {
1627     typesort .code:n =
1628     {
1629         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1630         \seq_reverse:N \l__zrefclever_typesort_seq
1631     } ,
1632     typesort .initial:n =
1633     { part , chapter , section , paragraph },
1634     typesort .value_required:n = true ,
1635     notypesort .code:n =
1636     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1637     notypesort .value_forbidden:n = true ,
1638 }

```

### comp option

```

1639 \bool_new:N \l__zrefclever_typeset_compress_bool
1640 \keys_define:nn { zref-clever/reference }
1641 {
1642     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1643     comp .initial:n = true ,
1644     comp .default:n = true ,
1645     nocomp .meta:n = { comp = false },
1646     nocomp .value_forbidden:n = true ,
1647 }

```

### endrange option

The working of `endrange` option depends on two underlying option values / variables: `endrangefunc` and `endrangeprop`. `endrangefunc` is the more general one, and `endrangeprop` is used when the first is set to `\__zrefclever_get_endrange_property:VvN`, which is the case when the user is setting `endrange` to an arbitrary `zref` property, instead of one of the `\str_case:nn` matches.

`endrangefunc` *must* receive three arguments and, more specifically, its signature *must* be `VvN`. For this reason, `endrangefunc` should be stored without the signature, which is added, and hard-coded, at the calling place. The first argument is `<beg range label>`, the second `<end range label>`, and the last `<t1 var to set>`. Of course, `<t1 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 expand the references before processing them, when such processing is required. At first sight, it makes sense to do so, since we are aiming at “removing common parts” as close as possible to the printed representation of the references (`cleveref` does expand them in `\crefstripprefix`). On the other hand, this brings some new challenges: if a fragile command gets there, we are in trouble; also, if a protected one gets there, though things won’t break as badly, we may “strip” the macro and stay with different arguments, which will then end up in the input stream. I think `biblatex` is a good reference here, and it offers `\NumCheckSetup`, `\NumsCheckSetup`, and `\PagesCheckSetup` aimed at locally redefining some commands which may interfere with the processing. This is a good idea, thus we offer a similar hook for the same purpose: `endrange-setup`.

```

1648 \NewHook { zref-clever/endrange-setup }
1649 \keys_define:nn { zref-clever/reference }
1650 {
1651   endrange .code:n =
1652   {
1653     \str_case:nnF {#1}
1654     {
1655       { ref }
1656       {
1657         \__zrefclever_opt_tl_clear:c
1658         {
1659           \__zrefclever_opt_varname_general:nn
1660           { endrangefunc } { tl }
1661         }
1662         \__zrefclever_opt_tl_clear:c
1663         {
1664           \__zrefclever_opt_varname_general:nn
1665           { endrangeprop } { tl }
1666         }
1667       }
1668     } { stripprefix }
1669     {
1670       \__zrefclever_opt_tl_set:cn
1671       {
1672         \__zrefclever_opt_varname_general:nn
1673         { endrangefunc } { tl }
1674       }
1675     } { __zrefclever_get_endrange_stripprefix }
1676   } \__zrefclever_opt_tl_clear:c
1677   {
1678     \__zrefclever_opt_varname_general:nn
1679     { endrangeprop } { tl }
1680   }

```

```

1681     }
1682     { pagecomp }
1683     {
1684         \__zrefclever_opt_t1_set:cn
1685         {
1686             \__zrefclever_opt_varname_general:nn
1687             { endrangefunc } { t1 }
1688         }
1689         { __zrefclever_get_endrange_pagecomp }
1690         \__zrefclever_opt_t1_clear:c
1691         {
1692             \__zrefclever_opt_varname_general:nn
1693             { endrangeprop } { t1 }
1694         }
1695     }
1696     { pagecomp2 }
1697     {
1698         \__zrefclever_opt_t1_set:cn
1699         {
1700             \__zrefclever_opt_varname_general:nn
1701             { endrangefunc } { t1 }
1702         }
1703         { __zrefclever_get_endrange_pagecomptwo }
1704         \__zrefclever_opt_t1_clear:c
1705         {
1706             \__zrefclever_opt_varname_general:nn
1707             { endrangeprop } { t1 }
1708         }
1709     }
1710     { unset }
1711     {
1712         \__zrefclever_opt_t1_unset:c
1713         {
1714             \__zrefclever_opt_varname_general:nn
1715             { endrangefunc } { t1 }
1716         }
1717         \__zrefclever_opt_t1_unset:c
1718         {
1719             \__zrefclever_opt_varname_general:nn
1720             { endrangeprop } { t1 }
1721         }
1722     }
1723 }
1724 {
1725     \tl_if_empty:nTF {#1}
1726     {
1727         \msg_warning:nnn { zref-clever }
1728         { endrange-property-undefined } {#1}
1729     }
1730     {
1731         \zref@ifpropundefined {#1}
1732         {
1733             \msg_warning:nnn { zref-clever }
1734             { endrange-property-undefined } {#1}

```

```

1735     }
1736     {
1737     \__zrefclever_opt_tl_set:cn
1738     {
1739     \__zrefclever_opt_varname_general:nn
1740     { endrangefunc } { t1 }
1741     }
1742     { __zrefclever_get_endrange_property }
1743     \__zrefclever_opt_tl_set:cn
1744     {
1745     \__zrefclever_opt_varname_general:nn
1746     { endrangeprop } { t1 }
1747     }
1748     {#1}
1749     }
1750     }
1751   }
1752   } ,
1753   endrange .value_required:n = true ,
1754 }
1755 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1756 {
1757   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1758   {
1759     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1760     {
1761       \__zrefclever_extract_default:Nnvn #3
1762       {#2} { \l__zrefclever_ref_property_tl } { }
1763     }
1764     { \tl_set:Nn #3 { zc@missingproperty } }
1765   }
1766   {
1767     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1768     {

```

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.

```

1769     \bool_if:NTF \l__zrefclever_typeset_range_bool
1770     {
1771       \group_begin:
1772       \bool_set_false:N \l__zrefclever_tmpa_bool
1773       \exp_args:Nee \tl_if_eq:nnT
1774       {
1775         \__zrefclever_extract_unexp:nnn
1776         {#1} { externaldocument } { }
1777       }
1778       {
1779         \__zrefclever_extract_unexp:nnn
1780         {#2} { externaldocument } { }
1781       }
1782     }

```



```

1783 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1784 {
1785   \exp_args:Nee \tl_if_eq:nnT
1786   {
1787     \__zrefclever_extract_unexp:nnn
1788     {#1} { zc@pgfmt } { }
1789   }
1790   {
1791     \__zrefclever_extract_unexp:nnn
1792     {#2} { zc@pgfmt } { }
1793   }
1794   { \bool_set_true:N \l__zrefclever_tmpa_bool }
1795 }
1796 {
1797   \exp_args:Nee \tl_if_eq:nnT
1798   {
1799     \__zrefclever_extract_unexp:nnn
1800     {#1} { zc@counter } { }
1801   }
1802   {
1803     \__zrefclever_extract_unexp:nnn
1804     {#2} { zc@counter } { }
1805   }
1806   {
1807     \exp_args:Nee \tl_if_eq:nnT
1808     {
1809       \__zrefclever_extract_unexp:nnn
1810       {#1} { zc@enclval } { }
1811     }
1812     {
1813       \__zrefclever_extract_unexp:nnn
1814       {#2} { zc@enclval } { }
1815     }
1816     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1817   }
1818 }
1819 }
1820 \bool_if:NTF \l__zrefclever_tmpa_bool
1821 {
1822   \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1823   {#2} { l__zrefclever_endrangeprop_tl } { }
1824 }
1825 {
1826   \zref@ifrefcontainsprop
1827   {#2} { \l__zrefclever_ref_property_tl }
1828   {
1829     \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1830     {#2} { l__zrefclever_ref_property_tl } { }
1831   }
1832   { \tl_set:Nn \l__zrefclever_tmpb_tl { zc@missingproperty } }
1833 }
1834 \exp_args:NNNV
1835 \group_end:
1836 \tl_set:Nn #3 \l__zrefclever_tmpb_tl

```

```

1837     }
1838     {
1839         \__zrefclever_extract_default:Nnvn #3
1840         {#2} { l__zrefclever_endrangeprop_tl } { }
1841     }
1842 }
1843 {
1844     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1845     {
1846         \__zrefclever_extract_default:Nnvn #3
1847         {#2} { l__zrefclever_ref_property_tl } { }
1848     }
1849     { \tl_set:Nn #3 { zc@missingproperty } }
1850 }
1851 }
1852 }
1853 \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>.

```

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

```

```

1888         \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1889     }
1890     { \tl_set:Nn #3 { zc@missingproperty } }
1891 }
1892 \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>).

```

1893 \prg_new_protected_conditional:Npnn
1894   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1895 {
1896   \regex_match:nnTF { \A\d+\Z } {#1}
1897   { \prg_return_true: }
1898   { \prg_return_false: }
1899 }
1900 \prg_generate_conditional_variant:Nnn
1901   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End of definition for \\_\_zrefclever\_is\_integer\_rgx:n.)

```

1902 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1903 {
1904   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1905   {
1906     \group_begin:
1907     \UseHook { zref-clever/endrange-setup }
1908     \tl_set:Ne \l__zrefclever_tmpa_tl
1909     {
1910       \__zrefclever_extract:nnn
1911       {#1} { \l__zrefclever_ref_property_tl } { }
1912     }
1913     \tl_set:Ne \l__zrefclever_tmpb_tl
1914     {
1915       \__zrefclever_extract:nnn
1916       {#2} { \l__zrefclever_ref_property_tl } { }
1917     }
1918     \bool_set_false:N \l__zrefclever_tmpa_bool
1919     \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1920     {
1921       \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1922       { \bool_set_true:N \l__zrefclever_tmpa_bool }
1923     }
1924     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1925     \bool_until_do:Nn \l__zrefclever_tmpa_bool
1926     {
1927       \exp_args:Nee \tl_if_eq:nnTF
1928       { \tl_head:V \l__zrefclever_tmpa_tl }
1929       { \tl_head:V \l__zrefclever_tmpb_tl }
1930       {
1931         \tl_set:Ne \l__zrefclever_tmpa_tl
1932         { \tl_tail:V \l__zrefclever_tmpa_tl }
1933         \tl_set:Ne \l__zrefclever_tmpb_tl
1934         { \tl_tail:V \l__zrefclever_tmpb_tl }
1935         \tl_if_empty:NT \l__zrefclever_tmpb_tl
1936         { \bool_set_true:N \l__zrefclever_tmpa_bool }

```

```

1937         }
1938         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1939     }
1940     \exp_args:NNNV
1941     \group_end:
1942     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1943 }
1944 { \tl_set:Nn #3 { zc@missingproperty } }
1945 }
1946 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1947 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1948 {
1949     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1950     {
1951         \group_begin:
1952         \UseHook { zref-clever/endorange-setup }
1953         \tl_set:Ne \l__zrefclever_tmpa_tl
1954         {
1955             \__zrefclever_extract:nnn
1956             {#1} { \l__zrefclever_ref_property_tl } { }
1957         }
1958         \tl_set:Ne \l__zrefclever_tmpb_tl
1959         {
1960             \__zrefclever_extract:nnn
1961             {#2} { \l__zrefclever_ref_property_tl } { }
1962         }
1963         \bool_set_false:N \l__zrefclever_tmpa_bool
1964         \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1965         {
1966             \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1967             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1968         }
1969         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1970     \bool_until_do:Nn \l__zrefclever_tmpa_bool
1971     {
1972         \exp_args:Nee \tl_if_eq:nnTF
1973         { \tl_head:V \l__zrefclever_tmpa_tl }
1974         { \tl_head:V \l__zrefclever_tmpb_tl }
1975         {
1976             \bool_lazy_or:nnTF
1977             { \int_compare_p:nNn { \l__zrefclever_tmpb_tl } > { 99 } }
1978             {
1979                 \int_compare_p:nNn
1980                 { \tl_head:V \l__zrefclever_tmpb_tl } = { 0 }
1981             }
1982             {
1983                 \tl_set:Ne \l__zrefclever_tmpa_tl
1984                 { \tl_tail:V \l__zrefclever_tmpa_tl }
1985                 \tl_set:Ne \l__zrefclever_tmpb_tl
1986                 { \tl_tail:V \l__zrefclever_tmpb_tl }
1987             }
1988             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1989         }
1990     }
    { \bool_set_true:N \l__zrefclever_tmpa_bool }

```

```

1991     }
1992     \exp_args:NNNV
1993     \group_end:
1994     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1995   }
1996   { \tl_set:Nn #3 { zc@missingproperty } }
1997 }
1998 \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`.

```

1999 \bool_new:N \l__zrefclever_typeset_range_bool
2000 \keys_define:nn { zref-clever/reference }
2001 {
2002   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
2003   range .initial:n = false ,
2004   range .default:n = true ,
2005 }

```

### 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`.

```

2006 \bool_new:N \l__zrefclever_capfirst_bool
2007 \keys_define:nn { zref-clever/reference }
2008 {
2009   capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
2010   capfirst .initial:n = false ,
2011   capfirst .default:n = true ,
2012 }

```

### 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`.

```

2013 \bool_new:N \l__zrefclever_noabbrev_first_bool
2014 \keys_define:nn { zref-clever/reference }
2015 {
2016   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
2017   noabbrevfirst .initial:n = false ,
2018   noabbrevfirst .default:n = true ,
2019 }

```

### S option

```

2020 \keys_define:nn { zref-clever/reference }
2021 {
2022   S .meta:n =
2023     { capfirst = {#1} , noabbrevfirst = {#1} },
2024   S .default:n = true ,

```

```
2025 }
```

### hyperref option

```
2026 \bool_new:N \l__zrefclever_hyperlink_bool
2027 \bool_new:N \l__zrefclever_hyperref_warn_bool
2028 \keys_define:nn { zref-clever/reference }
2029 {
2030   hyperref .choice: ,
2031   hyperref / auto .code:n =
2032     {
2033       \bool_set_true:N \l__zrefclever_hyperlink_bool
2034       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2035     } ,
2036   hyperref / true .code:n =
2037     {
2038       \bool_set_true:N \l__zrefclever_hyperlink_bool
2039       \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2040     } ,
2041   hyperref / false .code:n =
2042     {
2043       \bool_set_false:N \l__zrefclever_hyperlink_bool
2044       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2045     } ,
2046   hyperref .initial:n = auto ,
2047   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.

```
2048   nohyperref .meta:n = { hyperref = false } ,
2049   nohyperref .value_forbidden:n = true ,
2050 }
2051 \AddToHook { begindocument }
2052 {
2053   \__zrefclever_if_package_loaded:nTF { hyperref }
2054     {
2055       \bool_if:NT \l__zrefclever_hyperlink_bool
2056         { \RequirePackage { zref-hyperref } }
2057     }
2058     {
2059       \bool_if:NT \l__zrefclever_hyperref_warn_bool
2060         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2061       \bool_set_false:N \l__zrefclever_hyperlink_bool
2062     }
2063   \keys_define:nn { zref-clever/reference }
2064     {
2065       hyperref .code:n =
2066         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2067       nohyperref .code:n =
2068         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2069     }
2070 }
```

### nameinlink option

```
2071 \str_new:N \l__zrefclever_nameinlink_str
2072 \keys_define:nn { zref-clever/reference }
2073 {
2074   nameinlink .choice: ,
2075   nameinlink / true .code:n =
2076     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2077   nameinlink / false .code:n =
2078     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2079   nameinlink / single .code:n =
2080     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2081   nameinlink / tsingle .code:n =
2082     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2083   nameinlink .initial:n = tsingle ,
2084   nameinlink .default:n = true ,
2085 }
```

### preposinlink option (deprecated)

```
2086 \keys_define:nn { zref-clever/reference }
2087 {
2088   preposinlink .code:n =
2089     {
2090       % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2091       \msg_warning:nnnn { zref-clever }{ option-deprecated }
2092       { preposinlink } { refbounds }
2093     } ,
2094 }
```

### 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`.

```
2095 \AddToHook { begindocument }
2096 {
2097   \__zrefclever_if_package_loaded:nTF { babel }
2098   {
2099     \tl_set:Nn \l__zrefclever_current_language_tl { \language }

```

```

2100     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2101   }
2102   {
2103     \__zrefclever_if_package_loaded:nTF { polyglossia }
2104     {
2105       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2106       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2107     }
2108     {
2109       \tl_set:Nn \l__zrefclever_current_language_tl { english }
2110       \tl_set:Nn \l__zrefclever_main_language_tl { english }
2111     }
2112   }
2113 }
2114 \keys_define:nn { zref-clever/reference }
2115 {
2116   lang .code:n =
2117   {
2118     \AddToHook { begindocument }
2119     {
2120       \str_case:nnF {#1}
2121       {
2122         { current }
2123         {
2124           \tl_set:Nn \l__zrefclever_ref_language_tl
2125             { \l__zrefclever_current_language_tl }
2126         }
2127         { main }
2128         {
2129           \tl_set:Nn \l__zrefclever_ref_language_tl
2130             { \l__zrefclever_main_language_tl }
2131         }
2132       }
2133     }
2134     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2135     \__zrefclever_language_if_declared:nF {#1}
2136     {
2137       \msg_warning:nnn { zref-clever }
2138         { unknown-language-opt } {#1}
2139     }
2140   }
2141   \__zrefclever_provide_langfile:e
2142     { \l__zrefclever_ref_language_tl }
2143 }
2144 } ,
2145 lang .initial:n = current ,
2146 lang .value_required:n = true ,
2147 }
2148 \AddToHook { begindocument / before }
2149 {
2150   \AddToHook { begindocument }
2151   {

```

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.

```

2152     \keys_define:nn { zref-clever/reference }
2153     {
2154         lang .code:n =
2155         {
2156             \str_case:nnF {#1}
2157             {
2158                 { current }
2159                 {
2160                     \tl_set:Nn \l__zrefclever_ref_language_tl
2161                     { \l__zrefclever_current_language_tl }
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 xref package (<https://github.com/frougon/xref>), 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 / ifffinal .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   nonnudge .meta:n = { nudge = false } ,
2223   nonnudge .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_zceref_note_tl
2290 \keys_define:nn { zref-clever/reference }
2291 {
2292   note .tl_set:N = \l__zrefclever_zceref_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 `cleverf's` optional argument to `\label`.

NOTE `tcolorbox` uses the `reftype` option to support its `label type` option. 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.

Note that, as far as L<sup>A</sup>T<sub>E</sub>X is concerned, a given counter can be reset by *any number of counters*. `\counterwithin` just adds a new “within-counter” for “counter” without removing any other existing ones. However, the data structure of `zref-clever` can only account for *one* enclosing counter. In a way, this is hard to circumvent, because the underlying counter reset behavior works “top-down”, but when looking to a label built from a given counter we need to infer the enclosing counters “bottom-up”. As a result, the reset chain we find is path dependent or, more formally, what `\__zrefclever_counter_reset_by:n` returns depends on the order in which it searches the list of `\l__zrefclever_counter_resetters_seq`, since it stops on the first match. This representation mismatch should not be a problem in most cases. But one should be aware of the limits it imposes.

Consider the following case: the `book` class sets, by default `figure` and `table` counters to be reset every `chapter`, `section` is also reset every `chapter`, of course. Suppose now we say `\counterwithin{figure}{section}`. Technically, `figure` is being reset every `section` and every `chapter`, but since `section` is also reset every `chapter`, the original “`chapter` resets `figure`” behavior is now redundant. Innocuous, but is still there. Now, suppose we want to find which counter is resetting `figure` using `\__zrefclever_counter_reset_by:n`. If `chapter` comes before `section` in `\l__zrefclever_counter_resetters_seq`, `chapter` will be returned, and that’s not what we want. That’s the reason `counterresetters` initial value goes bottom-up in the sectioning level, since we’d expect the nesting of the reset chain to *typically* work top-down.

If, despite all this, unexpected results still ensue, users can take care to “clean” redundant resetting settings with `\counterwithout`. 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.

For the above reasons, since order matters, the `counterresetters` option can only be set by the full list of counters. In other words, users wanting to change this should take the initial value as their starting base.

The `zc@enclcnt` `zref` property, not included by default in the `main` property list, is provided for the purpose of easing the debugging of counter reset chains. So, by adding `\zref@addprop{main}{zc@enclcnt}` you can inspect what the values in the `zc@enclval` property correspond to.

```

2376 \seq_new:N \l__zrefclever_counter_resettters_seq
2377 \keys_define:nn { zref-clever/label }
2378 {
2379   counterresettters .code:n =
2380     { \seq_set_from_clist:Nn \l__zrefclever_counter_resettters_seq {#1} } ,
2381   counterresettters .initial:n =
2382     {
2383       subparagraph ,
2384       paragraph ,
2385       subsubsection ,
2386       subsection ,
2387       section ,
2388       chapter ,
2389       part ,
2390     } ,
2391   counterresettters .value_required:n = true ,
2392 }

```

#### counterresetby option

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

```

2393 \prop_new:N \l__zrefclever_counter_resetby_prop
2394 \keys_define:nn { zref-clever/label }
2395 {
2396   counterresetby .code:n =
2397     {
2398       \keyval_parse:nnn
2399       {
2400         \msg_warning:nnn { zref-clever }
2401         { key-requires-value } { counterresetby }
2402       }
2403       {
2404         \__zrefclever_prop_put_non_empty:Nnn
2405         \l__zrefclever_counter_resetby_prop
2406       }
2407       {#1}
2408     } ,
2409   counterresetby .value_required:n = true ,
2410   counterresetby .initial:n =
2411     {

```

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.

```

2412   enumii = enumi ,

```

```

2413         enumiii = enumii ,
2414         enumiv = enumiii ,
2415     } ,
2416 }

```

### 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.

```

2417 \tl_new:N \l__zrefclever_current_counter_tl
2418 \keys_define:nn { zref-clever/label }
2419 {
2420     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2421     currentcounter .default:n = \@currentcounter ,
2422     currentcounter .initial:n = \@currentcounter ,
2423 }

```

### labelhook option

```

2424 \bool_new:N \l__zrefclever_labelhook_bool
2425 \keys_define:nn { zref-clever/label }
2426 {
2427     labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2428     labelhook .initial:n = true ,
2429     labelhook .default:n = true ,
2430 }

```

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>. Conversely, if `\label` is gobbled, the label hook also won’t be called.

```

2431 \AddToHookWithArguments { label }
2432 {
2433     \bool_if:NT \l__zrefclever_labelhook_bool
2434     { \zref@wrapper@babel \zref@label {#1} }
2435 }

```

### nocompat option

```

2436 \bool_new:N \g__zrefclever_nocompat_bool
2437 \seq_new:N \g__zrefclever_nocompat_modules_seq
2438 \keys_define:nn { zref-clever/reference }
2439 {
2440     nocompat .code:n =
2441     {
2442         \tl_if_empty:nTF {#1}
2443         { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2444         {
2445             \clist_map_inline:nn {#1}
2446             {

```



```

2447         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2448         {
2449             \seq_gput_right:Nn
2450             \g__zrefclever_nocompat_modules_seq {##1}
2451         }
2452     }
2453 }
2454 } ,
2455 }
2456 \AddToHook { begindocument }
2457 {
2458     \keys_define:nn { zref-clever/reference }
2459     {
2460         nocompat .code:n =
2461         {
2462             \msg_warning:nnn { zref-clever }
2463             { option-preamble-only } { nocompat }
2464         }
2465     }
2466 }
2467 \AtEndOfPackage
2468 {
2469     \AddToHook { begindocument }
2470     {
2471         \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2472         { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2473     }
2474 }

```

`\_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 `<module>` 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>}

2475 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2476 {
2477     \AddToHook { begindocument }
2478     {
2479         \bool_if:NF \g__zrefclever_nocompat_bool
2480         { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2481         \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2482     }
2483 }

```

(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.

```
2484 \seq_map_inline:Nn
2485   \g__zrefclever_rf_opts_tl_reference_seq
2486   {
2487     \keys_define:nn { zref-clever/reference }
2488     {
2489       #1 .default:o = \c_novalue_tl ,
2490       #1 .code:n =
2491         {
2492           \tl_if_novalue:nTF {##1}
2493             {
2494               \__zrefclever_opt_tl_unset:c
2495                 { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2496             }
2497             {
2498               \__zrefclever_opt_tl_set:cn
2499                 { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2500                 {##1}
2501             }
2502           } ,
2503         }
2504     }
2505 \keys_define:nn { zref-clever/reference }
2506 {
2507   refpre .code:n =
2508     {
2509       % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2510       \msg_warning:nmmm { zref-clever }{ option-deprecated }
2511       { refpre } { refbounds }
2512     } ,
2513   refpos .code:n =
2514     {
2515       % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2516       \msg_warning:nmmm { zref-clever }{ option-deprecated }
2517       { refpos } { refbounds }
2518     } ,
2519   preref .code:n =
2520     {
2521       % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2522       \msg_warning:nmmm { zref-clever }{ option-deprecated }
2523       { preref } { refbounds }
2524     } ,
2525   postref .code:n =
2526     {
2527       % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2528       \msg_warning:nmmm { zref-clever }{ option-deprecated }
2529       { postref } { refbounds }
2530     } ,
2531 }
2532 \seq_map_inline:Nn
```

```

2533 \g__zrefclever_rf_opts_seq_refbounds_seq
2534 {
2535   \keys_define:nn { zref-clever/reference }
2536   {
2537     #1 .default:o = \c_novalue_tl ,
2538     #1 .code:n =
2539     {
2540       \tl_if_novalue:nTF {##1}
2541       {
2542         \__zrefclever_opt_seq_unset:c
2543         { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2544       }
2545       {
2546         \seq_clear:N \l__zrefclever_tmpa_seq
2547         \__zrefclever_opt_seq_set_clist_split:Nn
2548         \l__zrefclever_tmpa_seq {##1}
2549         \bool_lazy_or:nnTF
2550         { \tl_if_empty_p:n {##1} }
2551         {
2552           \int_compare_p:nNn
2553           { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2554         }
2555         {
2556           \__zrefclever_opt_seq_set_eq:cN
2557           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2558           \l__zrefclever_tmpa_seq
2559         }
2560         {
2561           \msg_warning:nnee { zref-clever }
2562           { refbounds-must-be-four }
2563           {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2564         }
2565       }
2566     } ,
2567   }
2568 }
2569 \seq_map_inline:Nn
2570 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2571 {
2572   \keys_define:nn { zref-clever/reference }
2573   {
2574     #1 .choice: ,
2575     #1 / true .code:n =
2576     {
2577       \__zrefclever_opt_bool_set_true:c
2578       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2579     } ,
2580     #1 / false .code:n =
2581     {
2582       \__zrefclever_opt_bool_set_false:c
2583       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2584     } ,
2585     #1 / unset .code:n =
2586     {

```

```

2587         \_zrefclever_opt_bool_unset:c
2588         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2589     } ,
2590     #1 .default:n = true ,
2591     no #1 .meta:n = { #1 = false } ,
2592     no #1 .value_forbidden:n = true ,
2593 }
2594 }

```

## Package options

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

See <https://github.com/latex3/latex3/issues/1254>.

```

2595 \keys_define:nn { zref-clever }
2596 {
2597     zcsetup .inherit:n =
2598     {
2599         zref-clever/label ,
2600         zref-clever/reference ,
2601     }
2602 }

```

`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.

```

2603 \bool_lazy_and:nnT
2604 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2605 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2606 { \msg_warning:nn { zref-clever } { load-time-options } }

```

## 5 Configuration

### 5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

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

```

2607 \NewDocumentCommand \zcsetup { m }
2608 { \_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>}
```

```

2609 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2610 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2611 \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>}

2612 \NewDocumentCommand \zcRefTypeSetup { m m }
2613 {
2614   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2615   \keys_set:nn { zref-clever/typesetup } {#2}
2616   \tl_clear:N \l__zrefclever_setup_type_tl
2617 }

(End of definition for \zcRefTypeSetup.)

2618 \seq_map_inline:Nn
2619 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2620 {
2621   \keys_define:nn { zref-clever/typesetup }
2622   {
2623     #1 .code:n =
2624     {
2625       \msg_warning:nnn { zref-clever }
2626       { option-not-type-specific } {#1}
2627     } ,
2628   }
2629 }
2630 \seq_map_inline:Nn
2631 \g__zrefclever_rf_opts_tl_typesetup_seq
2632 {
2633   \keys_define:nn { zref-clever/typesetup }
2634   {
2635     #1 .default:o = \c_novalue_tl ,
2636     #1 .code:n =
2637     {
2638       \tl_if_novalue:nTF {##1}
2639       {
2640         \__zrefclever_opt_tl_unset:c
2641         {
2642           \__zrefclever_opt_varname_type:enn
2643           { \l__zrefclever_setup_type_tl } {#1} { tl }
2644         }
2645       }
2646     {

```

```

2647         \__zrefclever_opt_tl_set:cn
2648         {
2649             \__zrefclever_opt_varname_type:enn
2650             { \l__zrefclever_setup_type_tl } {#1} { t1 }
2651         }
2652         {##1}
2653     }
2654 },
2655 }
2656 }
2657 \keys_define:nn { zref-clever/typesetup }
2658 {
2659     endrange .code:n =
2660     {
2661         \str_case:nnF {#1}
2662         {
2663             { ref }
2664             {
2665                 \__zrefclever_opt_tl_clear:c
2666                 {
2667                     \__zrefclever_opt_varname_type:enn
2668                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2669                 }
2670                 \__zrefclever_opt_tl_clear:c
2671                 {
2672                     \__zrefclever_opt_varname_type:enn
2673                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2674                 }
2675             }
2676             { stripprefix }
2677             {
2678                 \__zrefclever_opt_tl_set:cn
2679                 {
2680                     \__zrefclever_opt_varname_type:enn
2681                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2682                 }
2683                 { __zrefclever_get_endrange_stripprefix }
2684                 \__zrefclever_opt_tl_clear:c
2685                 {
2686                     \__zrefclever_opt_varname_type:enn
2687                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2688                 }
2689             }
2690             { pagecomp }
2691             {
2692                 \__zrefclever_opt_tl_set:cn
2693                 {
2694                     \__zrefclever_opt_varname_type:enn
2695                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2696                 }
2697                 { __zrefclever_get_endrange_pagecomp }
2698                 \__zrefclever_opt_tl_clear:c
2699                 {
2700                     \__zrefclever_opt_varname_type:enn

```

```

2701         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2702     }
2703 }
2704 { pagecomp2 }
2705 {
2706     \__zrefclever_opt_tl_set:cn
2707     {
2708         \__zrefclever_opt_varname_type:enn
2709         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2710     }
2711     { __zrefclever_get_endrange_pagecomptwo }
2712     \__zrefclever_opt_tl_clear:c
2713     {
2714         \__zrefclever_opt_varname_type:enn
2715         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2716     }
2717 }
2718 { unset }
2719 {
2720     \__zrefclever_opt_tl_unset:c
2721     {
2722         \__zrefclever_opt_varname_type:enn
2723         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2724     }
2725     \__zrefclever_opt_tl_unset:c
2726     {
2727         \__zrefclever_opt_varname_type:enn
2728         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2729     }
2730 }
2731 }
2732 {
2733     \tl_if_empty:nTF {#1}
2734     {
2735         \msg_warning:nnn { zref-clever }
2736         { endrange-property-undefined } {#1}
2737     }
2738     {
2739         \zref@ifpropundefined {#1}
2740         {
2741             \msg_warning:nnn { zref-clever }
2742             { endrange-property-undefined } {#1}
2743         }
2744         {
2745             \__zrefclever_opt_tl_set:cn
2746             {
2747                 \__zrefclever_opt_varname_type:enn
2748                 { \l__zrefclever_setup_type_tl }
2749                 { endrangefunc } { tl }
2750             }
2751             { __zrefclever_get_endrange_property }
2752             \__zrefclever_opt_tl_set:cn
2753             {
2754                 \__zrefclever_opt_varname_type:enn

```

```

2755             { \l__zrefclever_setup_type_t1 }
2756             { endrangeprop } { t1 }
2757         }
2758         {#1}
2759     }
2760 }
2761 }
2762 } ,
2763 endrange .value_required:n = true ,
2764 }
2765 \keys_define:nn { zref-clever/typesetup }
2766 {
2767     refpre .code:n =
2768     {
2769         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2770         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2771         { refpre } { refbounds }
2772     } ,
2773     refpos .code:n =
2774     {
2775         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2776         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2777         { refpos } { refbounds }
2778     } ,
2779     preref .code:n =
2780     {
2781         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2782         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2783         { preref } { refbounds }
2784     } ,
2785     postref .code:n =
2786     {
2787         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2788         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2789         { postref } { refbounds }
2790     } ,
2791 }
2792 \seq_map_inline:Nn
2793 \g__zrefclever_rf_opts_seq_refbounds_seq
2794 {
2795     \keys_define:nn { zref-clever/typesetup }
2796     {
2797         #1 .default:o = \c_novalue_t1 ,
2798         #1 .code:n =
2799         {
2800             \tl_if_novalue:nTF {##1}
2801             {
2802                 \__zrefclever_opt_seq_unset:c
2803                 {
2804                     \__zrefclever_opt_varname_type:enn
2805                     { \l__zrefclever_setup_type_t1 } {#1} { seq }
2806                 }
2807             }
2808         }

```



```

2809         \seq_clear:N \l__zrefclever_tmpa_seq
2810         \__zrefclever_opt_seq_set_clist_split:Nn
2811         \l__zrefclever_tmpa_seq {##1}
2812         \bool_lazy_or:nnTF
2813         { \tl_if_empty_p:n {##1} }
2814         {
2815             \int_compare_p:nNn
2816             { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2817         }
2818         {
2819             \__zrefclever_opt_seq_set_eq:cN
2820             {
2821                 \__zrefclever_opt_varname_type:enn
2822                 { \l__zrefclever_setup_type_tl } {##1} { seq }
2823             }
2824             \l__zrefclever_tmpa_seq
2825         }
2826         {
2827             \msg_warning:nnee { zref-clever }
2828             { refbounds-must-be-four }
2829             {##1} { \seq_count:N \l__zrefclever_tmpa_seq }
2830         }
2831     }
2832 },
2833 }
2834 }
2835 \seq_map_inline:Nn
2836 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2837 {
2838     \keys_define:nn { zref-clever/typesetup }
2839     {
2840         #1 .choice: ,
2841         #1 / true .code:n =
2842         {
2843             \__zrefclever_opt_bool_set_true:c
2844             {
2845                 \__zrefclever_opt_varname_type:enn
2846                 { \l__zrefclever_setup_type_tl }
2847                 {##1} { bool }
2848             }
2849         } ,
2850         #1 / false .code:n =
2851         {
2852             \__zrefclever_opt_bool_set_false:c
2853             {
2854                 \__zrefclever_opt_varname_type:enn
2855                 { \l__zrefclever_setup_type_tl }
2856                 {##1} { bool }
2857             }
2858         } ,
2859         #1 / unset .code:n =
2860         {
2861             \__zrefclever_opt_bool_unset:c
2862             {

```

```

2863         \_zrefclever_opt_varname_type:enn
2864         { \l__zrefclever_setup_type_tl }
2865         {#1} { bool }
2866     }
2867 },
2868 #1 .default:n = true ,
2869 no #1 .meta:n = { #1 = false } ,
2870 no #1 .value_forbidden:n = true ,
2871 }
2872 }

```

### 5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `<options>` argument of \zcLanguageSetup, any options made before the first `type` key declare “default” (non type-specific) language options. When the `type` key is given with a value, the options following it will set “type-specific” language options for that type. The current type can be switched off by an empty `type` key. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
2873 \NewDocumentCommand \zcLanguageSetup { m m }
2874 {
2875   \group_begin:
2876   \_zrefclever_language_if_declared:nTF {#1}
2877   {
2878     \tl_clear:N \l__zrefclever_setup_type_tl
2879     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2880     \_zrefclever_opt_seq_get:cNF
2881     {
2882       \_zrefclever_opt_varname_language:nnn
2883       {#1} { declension } { seq }
2884     }
2885     \l__zrefclever_lang_declension_seq
2886     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2887     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2888     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2889     {
2890       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2891       \l__zrefclever_lang_decl_case_tl
2892     }
2893     \_zrefclever_opt_seq_get:cNF
2894     {
2895       \_zrefclever_opt_varname_language:nnn
2896       {#1} { gender } { seq }
2897     }
2898     \l__zrefclever_lang_gender_seq
2899     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2900     \keys_set:nn { zref-clever/langsetup } {#2}
2901   }
2902   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2903   \group_end:

```

```

2904 }
2905 \@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.

2906 \keys_define:nn { zref-clever/langsetup }
2907 {
2908   type .code:n =
2909   {
2910     \tl_if_empty:nTF {#1}
2911     { \tl_clear:N \l__zrefclever_setup_type_tl }
2912     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2913   } ,
2914   case .code:n =
2915   {
2916     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2917     {
2918       \msg_warning:nnee { zref-clever } { language-no-decl-setup }
2919       { \l__zrefclever_setup_language_tl } {#1}
2920     }
2921     {
2922       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2923       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2924       {
2925         \msg_warning:nnee { zref-clever } { unknown-decl-case }
2926         {#1} { \l__zrefclever_setup_language_tl }
2927         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2928         \l__zrefclever_lang_decl_case_tl
2929       }
2930     }
2931   } ,
2932   case .value_required:n = true ,
2933   gender .value_required:n = true ,
2934   gender .code:n =
2935   {
2936     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2937     {
2938       \msg_warning:nnee { zref-clever } { language-no-gender }
2939       { \l__zrefclever_setup_language_tl } { gender } {#1}
2940     }
2941     {
2942       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2943       {
2944         \msg_warning:nnn { zref-clever }
2945         { option-only-type-specific } { gender }
2946       }
2947       {
2948         \seq_clear:N \l__zrefclever_tmpa_seq
2949         \clist_map_inline:nn {#1}
2950         {
2951           \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2952           { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
2953           {

```

```

2954         \msg_warning:nnee { zref-clever }
2955         { gender-not-declared }
2956         { \l__zrefclever_setup_language_tl } {##1}
2957     }
2958 }
2959 \__zrefclever_opt_seq_gset_eq:cN
2960 {
2961     \__zrefclever_opt_varname_lang_type:een
2962     { \l__zrefclever_setup_language_tl }
2963     { \l__zrefclever_setup_type_tl }
2964     { gender }
2965     { seq }
2966 }
2967 \l__zrefclever_tmpa_seq
2968 }
2969 } ,
2970 }
2971 }
2972 \seq_map_inline:Nn
2973 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2974 {
2975     \keys_define:nn { zref-clever/langsetup }
2976     {
2977         #1 .value_required:n = true ,
2978         #1 .code:n =
2979         {
2980             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2981             {
2982                 \__zrefclever_opt_tl_gset:cn
2983                 {
2984                     \__zrefclever_opt_varname_lang_default:enn
2985                     { \l__zrefclever_setup_language_tl } {##1} { tl }
2986                 }
2987                 {##1}
2988             }
2989             {
2990                 \msg_warning:nnn { zref-clever }
2991                 { option-not-type-specific } {##1}
2992             }
2993         } ,
2994     }
2995 }
2996 \seq_map_inline:Nn
2997 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2998 {
2999     \keys_define:nn { zref-clever/langsetup }
3000     {
3001         #1 .value_required:n = true ,
3002         #1 .code:n =
3003         {
3004             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3005             {
3006                 \__zrefclever_opt_tl_gset:cn
3007                 {

```

```

3008         \_zrefclever_opt_varname_lang_default:enn
3009         { \l_zrefclever_setup_language_tl } {#1} { tl }
3010     }
3011     {##1}
3012 }
3013 {
3014     \_zrefclever_opt_tl_gset:cn
3015     {
3016         \_zrefclever_opt_varname_lang_type:eenn
3017         { \l_zrefclever_setup_language_tl }
3018         { \l_zrefclever_setup_type_tl }
3019         {#1} { tl }
3020     }
3021     {##1}
3022 }
3023 },
3024 }
3025 }
3026 \keys_define:nn { zref-clever/langsetup }
3027 {
3028     endrange .value_required:n = true ,
3029     endrange .code:n =
3030     {
3031         \str_case:nnF {#1}
3032         {
3033             { ref }
3034             {
3035                 \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3036                 {
3037                     \_zrefclever_opt_tl_gclear:c
3038                     {
3039                         \_zrefclever_opt_varname_lang_default:enn
3040                         { \l_zrefclever_setup_language_tl }
3041                         { endrangefunc } { tl }
3042                     }
3043                     \_zrefclever_opt_tl_gclear:c
3044                     {
3045                         \_zrefclever_opt_varname_lang_default:enn
3046                         { \l_zrefclever_setup_language_tl }
3047                         { endrangeprop } { tl }
3048                     }
3049                 }
3050             }
3051             \_zrefclever_opt_tl_gclear:c
3052             {
3053                 \_zrefclever_opt_varname_lang_type:eenn
3054                 { \l_zrefclever_setup_language_tl }
3055                 { \l_zrefclever_setup_type_tl }
3056                 { endrangefunc } { tl }
3057             }
3058             \_zrefclever_opt_tl_gclear:c
3059             {
3060                 \_zrefclever_opt_varname_lang_type:eenn
3061                 { \l_zrefclever_setup_language_tl }

```

```

3062         { \l__zrefclever_setup_type_tl }
3063         { endrangeprop } { tl }
3064     }
3065 }
3066 }
3067 { stripprefix }
3068 {
3069   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3070   {
3071     \__zrefclever_opt_tl_gset:cn
3072     {
3073       \__zrefclever_opt_varname_lang_default:enn
3074       { \l__zrefclever_setup_language_tl }
3075       { endrangefunc } { tl }
3076     }
3077     { __zrefclever_get_endrange_stripprefix }
3078     \__zrefclever_opt_tl_gclear:c
3079     {
3080       \__zrefclever_opt_varname_lang_default:enn
3081       { \l__zrefclever_setup_language_tl }
3082       { endrangeprop } { tl }
3083     }
3084   }
3085   {
3086     \__zrefclever_opt_tl_gset:cn
3087     {
3088       \__zrefclever_opt_varname_lang_type:eenn
3089       { \l__zrefclever_setup_language_tl }
3090       { \l__zrefclever_setup_type_tl }
3091       { endrangefunc } { tl }
3092     }
3093     { __zrefclever_get_endrange_stripprefix }
3094     \__zrefclever_opt_tl_gclear:c
3095     {
3096       \__zrefclever_opt_varname_lang_type:eenn
3097       { \l__zrefclever_setup_language_tl }
3098       { \l__zrefclever_setup_type_tl }
3099       { endrangeprop } { tl }
3100     }
3101   }
3102 }
3103 { pagecomp }
3104 {
3105   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3106   {
3107     \__zrefclever_opt_tl_gset:cn
3108     {
3109       \__zrefclever_opt_varname_lang_default:enn
3110       { \l__zrefclever_setup_language_tl }
3111       { endrangefunc } { tl }
3112     }
3113     { __zrefclever_get_endrange_pagecomp }
3114     \__zrefclever_opt_tl_gclear:c
3115     {

```

```

3116         \_zrefclever_opt_varname_lang_default:enn
3117         { \l_zrefclever_setup_language_tl }
3118         { endrangeprop } { tl }
3119     }
3120 }
3121 {
3122     \_zrefclever_opt_tl_gset:cn
3123     {
3124         \_zrefclever_opt_varname_lang_type:eenn
3125         { \l_zrefclever_setup_language_tl }
3126         { \l_zrefclever_setup_type_tl }
3127         { endrangefunc } { tl }
3128     }
3129     { __zrefclever_get_endrange_pagecomp }
3130     \_zrefclever_opt_tl_gclear:c
3131     {
3132         \_zrefclever_opt_varname_lang_type:eenn
3133         { \l_zrefclever_setup_language_tl }
3134         { \l_zrefclever_setup_type_tl }
3135         { endrangeprop } { tl }
3136     }
3137 }
3138 }
3139 { pagecomp2 }
3140 {
3141     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3142     {
3143         \_zrefclever_opt_tl_gset:cn
3144         {
3145             \_zrefclever_opt_varname_lang_default:enn
3146             { \l_zrefclever_setup_language_tl }
3147             { endrangefunc } { tl }
3148         }
3149         { __zrefclever_get_endrange_pagecomptwo }
3150         \_zrefclever_opt_tl_gclear:c
3151         {
3152             \_zrefclever_opt_varname_lang_default:enn
3153             { \l_zrefclever_setup_language_tl }
3154             { endrangeprop } { tl }
3155         }
3156     }
3157 }
3158     \_zrefclever_opt_tl_gset:cn
3159     {
3160         \_zrefclever_opt_varname_lang_type:eenn
3161         { \l_zrefclever_setup_language_tl }
3162         { \l_zrefclever_setup_type_tl }
3163         { endrangefunc } { tl }
3164     }
3165     { __zrefclever_get_endrange_pagecomptwo }
3166     \_zrefclever_opt_tl_gclear:c
3167     {
3168         \_zrefclever_opt_varname_lang_type:eenn
3169         { \l_zrefclever_setup_language_tl }

```

```

3170             { \l__zrefclever_setup_type_tl }
3171             { endrangeprop } { tl }
3172         }
3173     }
3174 }
3175 }
3176 {
3177     \tl_if_empty:nTF {#1}
3178     {
3179         \msg_warning:nnn { zref-clever }
3180         { endrange-property-undefined } {#1}
3181     }
3182     {
3183         \zref@ifpropundefined {#1}
3184         {
3185             \msg_warning:nnn { zref-clever }
3186             { endrange-property-undefined } {#1}
3187         }
3188         {
3189             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3190             {
3191                 \__zrefclever_opt_tl_gset:cn
3192                 {
3193                     \__zrefclever_opt_varname_lang_default:enn
3194                     { \l__zrefclever_setup_language_tl }
3195                     { endrangefunc } { tl }
3196                 }
3197                 { __zrefclever_get_endrange_property }
3198                 \__zrefclever_opt_tl_gset:cn
3199                 {
3200                     \__zrefclever_opt_varname_lang_default:enn
3201                     { \l__zrefclever_setup_language_tl }
3202                     { endrangeprop } { tl }
3203                 }
3204                 {#1}
3205             }
3206             {
3207                 \__zrefclever_opt_tl_gset:cn
3208                 {
3209                     \__zrefclever_opt_varname_lang_type:eenn
3210                     { \l__zrefclever_setup_language_tl }
3211                     { \l__zrefclever_setup_type_tl }
3212                     { endrangefunc } { tl }
3213                 }
3214                 { __zrefclever_get_endrange_property }
3215                 \__zrefclever_opt_tl_gset:cn
3216                 {
3217                     \__zrefclever_opt_varname_lang_type:eenn
3218                     { \l__zrefclever_setup_language_tl }
3219                     { \l__zrefclever_setup_type_tl }
3220                     { endrangeprop } { tl }
3221                 }
3222                 {#1}
3223             }

```



```

3224     }
3225   }
3226 } ,
3227 } ,
3228 }
3229 \keys_define:nn { zref-clever/langsetup }
3230 {
3231   refpre .code:n =
3232   {
3233     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3234     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3235     { refpre } { refbounds }
3236   } ,
3237   refpos .code:n =
3238   {
3239     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3240     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3241     { refpos } { refbounds }
3242   } ,
3243   preref .code:n =
3244   {
3245     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3246     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3247     { preref } { refbounds }
3248   } ,
3249   postref .code:n =
3250   {
3251     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3252     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3253     { postref } { refbounds }
3254   } ,
3255 }
3256 \seq_map_inline:Nn
3257 \g__zrefclever_rf_opts_tl_type_names_seq
3258 {
3259   \keys_define:nn { zref-clever/langsetup }
3260   {
3261     #1 .value_required:n = true ,
3262     #1 .code:n =
3263     {
3264       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3265       {
3266         \msg_warning:nnn { zref-clever }
3267         { option-only-type-specific } {#1}
3268       }
3269       {
3270         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3271         {
3272           \__zrefclever_opt_tl_gset:cn
3273           {
3274             \__zrefclever_opt_varname_lang_type:eenn
3275             { \l__zrefclever_setup_language_tl }
3276             { \l__zrefclever_setup_type_tl }
3277             {#1} { tl }

```

```

3278     }
3279     {##1}
3280   }
3281   {
3282     \__zrefclever_opt_tl_gset:cn
3283     {
3284       \__zrefclever_opt_varname_lang_type:een
3285       { \l__zrefclever_setup_language_tl }
3286       { \l__zrefclever_setup_type_tl }
3287       { \l__zrefclever_lang_decl_case_tl - #1 }
3288       { tl }
3289     }
3290     {##1}
3291   }
3292 } ,
3293 }
3294 }
3295 }
3296 \seq_map_inline:Nn
3297 \g__zrefclever_rf_opts_seq_refbounds_seq
3298 {
3299   \keys_define:nn { zref-clever/langsetup }
3300   {
3301     #1 .value_required:n = true ,
3302     #1 .code:n =
3303     {
3304       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3305       {
3306         \seq_gclear:N \g__zrefclever_tmpa_seq
3307         \__zrefclever_opt_seq_gset_clist_split:Nn
3308         \g__zrefclever_tmpa_seq {##1}
3309         \bool_lazy_or:nnTF
3310         { \tl_if_empty_p:n {##1} }
3311         {
3312           \int_compare_p:nNn
3313           { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3314         }
3315         {
3316           \__zrefclever_opt_seq_gset_eq:cN
3317           {
3318             \__zrefclever_opt_varname_lang_default:enn
3319             { \l__zrefclever_setup_language_tl }
3320             {##1} { seq }
3321           }
3322           \g__zrefclever_tmpa_seq
3323         }
3324         {
3325           \msg_warning:nnee { zref-clever }
3326           { refbounds-must-be-four }
3327           {##1} { \seq_count:N \g__zrefclever_tmpa_seq }
3328         }
3329       }
3330     }
3331     \seq_gclear:N \g__zrefclever_tmpa_seq

```

```

3332         \__zrefclever_opt_seq_gset_clist_split:Nn
3333         \g__zrefclever_tmpa_seq {##1}
3334     \bool_lazy_or:nnTF
3335         { \tl_if_empty_p:n {##1} }
3336         {
3337             \int_compare_p:nNn
3338             { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3339         }
3340         {
3341             \__zrefclever_opt_seq_gset_eq:cN
3342             {
3343                 \__zrefclever_opt_varname_lang_type:eenn
3344                 { \l__zrefclever_setup_language_tl }
3345                 { \l__zrefclever_setup_type_tl } {#1} { seq }
3346             }
3347             \g__zrefclever_tmpa_seq
3348         }
3349         {
3350             \msg_warning:nnee { zref-clever }
3351             { refbounds-must-be-four }
3352             {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
3353         }
3354     }
3355 } ,
3356 }
3357 }
3358 \seq_map_inline:Nn
3359 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3360 {
3361     \keys_define:nn { zref-clever/langsetup }
3362     {
3363         #1 .choice: ,
3364         #1 / true .code:n =
3365         {
3366             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3367             {
3368                 \__zrefclever_opt_bool_gset_true:c
3369                 {
3370                     \__zrefclever_opt_varname_lang_default:enn
3371                     { \l__zrefclever_setup_language_tl }
3372                     {#1} { bool }
3373                 }
3374             }
3375             {
3376                 \__zrefclever_opt_bool_gset_true:c
3377                 {
3378                     \__zrefclever_opt_varname_lang_type:eenn
3379                     { \l__zrefclever_setup_language_tl }
3380                     { \l__zrefclever_setup_type_tl }
3381                     {#1} { bool }
3382                 }
3383             }
3384         } ,
3385         #1 / false .code:n =

```

```

3386     {
3387         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3388         {
3389             \__zrefclever_opt_bool_gset_false:c
3390             {
3391                 \__zrefclever_opt_varname_lang_default:enn
3392                 { \l__zrefclever_setup_language_tl }
3393                 {#1} { bool }
3394             }
3395         }
3396         {
3397             \__zrefclever_opt_bool_gset_false:c
3398             {
3399                 \__zrefclever_opt_varname_lang_type:eenn
3400                 { \l__zrefclever_setup_language_tl }
3401                 { \l__zrefclever_setup_type_tl }
3402                 {#1} { bool }
3403             }
3404         }
3405     } ,
3406     #1 .default:n = true ,
3407     no #1 .meta:n = { #1 = false } ,
3408     no #1 .value_forbidden:n = true ,
3409 }
3410 }

```

## 6 User interface

### 6.1 \zcref

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

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

```

3411 \NewDocumentCommand \zcref { s O { } m }
3412 { \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>}
```

```

3413 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3414 {
3415     \group_begin:

```

Set options.

```
3416     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

3417     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3418     \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.

```
3419     \__zrefclever_provide_langfile:e { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3420     \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3421     \bool_lazy_and:nnT
3422     { \l__zrefclever_zrefcheck_available_bool }
3423     { \l__zrefclever_zcref_with_check_bool }
3424     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3425     \bool_lazy_or:nnT
3426     { \l__zrefclever_typeset_sort_bool }
3427     { \l__zrefclever_typeset_range_bool }
3428     { \__zrefclever_sort_labels: }
```

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

```
3429     \group_begin:
3430     \l__zrefclever_ref_typeset_font_tl
3431     \__zrefclever_typeset_refs:
3432     \group_end:
```

Typeset note.

```
3433     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3434     {
3435         \__zrefclever_get_rf_opt_tl:neeN { notesep }
3436         { \l__zrefclever_label_type_a_tl }
3437         { \l__zrefclever_ref_language_tl }
3438         \l__zrefclever_tmpa_tl
3439         \l__zrefclever_tmpa_tl
3440         \l__zrefclever_zcref_note_tl
3441     }
```

Integration with `zref-check`.

```
3442     \bool_lazy_and:nnT
3443     { \l__zrefclever_zrefcheck_available_bool }
3444     { \l__zrefclever_zcref_with_check_bool }
3445     {
3446         \zrefcheck_zcref_end_label_maybe:
3447         \zrefcheck_zcref_run_checks_on_labels:n
3448         { \l__zrefclever_zcref_labels_seq }
3449     }
```

Integration with `mathtools`.

```
3450     \bool_if:NT \l__zrefclever_mathtools_loaded_bool
3451     {
3452         \__zrefclever_mathtools_showonlyrefs:n
3453         { \l__zrefclever_zcref_labels_seq }
3454     }
3455     \group_end:
3456 }
```

(End of definition for `\__zrefclever_zcref:nmmn`.)

`\l_zrefclever_zcref_labels_seq`  
`\l_zrefclever_link_star_bool`

```
3457 \seq_new:N \l__zrefclever_zcref_labels_seq
3458 \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{*}[\langle options \rangle]{\langle labels \rangle}
```

```
3459 \NewDocumentCommand \zcpageref { s O { } m }
3460 {
3461   \group_begin:
3462     \IfBooleanT {#1}
3463       { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3464     \zcref [#2, ref = page] {#3}
3465   \group_end:
3466 }
```

(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.

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

`\l_zrefclever_label_type_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`

```
3467 \tl_new:N \l__zrefclever_label_type_a_tl
3468 \tl_new:N \l__zrefclever_label_type_b_tl
3469 \tl_new:N \l__zrefclever_label_enclval_a_tl
3470 \tl_new:N \l__zrefclever_label_enclval_b_tl
3471 \tl_new:N \l__zrefclever_label_extdoc_a_tl
3472 \tl_new:N \l__zrefclever_label_extdoc_b_tl
```

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

`\l_zrefclever_sort_decided_bool`

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

```
3473 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End of definition for `\l__zrefclever_sort_decided_bool`.)

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

```
3474 \int_new:N \l__zrefclever_sort_prior_a_int
3475 \int_new:N \l__zrefclever_sort_prior_b_int
```

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

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

```
3476 \seq_new:N \l__zrefclever_label_types_seq
```

(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.

```
3477 \cs_new_protected:Npn \__zrefclever_sort_labels:
3478 {
```

Store label types sequence.

```
3479   \seq_clear:N \l__zrefclever_label_types_seq
3480   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
3481   {
3482     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3483     \__zrefclever_label_type_put_new_right:n
3484   }
```

Sort.

```
3485   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3486   {
3487     \zref@ifrefundefined {##1}
3488     {
3489       \zref@ifrefundefined {##2}
3490       {
3491         % Neither label is defined.
3492         \sort_return_same:
3493       }
3494       {
3495         % The second label is defined, but the first isn't, leave the
3496         % undefined first (to be more visible).
3497         \sort_return_same:
3498       }
3499     }
3500     {
3501       \zref@ifrefundefined {##2}
3502       {
3503         % The first label is defined, but the second isn't, bring the
```

```

3504         % second forward.
3505         \sort_return_swapped:
3506     }
3507     {
3508         % The interesting case: both labels are defined. References
3509         % to the "default" property or to the "page" are quite
3510         % different with regard to sorting, so we branch them here to
3511         % specialized functions.
3512         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3513             { \__zrefclever_sort_page:n {##1} {##2} }
3514             { \__zrefclever_sort_default:n {##1} {##2} }
3515     }
3516 }
3517 }
3518 }

```

(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>}

```

```

3519 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3520 {
3521     \__zrefclever_extract_default:Nnnn
3522     \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3523     \seq_if_in:NVF \l__zrefclever_label_types_seq
3524     \l__zrefclever_label_type_a_tl
3525     {
3526         \seq_put_right:NV \l__zrefclever_label_types_seq
3527         \l__zrefclever_label_type_a_tl
3528     }
3529 }

```

(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>}

```

```

3530 \cs_new_protected:Npn \__zrefclever_sort_default:mn #1#2
3531 {
3532     \__zrefclever_extract_default:Nnnn
3533     \l__zrefclever_label_type_a_tl {#1} { zc@type } { zc@missingtype }

```



```

3534 \zrefclever_extract_default:Nnnn
3535 \l__zrefclever_label_type_b_tl {#2} { zc@type } { zc@missingtype }
3536 \tl_if_eq:NNTF
3537 \l__zrefclever_label_type_a_tl
3538 \l__zrefclever_label_type_b_tl
3539 { \zrefclever_sort_default_same_type:nn {#1} {#2} }
3540 { \zrefclever_sort_default_different_types:nn {#1} {#2} }
3541 }

```

(End of definition for \zrefclever\_sort\_default:nn.)

\zrefclever\_sort\_default\_same\_type:nn

```

\zrefclever_sort_default_same_type:nn {(label a)} {(label b)}
3542 \cs_new_protected:Npn \zrefclever_sort_default_same_type:nn #1#2
3543 {
3544 \zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3545 {#1} { zc@enclval } { }
3546 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3547 \zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3548 {#2} { zc@enclval } { }
3549 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3550 \zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3551 {#1} { externaldocument } { }
3552 \zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3553 {#2} { externaldocument } { }
3554 \bool_set_false:N \l__zrefclever_sort_decided_bool
3555 % First we check if there's any "external document" difference (coming
3556 % from `zref-xr') and, if so, sort based on that.
3557 \tl_if_eq:NNF
3558 \l__zrefclever_label_extdoc_a_tl
3559 \l__zrefclever_label_extdoc_b_tl
3560 {
3561 \bool_if:nTF
3562 {
3563 \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3564 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3565 }
3566 {
3567 \bool_set_true:N \l__zrefclever_sort_decided_bool
3568 \sort_return_same:
3569 }
3570 {
3571 \bool_if:nTF
3572 {
3573 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3574 \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3575 }
3576 {
3577 \bool_set_true:N \l__zrefclever_sort_decided_bool
3578 \sort_return_swapped:
3579 }
3580 {
3581 \bool_set_true:N \l__zrefclever_sort_decided_bool
3582 % Two different "external documents": last resort, sort by the
3583 % document name itself.

```

```

3584         \str_compare:eNeTF
3585         { \l__zrefclever_label_extdoc_b_tl } <
3586         { \l__zrefclever_label_extdoc_a_tl }
3587         { \sort_return_swapped: }
3588         { \sort_return_same:   }
3589     }
3590 }
3591 }
3592 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3593 {
3594     \bool_if:nTF
3595     {
3596         % Both are empty: neither label has any (further) "enclosing
3597         % counters" (left).
3598         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3599         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3600     }
3601     {
3602         \bool_set_true:N \l__zrefclever_sort_decided_bool
3603         \int_compare:nNnTF
3604         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3605         >
3606         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3607         { \sort_return_swapped: }
3608         { \sort_return_same:   }
3609     }
3610     {
3611         \bool_if:nTF
3612         {
3613             % `a' is empty (and `b' is not): `b' may be nested in `a'.
3614             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3615         }
3616         {
3617             \bool_set_true:N \l__zrefclever_sort_decided_bool
3618             \int_compare:nNnTF
3619             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3620             >
3621             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3622             { \sort_return_swapped: }
3623             { \sort_return_same:   }
3624         }
3625         {
3626             \bool_if:nTF
3627             {
3628                 % `b' is empty (and `a' is not): `a' may be nested in `b'.
3629                 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3630             }
3631             {
3632                 \bool_set_true:N \l__zrefclever_sort_decided_bool
3633                 \int_compare:nNnTF
3634                 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3635                 <
3636                 { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3637                 { \sort_return_same:   }

```

```

3638         { \sort_return_swapped: }
3639     }
3640     {
3641         % Neither is empty: we can compare the values of the
3642         % current enclosing counter in the loop, if they are
3643         % equal, we are still in the loop, if they are not, a
3644         % sorting decision can be made directly.
3645         \int_compare:nNnTF
3646         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3647         =
3648         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3649         {
3650             \tl_set:Ne \l__zrefclever_label_enclval_a_tl
3651             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3652             \tl_set:Ne \l__zrefclever_label_enclval_b_tl
3653             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3654         }
3655         {
3656             \bool_set_true:N \l__zrefclever_sort_decided_bool
3657             \int_compare:nNnTF
3658             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3659             >
3660             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3661             { \sort_return_swapped: }
3662             { \sort_return_same: }
3663         }
3664     }
3665 }
3666 }
3667 }
3668 }

```

(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>}

```

```

3669 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3670 {

```

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”.

```

3671     \int_zero:N \l__zrefclever_sort_prior_a_int
3672     \int_zero:N \l__zrefclever_sort_prior_b_int
3673     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3674     {
3675         \tl_if_eq:nnTF {##2} {{othertypes}}
3676         {
3677             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3678             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3679             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3680             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3681         }
3682         {
3683             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}

```

```

3684         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3685     {
3686         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3687         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3688     }
3689 }
3690 }

```

Then do the actual sorting.

```

3691 \bool_if:nTF
3692 {
3693     \int_compare_p:nNn
3694     { \l__zrefclever_sort_prior_a_int } <
3695     { \l__zrefclever_sort_prior_b_int }
3696 }
3697 { \sort_return_same: }
3698 {
3699     \bool_if:nTF
3700     {
3701         \int_compare_p:nNn
3702         { \l__zrefclever_sort_prior_a_int } >
3703         { \l__zrefclever_sort_prior_b_int }
3704     }
3705     { \sort_return_swapped: }
3706     {
3707         % Sort priorities are equal: the type that occurs first in
3708         % `labels', as given by the user, is kept (or brought) forward.
3709         \seq_map_inline:Nn \l__zrefclever_label_types_seq
3710         {
3711             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3712             { \seq_map_break:n { \sort_return_same: } }
3713             {
3714                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3715                 { \seq_map_break:n { \sort_return_swapped: } }
3716             }
3717         }
3718     }
3719 }
3720 }

```

(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)}
3721 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3722 {
3723     \int_compare:nNnTF
3724     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3725     >

```

```

3726     { \_zrefclever_extract:nnn {#2} { abspage } { -1 } }
3727     { \sort_return_swapped: }
3728     { \sort_return_same:    }
3729   }

```

*(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 do 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 arbitrarily 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 3730 \seq_new:N \l__zrefclever_typeset_labels_seq
\l__zrefclever_last_of_type_bool 3731 \bool_new:N \l__zrefclever_typeset_last_bool
3732 \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 3733 \int_new:N \l__zrefclever_type_count_int
\l__zrefclever_ref_count_int 3734 \int_new:N \l__zrefclever_label_count_int
3735 \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 3736 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_typeset_queue_prev_tl 3737 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_typeset_queue_curr_tl 3738 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_type_first_label_tl 3739 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
\l__zrefclever_type_first_label_type_tl

```

```

3740 \tl_new:N \l__zrefclever_type_first_label_tl
3741 \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_type_name_tl
  \l__zrefclever_name_in_link_bool
\l__zrefclever_type_name_missing_bool
  \l__zrefclever_name_format_tl
\l__zrefclever_name_format_fallback_tl
  \l__zrefclever_type_name_gender_seq

```

*(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_count_int
\l__zrefclever_range_same_count_int
  \l__zrefclever_range_beg_label_tl
\l__zrefclever_range_beg_is_first_bool
  \l__zrefclever_range_end_ref_tl
\l__zrefclever_next_maybe_range_bool
  \l__zrefclever_next_is_same_bool

```

*(End of definition for \l\_\_zrefclever\_range\_count\_int and others.)*

\l\_\_zrefclever\_tpairsep\_tl \l\_\_zrefclever\_tlistsep\_tl Auxiliary variables for \\_\_zrefclever\_typeset\_refs: separators, and font and other options.

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangesep_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_tl
  \l__zrefclever_endrangefunc_tl
  \l__zrefclever_endrangeprop_tl
\l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool
  \l__zrefclever_rangetopair_bool

```

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

\l\_\_zrefclever\_refbounds\_first\_seq \l\_\_zrefclever\_refbounds\_first\_sg\_seq \l\_\_zrefclever\_refbounds\_first\_pb\_seq \l\_\_zrefclever\_refbounds\_first\_rb\_seq Auxiliary variables for \\_\_zrefclever\_typeset\_refs:: advanced reference format options.

```

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

```

```

3777 \seq_new:N \l__zrefclever_refbounds_last_seq
3778 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3779 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3780 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3781 \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`.

```

3782 \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`.

```

3783 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3784 {
3785   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3786   \l__zrefclever_zcref_labels_seq
3787   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3788   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3789   \tl_clear:N \l__zrefclever_type_first_label_tl
3790   \tl_clear:N \l__zrefclever_type_first_label_type_tl
3791   \tl_clear:N \l__zrefclever_range_beg_label_tl
3792   \tl_clear:N \l__zrefclever_range_end_ref_tl
3793   \int_zero:N \l__zrefclever_label_count_int
3794   \int_zero:N \l__zrefclever_type_count_int
3795   \int_zero:N \l__zrefclever_ref_count_int
3796   \int_zero:N \l__zrefclever_range_count_int
3797   \int_zero:N \l__zrefclever_range_same_count_int
3798   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3799   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3800   % Get type block options (not type-specific).
3801   \__zrefclever_get_rf_opt_tl:neeN { tpairsep }
3802   { \l__zrefclever_label_type_a_tl }
3803   { \l__zrefclever_ref_language_tl }
3804   \l__zrefclever_tpairsep_tl
3805   \__zrefclever_get_rf_opt_tl:neeN { tlistsep }
3806   { \l__zrefclever_label_type_a_tl }
3807   { \l__zrefclever_ref_language_tl }
3808   \l__zrefclever_tlistsep_tl
3809   \__zrefclever_get_rf_opt_tl:neeN { tlastsep }
3810   { \l__zrefclever_label_type_a_tl }
3811   { \l__zrefclever_ref_language_tl }
3812   \l__zrefclever_tlastsep_tl
3813   % Process label stack.
3814   \bool_set_false:N \l__zrefclever_typeset_last_bool
3815   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3816   {
3817     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3818     \l__zrefclever_label_a_tl

```



```

3819 \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3820 {
3821   \tl_clear:N \l__zrefclever_label_b_tl
3822   \bool_set_true:N \l__zrefclever_typeset_last_bool
3823 }
3824 {
3825   \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3826   \l__zrefclever_label_b_tl
3827 }
3828 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3829 {
3830   \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3831   \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3832 }
3833 {
3834   \__zrefclever_extract_default:NVnn
3835   \l__zrefclever_label_type_a_tl
3836   \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3837   \__zrefclever_extract_default:NVnn
3838   \l__zrefclever_label_type_b_tl
3839   \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3840 }
3841 % First, we establish whether the "current label" (i.e. `a') is the
3842 % last one of its type. This can happen because the "next label"
3843 % (i.e. `b') is of a different type (or different definition status),
3844 % or because we are at the end of the list.
3845 \bool_if:NTF \l__zrefclever_typeset_last_bool
3846 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3847 {
3848   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3849   {
3850     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3851     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3852     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3853   }
3854   {
3855     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3856     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3857     {
3858       % Neither is undefined, we must check the types.
3859       \tl_if_eq:NNTF
3860         \l__zrefclever_label_type_a_tl
3861         \l__zrefclever_label_type_b_tl
3862         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3863         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3864       }
3865     }
3866   }
3867 % Handle warnings in case of reference or type undefined.
3868 % Test: `zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3869 \zref@refused { \l__zrefclever_label_a_tl }
3870 % Test: `zc-typeset01.lvt': "Typeset refs: warn missing type"
3871 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3872 {}

```

```

3873 {
3874   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3875   {
3876     \msg_warning:nne { zref-clever } { missing-type }
3877     { \l__zrefclever_label_a_tl }
3878   }
3879   \zref@ifrefcontainsprop
3880   { \l__zrefclever_label_a_tl }
3881   { \l__zrefclever_ref_property_tl }
3882   { }
3883   {
3884     \msg_warning:nnee { zref-clever } { missing-property }
3885     { \l__zrefclever_ref_property_tl }
3886     { \l__zrefclever_label_a_tl }
3887   }
3888 }
3889 % Get possibly type-specific separators, refbounds, font and other
3890 % options, once per type.
3891 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3892 {
3893   \__zrefclever_get_rf_opt_tl:neeN { namesep }
3894   { \l__zrefclever_label_type_a_tl }
3895   { \l__zrefclever_ref_language_tl }
3896   \l__zrefclever_namesep_tl
3897   \__zrefclever_get_rf_opt_tl:neeN { pairsep }
3898   { \l__zrefclever_label_type_a_tl }
3899   { \l__zrefclever_ref_language_tl }
3900   \l__zrefclever_pairsep_tl
3901   \__zrefclever_get_rf_opt_tl:neeN { listsep }
3902   { \l__zrefclever_label_type_a_tl }
3903   { \l__zrefclever_ref_language_tl }
3904   \l__zrefclever_listsep_tl
3905   \__zrefclever_get_rf_opt_tl:neeN { lastsep }
3906   { \l__zrefclever_label_type_a_tl }
3907   { \l__zrefclever_ref_language_tl }
3908   \l__zrefclever_lastsep_tl
3909   \__zrefclever_get_rf_opt_tl:neeN { rangesep }
3910   { \l__zrefclever_label_type_a_tl }
3911   { \l__zrefclever_ref_language_tl }
3912   \l__zrefclever_rangesep_tl
3913   \__zrefclever_get_rf_opt_tl:neeN { namefont }
3914   { \l__zrefclever_label_type_a_tl }
3915   { \l__zrefclever_ref_language_tl }
3916   \l__zrefclever_namefont_tl
3917   \__zrefclever_get_rf_opt_tl:neeN { reffont }
3918   { \l__zrefclever_label_type_a_tl }
3919   { \l__zrefclever_ref_language_tl }
3920   \l__zrefclever_reffont_tl
3921   \__zrefclever_get_rf_opt_tl:neeN { endrangefunc }
3922   { \l__zrefclever_label_type_a_tl }
3923   { \l__zrefclever_ref_language_tl }
3924   \l__zrefclever_endrangefunc_tl
3925   \__zrefclever_get_rf_opt_tl:neeN { endrangeprop }
3926   { \l__zrefclever_label_type_a_tl }

```

```

3927         { \l__zrefclever_ref_language_tl }
3928         \l__zrefclever_endrangeprop_tl
3929 \__zrefclever_get_rf_opt_bool:neeN { cap } { false }
3930     { \l__zrefclever_label_type_a_tl }
3931     { \l__zrefclever_ref_language_tl }
3932     \l__zrefclever_cap_bool
3933 \__zrefclever_get_rf_opt_bool:neeN { abbrev } { false }
3934     { \l__zrefclever_label_type_a_tl }
3935     { \l__zrefclever_ref_language_tl }
3936     \l__zrefclever_abbrev_bool
3937 \__zrefclever_get_rf_opt_bool:neeN { rangetopair } { true }
3938     { \l__zrefclever_label_type_a_tl }
3939     { \l__zrefclever_ref_language_tl }
3940     \l__zrefclever_rangetopair_bool
3941 \__zrefclever_get_rf_opt_seq:neeN { refbounds-first }
3942     { \l__zrefclever_label_type_a_tl }
3943     { \l__zrefclever_ref_language_tl }
3944     \l__zrefclever_refbounds_first_seq
3945 \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-sg }
3946     { \l__zrefclever_label_type_a_tl }
3947     { \l__zrefclever_ref_language_tl }
3948     \l__zrefclever_refbounds_first_sg_seq
3949 \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-pb }
3950     { \l__zrefclever_label_type_a_tl }
3951     { \l__zrefclever_ref_language_tl }
3952     \l__zrefclever_refbounds_first_pb_seq
3953 \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-rb }
3954     { \l__zrefclever_label_type_a_tl }
3955     { \l__zrefclever_ref_language_tl }
3956     \l__zrefclever_refbounds_first_rb_seq
3957 \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid }
3958     { \l__zrefclever_label_type_a_tl }
3959     { \l__zrefclever_ref_language_tl }
3960     \l__zrefclever_refbounds_mid_seq
3961 \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-rb }
3962     { \l__zrefclever_label_type_a_tl }
3963     { \l__zrefclever_ref_language_tl }
3964     \l__zrefclever_refbounds_mid_rb_seq
3965 \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-re }
3966     { \l__zrefclever_label_type_a_tl }
3967     { \l__zrefclever_ref_language_tl }
3968     \l__zrefclever_refbounds_mid_re_seq
3969 \__zrefclever_get_rf_opt_seq:neeN { refbounds-last }
3970     { \l__zrefclever_label_type_a_tl }
3971     { \l__zrefclever_ref_language_tl }
3972     \l__zrefclever_refbounds_last_seq
3973 \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-pe }
3974     { \l__zrefclever_label_type_a_tl }
3975     { \l__zrefclever_ref_language_tl }
3976     \l__zrefclever_refbounds_last_pe_seq
3977 \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-re }
3978     { \l__zrefclever_label_type_a_tl }
3979     { \l__zrefclever_ref_language_tl }
3980     \l__zrefclever_refbounds_last_re_seq

```

```

3981     }
3982     % Here we send this to a couple of auxiliary functions.
3983     \bool_if:NTF \l__zrefclever_last_of_type_bool
3984     % There exists no next label of the same type as the current.
3985     { \__zrefclever_typeset_refs_last_of_type: }
3986     % There exists a next label of the same type as the current.
3987     { \__zrefclever_typeset_refs_not_last_of_type: }
3988   }
3989 }

```

(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.

```

3990 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3991 {
3992   % Process the current label to the current queue.
3993   \int_case:nnF { \l__zrefclever_label_count_int }
3994   {
3995     % It is the last label of its type, but also the first one, and that's
3996     % what matters here: just store it.
3997     % Test: `zc-typeset01.lvt': "Last of type: single"
3998     { 0 }
3999     {
4000       \tl_set:NV \l__zrefclever_type_first_label_tl
4001         \l__zrefclever_label_a_tl
4002       \tl_set:NV \l__zrefclever_type_first_label_type_tl
4003         \l__zrefclever_label_type_a_tl
4004       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4005         \l__zrefclever_refbounds_first_sg_seq
4006       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4007     }
4008     % The last is the second: we have a pair (if not repeated).
4009     % Test: `zc-typeset01.lvt': "Last of type: pair"
4010     { 1 }
4011     {
4012       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4013       {
4014         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4015           \l__zrefclever_refbounds_first_sg_seq
4016         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4017       }
4018       {
4019         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4020         {
4021           \exp_not:V \l__zrefclever_pairsep_tl

```

```

4022         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4023         \l__zrefclever_refbounds_last_pe_seq
4024     }
4025     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4026     \l__zrefclever_refbounds_first_pb_seq
4027     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4028 }
4029 }
4030 }
4031 % Last is third or more of its type: without repetition, we'd have the
4032 % last element on a list, but control for possible repetition.
4033 {
4034     \int_case:nnF { \l__zrefclever_range_count_int }
4035     {
4036         % There was no range going on.
4037         % Test: `zc-typeset01.lvt': "Last of type: not range"
4038         { 0 }
4039         {
4040             \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4041             {
4042                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4043                 {
4044                     \exp_not:V \l__zrefclever_pairsep_tl
4045                     \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4046                     \l__zrefclever_refbounds_last_pe_seq
4047                 }
4048             }
4049             {
4050                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4051                 {
4052                     \exp_not:V \l__zrefclever_lastsep_tl
4053                     \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4054                     \l__zrefclever_refbounds_last_seq
4055                 }
4056             }
4057         }
4058         % Last in the range is also the second in it.
4059         % Test: `zc-typeset01.lvt': "Last of type: pair in sequence"
4060         { 1 }
4061         {
4062             \int_compare:nNnTF
4063             { \l__zrefclever_range_same_count_int } = { 1 }
4064             {
4065                 % We know `range_beg_is_first_bool' is false, since this is
4066                 % the second element in the range, but the third or more in
4067                 % the type list.
4068                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4069                 {
4070                     \exp_not:V \l__zrefclever_pairsep_tl
4071                     \_zrefclever_get_ref:VN
4072                     \l__zrefclever_range_beg_label_tl
4073                     \l__zrefclever_refbounds_last_pe_seq
4074                 }
4075                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq

```

```

4076         \l__zrefclever_refbounds_first_pb_seq
4077     \bool_set_true:N
4078         \l__zrefclever_type_first_refbounds_set_bool
4079     }
4080     {
4081     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4082         {
4083         \exp_not:V \l__zrefclever_listsep_tl
4084         \__zrefclever_get_ref:VN
4085             \l__zrefclever_range_beg_label_tl
4086             \l__zrefclever_refbounds_mid_seq
4087         \exp_not:V \l__zrefclever_lastsep_tl
4088         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4089             \l__zrefclever_refbounds_last_seq
4090         }
4091     }
4092 }
4093 }
4094 % Last in the range is third or more in it.
4095 {
4096     \int_case:nnF
4097     {
4098         \l__zrefclever_range_count_int -
4099         \l__zrefclever_range_same_count_int
4100     }
4101     {
4102         % Repetition, not a range.
4103         % Test: `zc-typeset01.lvt': "Last of type: range to one"
4104         { 0 }
4105         {
4106             % If `range_beg_is_first_bool' is true, it means it was also
4107             % the first of the type, and hence its typesetting was
4108             % already handled, and we just have to set refbounds.
4109             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4110             {
4111                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4112                 \l__zrefclever_refbounds_first_sg_seq
4113                 \bool_set_true:N
4114                 \l__zrefclever_type_first_refbounds_set_bool
4115             }
4116             {
4117                 \int_compare:nNnTF
4118                 { \l__zrefclever_ref_count_int } < { 2 }
4119                 {
4120                     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4121                     {
4122                         \exp_not:V \l__zrefclever_pairsep_tl
4123                         \__zrefclever_get_ref:VN
4124                         \l__zrefclever_range_beg_label_tl
4125                         \l__zrefclever_refbounds_last_pe_seq
4126                     }
4127                 }
4128                 {
4129                     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl

```

```

4130         {
4131             \exp_not:V \l__zrefclever_lastsep_tl
4132             \__zrefclever_get_ref:VN
4133             \l__zrefclever_range_beg_label_tl
4134             \l__zrefclever_refbounds_last_seq
4135         }
4136     }
4137 }
4138 }
4139 % A `range', but with no skipped value, treat as pair if range
4140 % started with first of type, otherwise as list.
4141 % Test: `zc-typeset01.lvt': "Last of type: range to pair"
4142 { 1 }
4143 {
4144     % Ditto.
4145     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4146     {
4147         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4148         \l__zrefclever_refbounds_first_pb_seq
4149         \bool_set_true:N
4150         \l__zrefclever_type_first_refbounds_set_bool
4151         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4152         {
4153             \exp_not:V \l__zrefclever_pairsep_tl
4154             \__zrefclever_get_ref:VN \l__zrefclever_label_a_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_listsep_tl
4162             \__zrefclever_get_ref:VN
4163             \l__zrefclever_range_beg_label_tl
4164             \l__zrefclever_refbounds_mid_seq
4165         }
4166         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4167         {
4168             \exp_not:V \l__zrefclever_lastsep_tl
4169             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4170             \l__zrefclever_refbounds_last_seq
4171         }
4172     }
4173 }
4174 }
4175 {
4176     % An actual range.
4177     % Test: `zc-typeset01.lvt': "Last of type: range"
4178     % Ditto.
4179     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4180     {
4181         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4182         \l__zrefclever_refbounds_first_rb_seq
4183         \bool_set_true:N

```

```

4184         \l__zrefclever_type_first_refbounds_set_bool
4185     }
4186     {
4187     \int_compare:nNnTF
4188     { \l__zrefclever_ref_count_int } < { 2 }
4189     {
4190         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4191         {
4192             \exp_not:V \l__zrefclever_pairsep_tl
4193             \__zrefclever_get_ref:VN
4194             \l__zrefclever_range_beg_label_tl
4195             \l__zrefclever_refbounds_mid_rb_seq
4196         }
4197         \seq_set_eq:NN
4198         \l__zrefclever_type_first_refbounds_seq
4199         \l__zrefclever_refbounds_first_pb_seq
4200         \bool_set_true:N
4201         \l__zrefclever_type_first_refbounds_set_bool
4202     }
4203     {
4204         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4205         {
4206             \exp_not:V \l__zrefclever_lastsep_tl
4207             \__zrefclever_get_ref:VN
4208             \l__zrefclever_range_beg_label_tl
4209             \l__zrefclever_refbounds_mid_rb_seq
4210         }
4211     }
4212 }
4213 \bool_lazy_and:nnTF
4214 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4215 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4216 {
4217     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4218     \l__zrefclever_range_beg_label_tl
4219     \l__zrefclever_label_a_tl
4220     \l__zrefclever_range_end_ref_tl
4221     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4222     {
4223         \exp_not:V \l__zrefclever_rangesep_tl
4224         \__zrefclever_get_ref_endrange:VVN
4225         \l__zrefclever_label_a_tl
4226         \l__zrefclever_range_end_ref_tl
4227         \l__zrefclever_refbounds_last_re_seq
4228     }
4229 }
4230 {
4231     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4232     {
4233         \exp_not:V \l__zrefclever_rangesep_tl
4234         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4235         \l__zrefclever_refbounds_last_re_seq
4236     }
4237 }

```



```

4238     }
4239   }
4240 }
4241 % Handle "range" option. The idea is simple: if the queue is not empty,
4242 % we replace it with the end of the range (or pair). We can still
4243 % retrieve the end of the range from `label_a' since we know to be
4244 % processing the last label of its type at this point.
4245 \bool_if:NT \l__zrefclever_typeset_range_bool
4246 {
4247   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4248   {
4249     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4250     { }
4251     {
4252       \msg_warning:nne { zref-clever } { single-element-range }
4253       { \l__zrefclever_type_first_label_type_tl }
4254     }
4255   }
4256   {
4257     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4258     \bool_if:NT \l__zrefclever_rangetopair_bool
4259     {
4260       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4261       { }
4262       {
4263         \__zrefclever_labels_in_sequence:nn
4264         { \l__zrefclever_type_first_label_tl }
4265         { \l__zrefclever_label_a_tl }
4266       }
4267     }
4268     % Test: `zc-typeset01.lvt': "Last of type: option range"
4269     % Test: `zc-typeset01.lvt': "Last of type: option range to pair"
4270     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4271     {
4272       \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4273       {
4274         \exp_not:V \l__zrefclever_pairsep_tl
4275         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4276         \l__zrefclever_refbounds_last_pe_seq
4277       }
4278       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4279       \l__zrefclever_refbounds_first_pb_seq
4280       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4281     }
4282     {
4283       \bool_lazy_and:nnTF
4284       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4285       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4286       {
4287         % We must get `type_first_label_tl' instead of
4288         % `range_beg_label_tl' here, since it is not necessary
4289         % that the first of type was actually starting a range for
4290         % the `range' option to be used.
4291         \use:c { \l__zrefclever_endrangefunc_tl :VVN }

```

```

4292         \l__zrefclever_type_first_label_tl
4293         \l__zrefclever_label_a_tl
4294         \l__zrefclever_range_end_ref_tl
4295     \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4296     {
4297         \exp_not:V \l__zrefclever_rangeseq_tl
4298         \__zrefclever_get_ref_endrange:VVN
4299         \l__zrefclever_label_a_tl
4300         \l__zrefclever_range_end_ref_tl
4301         \l__zrefclever_refbounds_last_re_seq
4302     }
4303 }
4304 {
4305     \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4306     {
4307         \exp_not:V \l__zrefclever_rangeseq_tl
4308         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4309         \l__zrefclever_refbounds_last_re_seq
4310     }
4311 }
4312 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4313 \l__zrefclever_refbounds_first_rb_seq
4314 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4315 }
4316 }
4317 }
4318 % If none of the special cases for the first of type refbounds have been
4319 % set, do it.
4320 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4321 {
4322     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4323     \l__zrefclever_refbounds_first_seq
4324 }
4325 % Now that the type block is finished, we can add the name and the first
4326 % ref to the queue. Also, if "typeset" option is not "both", handle it
4327 % here as well.
4328 \__zrefclever_type_name_setup:
4329 \bool_if:nTF
4330 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4331 {
4332     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4333     { \__zrefclever_get_ref_first: }
4334 }
4335 {
4336     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4337     {
4338         % Test: `zc-typeset01.lvt': "Last of type: option typeset ref"
4339         \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4340         {
4341             \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4342             \l__zrefclever_type_first_refbounds_seq
4343         }
4344     }
4345     {

```

```

4346 \bool_if:NTF \l__zrefclever_typeset_name_bool
4347 {
4348   % Test: `zc-typeset01.lvt': "Last of type: option typeset name"
4349   \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4350   {
4351     \bool_if:NTF \l__zrefclever_name_in_link_bool
4352     {
4353       \exp_not:N \group_begin:
4354       \exp_not:V \l__zrefclever_namefont_tl
4355       \__zrefclever_hyperlink:nnn
4356       {
4357         \__zrefclever_extract_url_unexp:V
4358         \l__zrefclever_type_first_label_tl
4359       }
4360       {
4361         \__zrefclever_extract_unexp:Vnn
4362         \l__zrefclever_type_first_label_tl
4363         { anchor } { }
4364       }
4365       { \exp_not:V \l__zrefclever_type_name_tl }
4366     \exp_not:N \group_end:
4367   }
4368   {
4369     \exp_not:N \group_begin:
4370     \exp_not:V \l__zrefclever_namefont_tl
4371     \exp_not:V \l__zrefclever_type_name_tl
4372     \exp_not:N \group_end:
4373   }
4374 }
4375 }
4376 {
4377   % Logically, this case would correspond to "typeset=none", but
4378   % it should not occur, given that the options are set up to
4379   % typeset either "ref" or "name". Still, leave here a
4380   % sensible fallback, equal to the behavior of "both".
4381   % Test: `zc-typeset01.lvt': "Last of type: option typeset none"
4382   \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4383   { \__zrefclever_get_ref_first: }
4384 }
4385 }
4386 }
4387 % Typeset the previous type block, if there is one.
4388 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4389 {
4390   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4391   { \l__zrefclever_tlistsep_tl }
4392   \l__zrefclever_typeset_queue_prev_tl
4393 }
4394 % Extra log for testing.
4395 \bool_if:NT \l__zrefclever_verbose_testing_bool
4396 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4397 % Wrap up loop, or prepare for next iteration.
4398 \bool_if:NTF \l__zrefclever_typeset_last_bool
4399 {

```

```

4400 % We are finishing, typeset the current queue.
4401 \int_case:nnF { \l__zrefclever_type_count_int }
4402 {
4403   % Single type.
4404   % Test: `zc-typeset01.lvt': "Last of type: single type"
4405   { 0 }
4406   { \l__zrefclever_typeset_queue_curr_tl }
4407   % Pair of types.
4408   % Test: `zc-typeset01.lvt': "Last of type: pair of types"
4409   { 1 }
4410   {
4411     \l__zrefclever_tpairsep_tl
4412     \l__zrefclever_typeset_queue_curr_tl
4413   }
4414 }
4415 {
4416   % Last in list of types.
4417   % Test: `zc-typeset01.lvt': "Last of type: list of types"
4418   \l__zrefclever_tlastsep_tl
4419   \l__zrefclever_typeset_queue_curr_tl
4420 }
4421 % And nudge in case of multitype reference.
4422 \bool_lazy_all:nT
4423 {
4424   { \l__zrefclever_nudge_enabled_bool }
4425   { \l__zrefclever_nudge_multitype_bool }
4426   { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4427 }
4428 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4429 }
4430 {
4431   % There are further labels, set variables for next iteration.
4432   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4433     \l__zrefclever_typeset_queue_curr_tl
4434   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4435   \tl_clear:N \l__zrefclever_type_first_label_tl
4436   \tl_clear:N \l__zrefclever_type_first_label_type_tl
4437   \tl_clear:N \l__zrefclever_range_beg_label_tl
4438   \tl_clear:N \l__zrefclever_range_end_ref_tl
4439   \int_zero:N \l__zrefclever_label_count_int
4440   \int_zero:N \l__zrefclever_ref_count_int
4441   \int_incr:N \l__zrefclever_type_count_int
4442   \int_zero:N \l__zrefclever_range_count_int
4443   \int_zero:N \l__zrefclever_range_same_count_int
4444   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4445   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4446 }
4447 }

```

*(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.

```

4448 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4449 {

```

```

4450 % Signal if next label may form a range with the current one (only
4451 % considered if compression is enabled in the first place).
4452 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4453 \bool_set_false:N \l__zrefclever_next_is_same_bool
4454 \bool_if:NT \l__zrefclever_typeset_compress_bool
4455 {
4456   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4457   { }
4458   {
4459     \__zrefclever_labels_in_sequence:nn
4460     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
4461   }
4462 }
4463 % Process the current label to the current queue.
4464 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4465 {
4466   % Current label is the first of its type (also not the last, but it
4467   % doesn't matter here): just store the label.
4468   \tl_set:NV \l__zrefclever_type_first_label_tl
4469   \l__zrefclever_label_a_tl
4470   \tl_set:NV \l__zrefclever_type_first_label_type_tl
4471   \l__zrefclever_label_type_a_tl
4472   \int_incr:N \l__zrefclever_ref_count_int
4473   % If the next label may be part of a range, signal it (we deal with it
4474   % as the "first", and must do it there, to handle hyperlinking), but
4475   % also step the range counters.
4476   % Test: `zc-typeset01.lvt': "Not last of type: first is range"
4477   \bool_if:NT \l__zrefclever_next_maybe_range_bool
4478   {
4479     \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4480     \tl_set:NV \l__zrefclever_range_beg_label_tl
4481     \l__zrefclever_label_a_tl
4482     \tl_clear:N \l__zrefclever_range_end_ref_tl
4483     \int_incr:N \l__zrefclever_range_count_int
4484     \bool_if:NT \l__zrefclever_next_is_same_bool
4485     { \int_incr:N \l__zrefclever_range_same_count_int }
4486   }
4487 }
4488 {
4489   % Current label is neither the first (nor the last) of its type.
4490   \bool_if:NnTF \l__zrefclever_next_maybe_range_bool
4491   {
4492     % Starting, or continuing a range.
4493     \int_compare:nNnTF
4494     { \l__zrefclever_range_count_int } = { 0 }
4495     {
4496       % There was no range going, we are starting one.
4497       \tl_set:NV \l__zrefclever_range_beg_label_tl
4498       \l__zrefclever_label_a_tl
4499       \tl_clear:N \l__zrefclever_range_end_ref_tl
4500       \int_incr:N \l__zrefclever_range_count_int
4501       \bool_if:NT \l__zrefclever_next_is_same_bool
4502       { \int_incr:N \l__zrefclever_range_same_count_int }
4503     }

```

```

4504     {
4505     % Second or more in the range, but not the last.
4506     \int_incr:N \l__zrefclever_range_count_int
4507     \bool_if:NT \l__zrefclever_next_is_same_bool
4508     { \int_incr:N \l__zrefclever_range_same_count_int }
4509     }
4510   }
4511   {
4512   % Next element is not in sequence: there was no range, or we are
4513   % closing one.
4514   \int_case:nNF { \l__zrefclever_range_count_int }
4515   {
4516     % There was no range going on.
4517     % Test: `zc-typeset01.lvt': "Not last of type: no range"
4518     { 0 }
4519     {
4520       \int_incr:N \l__zrefclever_ref_count_int
4521       \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4522       {
4523         \exp_not:V \l__zrefclever_listsep_tl
4524         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4525         \l__zrefclever_refbounds_mid_seq
4526       }
4527     }
4528     % Last is second in the range: if `range_same_count' is also
4529     % `1', it's a repetition (drop it), otherwise, it's a "pair
4530     % within a list", treat as list.
4531     % Test: `zc-typeset01.lvt': "Not last of type: range pair to one"
4532     % Test: `zc-typeset01.lvt': "Not last of type: range pair"
4533     { 1 }
4534     {
4535       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4536       {
4537         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4538         \l__zrefclever_refbounds_first_seq
4539         \bool_set_true:N
4540         \l__zrefclever_type_first_refbounds_set_bool
4541       }
4542       {
4543         \int_incr:N \l__zrefclever_ref_count_int
4544         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4545         {
4546           \exp_not:V \l__zrefclever_listsep_tl
4547           \__zrefclever_get_ref:VN
4548           \l__zrefclever_range_beg_label_tl
4549           \l__zrefclever_refbounds_mid_seq
4550         }
4551       }
4552       \int_compare:nNnF
4553       { \l__zrefclever_range_same_count_int } = { 1 }
4554       {
4555         \int_incr:N \l__zrefclever_ref_count_int
4556         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4557         {

```

```

4558         \exp_not:V \l__zrefclever_listsep_tl
4559         \__zrefclever_get_ref:VN
4560         \l__zrefclever_label_a_tl
4561         \l__zrefclever_refbounds_mid_seq
4562     }
4563 }
4564 }
4565 }
4566 {
4567     % Last is third or more in the range: if `range_count' and
4568     % `range_same_count' are the same, its a repetition (drop it),
4569     % if they differ by `1', its a list, if they differ by more,
4570     % it is a real range.
4571     \int_case:nnF
4572     {
4573         \l__zrefclever_range_count_int -
4574         \l__zrefclever_range_same_count_int
4575     }
4576     {
4577         % Test: `zc-typeset01.lvt': "Not last of type: range to one"
4578         { 0 }
4579         {
4580             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4581             {
4582                 \seq_set_eq:NN
4583                 \l__zrefclever_type_first_refbounds_seq
4584                 \l__zrefclever_refbounds_first_seq
4585                 \bool_set_true:N
4586                 \l__zrefclever_type_first_refbounds_set_bool
4587             }
4588             {
4589                 \int_incr:N \l__zrefclever_ref_count_int
4590                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4591                 {
4592                     \exp_not:V \l__zrefclever_listsep_tl
4593                     \__zrefclever_get_ref:VN
4594                     \l__zrefclever_range_beg_label_tl
4595                     \l__zrefclever_refbounds_mid_seq
4596                 }
4597             }
4598         }
4599         % Test: `zc-typeset01.lvt': "Not last of type: range to pair"
4600         { 1 }
4601         {
4602             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4603             {
4604                 \seq_set_eq:NN
4605                 \l__zrefclever_type_first_refbounds_seq
4606                 \l__zrefclever_refbounds_first_seq
4607                 \bool_set_true:N
4608                 \l__zrefclever_type_first_refbounds_set_bool
4609             }
4610             {
4611                 \int_incr:N \l__zrefclever_ref_count_int

```

```

4612         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4613         {
4614             \exp_not:V \l__zrefclever_listsep_tl
4615             \__zrefclever_get_ref:VN
4616             \l__zrefclever_range_beg_label_tl
4617             \l__zrefclever_refbounds_mid_seq
4618         }
4619     }
4620     \int_incr:N \l__zrefclever_ref_count_int
4621     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4622     {
4623         \exp_not:V \l__zrefclever_listsep_tl
4624         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4625         \l__zrefclever_refbounds_mid_seq
4626     }
4627 }
4628 }
4629 {
4630 % Test: `zc-typeset01.lvt': "Not last of type: range"
4631 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4632 {
4633     \seq_set_eq:NN
4634     \l__zrefclever_type_first_refbounds_seq
4635     \l__zrefclever_refbounds_first_rb_seq
4636     \bool_set_true:N
4637     \l__zrefclever_type_first_refbounds_set_bool
4638 }
4639 {
4640     \int_incr:N \l__zrefclever_ref_count_int
4641     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4642     {
4643         \exp_not:V \l__zrefclever_listsep_tl
4644         \__zrefclever_get_ref:VN
4645         \l__zrefclever_range_beg_label_tl
4646         \l__zrefclever_refbounds_mid_rb_seq
4647     }
4648 }
4649 % For the purposes of the serial comma, and thus for the
4650 % distinction of `lastsep' and `pairsep', a "range" counts
4651 % as one. Since `range_beg' has already been counted
4652 % (here or with the first of type), we refrain from
4653 % incrementing `ref_count_int'.
4654 \bool_lazy_and:nnTF
4655 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4656 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4657 {
4658     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4659     \l__zrefclever_range_beg_label_tl
4660     \l__zrefclever_label_a_tl
4661     \l__zrefclever_range_end_ref_tl
4662     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4663     {
4664         \exp_not:V \l__zrefclever_rangesep_tl
4665         \__zrefclever_get_ref_endrange:VVN

```



```

4666         \l__zrefclever_label_a_tl
4667         \l__zrefclever_range_end_ref_tl
4668         \l__zrefclever_refbounds_mid_re_seq
4669     }
4670 }
4671 {
4672     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4673     {
4674         \exp_not:V \l__zrefclever_rangeseq_tl
4675         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4676         \l__zrefclever_refbounds_mid_re_seq
4677     }
4678 }
4679 }
4680 }
4681 % We just closed a range, reset `range_beg_is_first' in case a
4682 % second range for the same type occurs, in which case its
4683 % `range_beg' will no longer be `first'.
4684 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4685 % Reset counters.
4686 \int_zero:N \l__zrefclever_range_count_int
4687 \int_zero:N \l__zrefclever_range_same_count_int
4688 }
4689 }
4690 % Step label counter for next iteration.
4691 \int_incr:N \l__zrefclever_label_count_int
4692 }

```

(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 `e` 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.

```
4693 \cs_new_protected:Npn \__zrefclever_ref_default:
4694   { \zref@default }
4695 \cs_new_protected:Npn \__zrefclever_name_default:
4696   { \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>}
```

```
4697 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4698   {
4699     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4700     {
4701       \bool_if:nTF
4702         {
4703           \l__zrefclever_hyperlink_bool &&
4704           ! \l__zrefclever_link_star_bool
4705         }
4706         {
4707           \seq_item:Nn #2 { 1 }
4708           \__zrefclever_hyperlink:nnn
4709             { \__zrefclever_extract_url_unexp:n {#1} }
4710             { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4711             {
4712               \seq_item:Nn #2 { 2 }
4713               \exp_not:N \group_begin:
4714                 \exp_not:V \l__zrefclever_reffont_tl
4715                 \__zrefclever_extract_unexp:nv {#1}
4716                   { \l__zrefclever_ref_property_tl } { }
4717               \exp_not:N \group_end:
4718               \seq_item:Nn #2 { 3 }
4719             }
4720           \seq_item:Nn #2 { 4 }
4721         }
4722         {
4723           \seq_item:Nn #2 { 1 }
4724           \seq_item:Nn #2 { 2 }
4725           \exp_not:N \group_begin:
4726             \exp_not:V \l__zrefclever_reffont_tl
4727             \__zrefclever_extract_unexp:nv {#1}
4728               { \l__zrefclever_ref_property_tl } { }
4729           \exp_not:N \group_end:
4730           \seq_item:Nn #2 { 3 }
4731           \seq_item:Nn #2 { 4 }
4732         }
4733     }
```

```

4733     }
4734     { \_zrefclever_ref_default: }
4735   }
4736 \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>}
4737 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3
4738 {
4739   \str_if_eq:nnTF {#2} {zc@missingproperty} {
4740     { \_zrefclever_ref_default: }
4741     {
4742       \bool_if:nTF
4743         {
4744           \l_zrefclever_hyperlink_bool &&
4745           ! \l_zrefclever_link_star_bool
4746         }
4747         {
4748           \seq_item:Nn #3 { 1 }
4749           \_zrefclever_hyperlink:nnn
4750             { \_zrefclever_extract_url_unexp:n {#1} }
4751             { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4752             {
4753               \seq_item:Nn #3 { 2 }
4754               \exp_not:N \group_begin:
4755                 \exp_not:V \l_zrefclever_reffont_tl
4756                 \exp_not:n {#2}
4757               \exp_not:N \group_end:
4758               \seq_item:Nn #3 { 3 }
4759             }
4760           \seq_item:Nn #3 { 4 }
4761         }
4762         {
4763           \seq_item:Nn #3 { 1 }
4764           \seq_item:Nn #3 { 2 }
4765           \exp_not:N \group_begin:
4766             \exp_not:V \l_zrefclever_reffont_tl
4767             \exp_not:n {#2}
4768           \exp_not:N \group_end:
4769           \seq_item:Nn #3 { 3 }
4770           \seq_item:Nn #3 { 4 }
4771         }
4772       }
4773     }
4774 \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.

```

4775 \cs_new:Npn \__zrefclever_get_ref_first:
4776 {
4777   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4778   { \__zrefclever_ref_default: }
4779   {
4780     \bool_if:NTF \l__zrefclever_name_in_link_bool
4781     {
4782       \zref@ifrefcontainsprop
4783       { \l__zrefclever_type_first_label_tl }
4784       { \l__zrefclever_ref_property_tl }
4785       {
4786         \__zrefclever_hyperlink:nnn
4787         {
4788           \__zrefclever_extract_url_unexp:V
4789           \l__zrefclever_type_first_label_tl
4790         }
4791         {
4792           \__zrefclever_extract_unexp:Vnn
4793           \l__zrefclever_type_first_label_tl { anchor } { }
4794         }
4795         {
4796           \exp_not:N \group_begin:
4797           \exp_not:V \l__zrefclever_namefont_tl
4798           \exp_not:V \l__zrefclever_type_name_tl
4799           \exp_not:N \group_end:
4800           \exp_not:V \l__zrefclever_namesep_tl
4801           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4802           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4803           \exp_not:N \group_begin:
4804           \exp_not:V \l__zrefclever_reffont_tl
4805           \__zrefclever_extract_unexp:Vvn
4806           \l__zrefclever_type_first_label_tl
4807           { \l__zrefclever_ref_property_tl } { }
4808           \exp_not:N \group_end:
4809           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4810         }
4811         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4812       }
4813     }
4814     \exp_not:N \group_begin:
4815     \exp_not:V \l__zrefclever_namefont_tl
4816     \exp_not:V \l__zrefclever_type_name_tl
4817     \exp_not:N \group_end:
4818     \exp_not:V \l__zrefclever_namesep_tl
4819     \__zrefclever_ref_default:
4820   }
4821 }
4822 {
4823   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4824   {
4825     \__zrefclever_name_default:

```

```

4826         \exp_not:V \l__zrefclever_namesep_tl
4827     }
4828     {
4829         \exp_not:N \group_begin:
4830         \exp_not:V \l__zrefclever_namefont_tl
4831         \exp_not:V \l__zrefclever_type_name_tl
4832         \exp_not:N \group_end:
4833         \tl_if_empty:NF \l__zrefclever_type_name_tl
4834         { \exp_not:V \l__zrefclever_namesep_tl }
4835     }
4836 \zref@ifrefcontainsprop
4837 { \l__zrefclever_type_first_label_tl }
4838 { \l__zrefclever_ref_property_tl }
4839 {
4840     \bool_if:nTF
4841     {
4842         \l__zrefclever_hyperlink_bool &&
4843         ! \l__zrefclever_link_star_bool
4844     }
4845     {
4846         \seq_item:Nn
4847         \l__zrefclever_type_first_refbounds_seq { 1 }
4848         \__zrefclever_hyperlink:nnn
4849         {
4850             \__zrefclever_extract_url_unexp:V
4851             \l__zrefclever_type_first_label_tl
4852         }
4853         {
4854             \__zrefclever_extract_unexp:Vnn
4855             \l__zrefclever_type_first_label_tl { anchor } { }
4856         }
4857     }
4858     {
4859         \seq_item:Nn
4860         \l__zrefclever_type_first_refbounds_seq { 2 }
4861         \exp_not:N \group_begin:
4862         \exp_not:V \l__zrefclever_reffont_tl
4863         \__zrefclever_extract_unexp:Vvn
4864         \l__zrefclever_type_first_label_tl
4865         { \l__zrefclever_ref_property_tl } { }
4866         \exp_not:N \group_end:
4867         \seq_item:Nn
4868         \l__zrefclever_type_first_refbounds_seq { 3 }
4869     }
4870     \seq_item:Nn
4871     \l__zrefclever_type_first_refbounds_seq { 4 }
4872 }
4873 {
4874     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4875     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4876     \exp_not:N \group_begin:
4877     \exp_not:V \l__zrefclever_reffont_tl
4878     \__zrefclever_extract_unexp:Vvn
4879     \l__zrefclever_type_first_label_tl
4880     { \l__zrefclever_ref_property_tl } { }

```

```

4880         \exp_not:N \group_end:
4881         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4882         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4883     }
4884 }
4885 { \__zrefclever_ref_default: }
4886 }
4887 }
4888 }

```

(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, \l\_\_zrefclever\_name\_in\_link\_bool, and \l\_\_zrefclever\_type\_name\_missing\_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.

```

4889 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4890 {
4891     \bool_if:nTF
4892     { \l__zrefclever_typeset_ref_bool && ! \l__zrefclever_typeset_name_bool }
4893     {
4894         % `typeset=ref' / `noname' option
4895         % Probably redundant, since in this case the type name is not being
4896         % typeset. But, for completeness sake:
4897         \tl_clear:N \l__zrefclever_type_name_tl
4898         \bool_set_false:N \l__zrefclever_name_in_link_bool
4899         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4900     }
4901     {
4902         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4903         {
4904             \tl_clear:N \l__zrefclever_type_name_tl
4905             \bool_set_true:N \l__zrefclever_type_name_missing_bool
4906         }
4907         {
4908             \tl_if_eq:NnTF
4909             \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4910             {
4911                 \tl_clear:N \l__zrefclever_type_name_tl
4912                 \bool_set_true:N \l__zrefclever_type_name_missing_bool
4913             }
4914             {
4915                 % Determine whether we should use capitalization,
4916                 % abbreviation, and plural.
4917                 \bool_lazy_or:nnTF

```

```

4918 { \l__zrefclever_cap_bool }
4919 {
4920   \l__zrefclever_capfirst_bool &&
4921   \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4922 }
4923 { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4924 { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4925 % If the queue is empty, we have a singular, otherwise,
4926 % plural.
4927 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4928 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4929 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4930 \bool_lazy_and:nnTF
4931 { \l__zrefclever_abbrev_bool }
4932 {
4933   ! \int_compare_p:nNn
4934     { \l__zrefclever_type_count_int } = { 0 } ||
4935   ! \l__zrefclever_noabbrev_first_bool
4936 }
4937 {
4938   \tl_set:NV \l__zrefclever_name_format_fallback_tl
4939     \l__zrefclever_name_format_tl
4940   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4941 }
4942 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4943 % Handle number and gender nudges.
4944 % Note that these nudges get disabled for `typeset=ref' /
4945 % `noname' option, but in this case they are not really
4946 % meaningful anyway.
4947 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4948 {
4949   \bool_if:NTF \l__zrefclever_nudge_singular_bool
4950   {
4951     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4952     {
4953       \msg_warning:nne { zref-clever }
4954       { nudge-plural-when-sg }
4955       { \l__zrefclever_type_first_label_type_tl }
4956     }
4957   }
4958   {
4959     \bool_lazy_all:nT
4960     {
4961       { \l__zrefclever_nudge_comptosing_bool }
4962       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4963       {
4964         \int_compare_p:nNn
4965           { \l__zrefclever_label_count_int } > { 0 }
4966       }
4967     }
4968     {
4969       \msg_warning:nne { zref-clever }
4970       { nudge-comptosing }
4971       { \l__zrefclever_type_first_label_type_tl }

```

```

4972     }
4973   }
4974   \bool_lazy_and:nnT
4975   { \l__zrefclever_nudge_gender_bool }
4976   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4977   {
4978     \__zrefclever_get_rf_opt_seq:neeN { gender }
4979     { \l__zrefclever_type_first_label_type_tl }
4980     { \l__zrefclever_ref_language_tl }
4981     \l__zrefclever_type_name_gender_seq
4982     \seq_if_in:NVF
4983     \l__zrefclever_type_name_gender_seq
4984     \l__zrefclever_ref_gender_tl
4985     {
4986       \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4987       {
4988         \msg_warning:nneee { zref-clever }
4989         { nudge-gender-not-declared-for-type }
4990         { \l__zrefclever_ref_gender_tl }
4991         { \l__zrefclever_type_first_label_type_tl }
4992         { \l__zrefclever_ref_language_tl }
4993       }
4994       {
4995         \msg_warning:nneeee { zref-clever }
4996         { nudge-gender-mismatch }
4997         { \l__zrefclever_type_first_label_type_tl }
4998         { \l__zrefclever_ref_gender_tl }
4999         {
5000           \seq_use:Nn
5001           \l__zrefclever_type_name_gender_seq { ,~ }
5002         }
5003         { \l__zrefclever_ref_language_tl }
5004       }
5005     }
5006   }
5007 }
5008 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5009 {
5010   \__zrefclever_opt_tl_get:cNF
5011   {
5012     \__zrefclever_opt_varname_type:een
5013     { \l__zrefclever_type_first_label_type_tl }
5014     { \l__zrefclever_name_format_tl }
5015     { tl }
5016   }
5017   \l__zrefclever_type_name_tl
5018   {
5019     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5020     {
5021       \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5022       \tl_put_left:NV \l__zrefclever_name_format_tl
5023       \l__zrefclever_ref_decl_case_tl
5024     }
5025     \__zrefclever_opt_tl_get:cNF

```



```

5026         {
5027             \__zrefclever_opt_varname_lang_type:eeen
5028             { \l__zrefclever_ref_language_tl }
5029             { \l__zrefclever_type_first_label_type_tl }
5030             { \l__zrefclever_name_format_tl }
5031             { tl }
5032         }
5033         \l__zrefclever_type_name_tl
5034         {
5035             \tl_clear:N \l__zrefclever_type_name_tl
5036             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5037             \msg_warning:nnee { zref-clever } { missing-name }
5038             { \l__zrefclever_name_format_tl }
5039             { \l__zrefclever_type_first_label_type_tl }
5040         }
5041     }
5042 }
5043 {
5044     \__zrefclever_opt_tl_get:cNF
5045     {
5046         \__zrefclever_opt_varname_type:eeen
5047         { \l__zrefclever_type_first_label_type_tl }
5048         { \l__zrefclever_name_format_tl }
5049         { tl }
5050     }
5051     \l__zrefclever_type_name_tl
5052     {
5053         \__zrefclever_opt_tl_get:cNF
5054         {
5055             \__zrefclever_opt_varname_type:eeen
5056             { \l__zrefclever_type_first_label_type_tl }
5057             { \l__zrefclever_name_format_fallback_tl }
5058             { tl }
5059         }
5060         \l__zrefclever_type_name_tl
5061         {
5062             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5063             {
5064                 \tl_put_left:Nn
5065                 \l__zrefclever_name_format_tl { - }
5066                 \tl_put_left:NV \l__zrefclever_name_format_tl
5067                 \l__zrefclever_ref_decl_case_tl
5068                 \tl_put_left:Nn
5069                 \l__zrefclever_name_format_fallback_tl { - }
5070                 \tl_put_left:NV
5071                 \l__zrefclever_name_format_fallback_tl
5072                 \l__zrefclever_ref_decl_case_tl
5073             }
5074             \__zrefclever_opt_tl_get:cNF
5075             {
5076                 \__zrefclever_opt_varname_lang_type:eeen
5077                 { \l__zrefclever_ref_language_tl }
5078                 { \l__zrefclever_type_first_label_type_tl }
5079                 { \l__zrefclever_name_format_tl }

```

```

5080         { t1 }
5081     }
5082     \l__zrefclever_type_name_tl
5083     {
5084         \__zrefclever_opt_tl_get:cNF
5085         {
5086             \__zrefclever_opt_varname_lang_type:eeen
5087             { \l__zrefclever_ref_language_tl }
5088             { \l__zrefclever_type_first_label_type_tl }
5089             { \l__zrefclever_name_format_fallback_tl }
5090             { t1 }
5091         }
5092         \l__zrefclever_type_name_tl
5093         {
5094             \tl_clear:N \l__zrefclever_type_name_tl
5095             \bool_set_true:N
5096             \l__zrefclever_type_name_missing_bool
5097             \msg_warning:nnee { zref-clever }
5098             { missing-name }
5099             { \l__zrefclever_name_format_tl }
5100             { \l__zrefclever_type_first_label_type_tl }
5101         }
5102     }
5103 }
5104 }
5105 }
5106 }
5107 }
5108 % Signal whether the type name is to be included in the hyperlink or
5109 % not.
5110 \bool_lazy_any:nTF
5111 {
5112     { ! \l__zrefclever_hyperlink_bool }
5113     { \l__zrefclever_link_star_bool }
5114     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5115     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5116 }
5117 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5118 {
5119     \bool_lazy_any:nTF
5120     {
5121         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5122         {
5123             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5124             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5125         }
5126         {
5127             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5128             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5129             \l__zrefclever_typeset_last_bool &&
5130             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5131         }
5132     }
5133     { \bool_set_true:N \l__zrefclever_name_in_link_bool }

```

```

5134         { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5135     }
5136 }
5137 }

```

(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>}

```

```

5138 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5139 {
5140   \tl_if_empty:nTF {#1}
5141     { \hyperlink {#2} {#3} }
5142     { \hyper@linkfile {#3} {#1} {#2} }
5143 }

```

(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 `e` expansion, `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. See documentation for `\__zrefclever_extract_unexp:nnn`.

```

5144 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5145 {
5146   \zref@ifpropundefined { urluse }
5147     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5148     {
5149       \zref@ifrefcontainsprop {#1} { urluse }
5150         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5151         { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5152     }
5153 }
5154 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

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

`\__zrefclever_labels_in_sequence:nn` Auxiliary function to `\__zrefclever_typeset_refs_not_last_of_type:`. Sets `\l__zrefclever_next_maybe_range_bool` to true if `<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:nn {<label a>} {<label b>}

```

```

5155 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5156 {
5157   \exp_args:Nee \tl_if_eq:nnT
5158     { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5159     { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }

```

```

5160 {
5161   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5162   {
5163     \exp_args:Nee \tl_if_eq:nnT
5164     { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5165     { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5166     {
5167       \int_compare:nNnTF
5168       { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5169       =
5170       { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5171       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5172       {
5173         \int_compare:nNnT
5174         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5175         =
5176         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5177         {
5178           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5179           \bool_set_true:N \l__zrefclever_next_is_same_bool
5180         }
5181       }
5182     }
5183   }
5184   {
5185     \exp_args:Nee \tl_if_eq:nnT
5186     { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5187     { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5188     {
5189       \exp_args:Nee \tl_if_eq:nnT
5190       { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5191       { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5192       {
5193         \int_compare:nNnTF
5194         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5195         =
5196         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5197         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5198         {
5199           \int_compare:nNnT
5200           { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5201           =
5202           { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5203         }

```

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.

```

5204     \exp_args:Nee \tl_if_eq:nnT
5205     {
5206       \__zrefclever_extract_unexp:nvn {#1}
5207       { \l__zrefclever_ref_property_tl } { }

```

```

5208     }
5209     {
5210     \__zrefclever_extract_unexp:nvn {#2}
5211     { l__zrefclever_ref_property_tl } { }
5212     }
5213     {
5214     \bool_set_true:N
5215     \l__zrefclever_next_maybe_range_bool
5216     \bool_set_true:N
5217     \l__zrefclever_next_is_same_bool
5218     }
5219     }
5220   }
5221 }
5222 }
5223 }
5224 }
5225 }

```

(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>}
5226 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5227 {
5228   % First attempt: general options.
5229   \__zrefclever_opt_tl_get:cNF
5230   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5231   #4
5232   {
5233     % If not found, try type specific options.
5234     \__zrefclever_opt_tl_get:cNF
5235     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5236     #4
5237     {
5238       % If not found, try type- and language-specific.
5239       \__zrefclever_opt_tl_get:cNF
5240       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5241       #4
5242       {
5243         % If not found, try language-specific default.
5244         \__zrefclever_opt_tl_get:cNF
5245         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5246         #4
5247         {
5248           % If not found, try fallback.
5249           \__zrefclever_opt_tl_get:cNF
5250           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5251           #4
5252           { \tl_clear:N #4 }

```

```

5253         }
5254     }
5255 }
5256 }
5257 }
5258 \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>}
5259 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5260 {
5261     % First attempt: general options.
5262     \__zrefclever_opt_seq_get:cNF
5263     { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5264     #4
5265     {
5266         % If not found, try type specific options.
5267         \__zrefclever_opt_seq_get:cNF
5268         { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5269         #4
5270         {
5271             % If not found, try type- and language-specific.
5272             \__zrefclever_opt_seq_get:cNF
5273             { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5274             #4
5275             {
5276                 % If not found, try language-specific default.
5277                 \__zrefclever_opt_seq_get:cNF
5278                 { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5279                 #4
5280                 {
5281                     % If not found, try fallback.
5282                     \__zrefclever_opt_seq_get:cNF
5283                     { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5284                     #4
5285                     { \seq_clear:N #4 }
5286                 }
5287             }
5288         }
5289     }
5290 }
5291 \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>}
5292 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5293 {
5294     % First attempt: general options.
5295     \__zrefclever_opt_bool_get:cNF
5296     { \__zrefclever_opt_varname_general:nn {#1} { bool } }

```

```

5297 #5
5298 {
5299 % If not found, try type specific options.
5300 \__zrefclever_opt_bool_get:cNF
5301 { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5302 #5
5303 {
5304 % If not found, try type- and language-specific.
5305 \__zrefclever_opt_bool_get:cNF
5306 { \__zrefclever_opt_varname_lang_type:nmmm {#4} {#3} {#1} { bool } }
5307 #5
5308 {
5309 % If not found, try language-specific default.
5310 \__zrefclever_opt_bool_get:cNF
5311 { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5312 #5
5313 {
5314 % If not found, try fallback.
5315 \__zrefclever_opt_bool_get:cNF
5316 { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5317 #5
5318 { \use:c { bool_set_ #2 :N } #5 }
5319 }
5320 }
5321 }
5322 }
5323 }
5324 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nmmmm { nmmmm }

```

(End of definition for `\__zrefclever_get_rf_opt_bool:nmmmm`.)

## 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.

For the record, <https://tex.stackexchange.com/a/724742> is of interest.

```

5325 \__zrefclever_compat_module:nn { appendix }
5326 {
5327   \newcounter { zc@appendix }
5328   \cs_if_exist:cTF { chapter }
5329   {
5330     \__zrefclever_zcsetup:e
5331     {
5332       counterresetby =
5333       {

```

In case someone did something like `\counterwithin{chapter}{part}`. Harmless otherwise.

```

5334         zc@appendix = \__zrefclever_counter_reset_by:n { chapter } ,
5335         chapter = zc@appendix ,
5336       } ,
5337     }
5338   }
5339   {
5340     \cs_if_exist:cT { section }
5341     {
5342       \__zrefclever_zcsetup:e
5343       {
5344         counterresetby =
5345         {
5346           zc@appendix = \__zrefclever_counter_reset_by:n { section } ,
5347           section = zc@appendix ,
5348         } ,
5349       }
5350     }
5351   }
5352 \AddToHook { cmd / appendix / before }
5353 {
5354   \setcounter { zc@appendix } { 1 }
5355   \__zrefclever_zcsetup:n
5356   {
5357     countertype =
5358     {
5359       chapter      = appendix ,
5360       section      = appendix ,
5361       subsection   = appendix ,
5362       subsubsection = appendix ,
5363       paragraph    = appendix ,
5364       subparagraph = appendix ,
5365     }
5366   }
5367 }
5368 }

```

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.

```

5369 \__zrefclever_compat_module:nm { appendices }
5370 {
5371   \__zrefclever_if_package_loaded:nT { appendix }
5372   {
5373     \AddToHook { env / appendices / begin }
5374     {

```

Technically, the `appendices` environment can be called multiple times. By default, successive calls keep track of numbering and start where the previous one left off. Which means just setting the `zc@appendix` counter to 1 is enough for things to work, since the distinction between the calls and the sorting of their respective references will depend on the underlying sectioning. `appendix`’s documentation however, provides a way to restart from A at each call (by redefining `\restoreapp` to do nothing). In this case, the references inside different calls to `appendices` get to be identical in every way, including printed form, counter value, enclosing counters, etc., despite being different. We could keep track of different calls to `appendices` by having the `zc@appendix` counter be “stepped” at each call. Doing so would mean though that `\zcref` would distinguish things which are typeset identically, granting some arguably weird results. True, the user *can* change the printed form for each `appendices` call, e.g. redefining `\thechapter`, but in this case, they are responsible for keeping track of this.

```

5375     \setcounter { zc@appendix } { 1 }
5376     \__zrefclever_zcsetup:n
5377     {
5378       countertype =
5379       {
5380         chapter      = appendix ,
5381         section      = appendix ,
5382         subsection   = appendix ,
5383         subsubsection = appendix ,
5384         paragraph    = appendix ,
5385         subparagraph = appendix ,
5386       }
5387     }
5388   }
5389   \AddToHook { env / appendices / end }
5390   { \setcounter { zc@appendix } { 0 } }
5391   \newcounter { zc@subappendix }
5392   \cs_if_exist:cTF { chapter }
5393   {
5394     \__zrefclever_zcsetup:e
5395     {
5396       counterresetby =

```

```

5397         {
5398             zc@subappendix = \__zrefclever_counter_reset_by:n { section } ,
5399             section = zc@subappendix ,
5400         } ,
5401     }
5402 }
5403 {
5404     \__zrefclever_zcsetup:e
5405     {
5406         counterresetby =
5407         {
5408             zc@subappendix = \__zrefclever_counter_reset_by:n { subsection } ,
5409             subsection = zc@subappendix ,
5410         } ,
5411     }
5412 }
5413 \AddToHook { env / subappendices / begin }
5414 {

```

The `subappendices` environment, on the other hand, appears not to support multiple calls inside the same chapter/section (the counter is reset by default). Either way, the same reasoning applies.

```

5415     \setcounter { zc@subappendix } { 1 }
5416     \__zrefclever_zcsetup:n
5417     {
5418         countertype =
5419         {
5420             section      = appendix ,
5421             subsection   = appendix ,
5422             subsubsection = appendix ,
5423             paragraph    = appendix ,
5424             subparagraph = appendix ,
5425         } ,
5426     }
5427 }
5428 \AddToHook { env / subappendices / end }
5429 { \setcounter { zc@subappendix } { 0 } }
5430 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5431 }
5432 }

```

### 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.

```
5433 \__zrefclever_compat_module:nn { memoir }
5434 {
5435   \__zrefclever_if_class_loaded:nT { memoir }
5436   {
```

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`.

```
5437   \__zrefclever_zcsetup:n
5438   {
5439     countertype =
5440     {
5441       subfigure = figure ,
5442       subtable  = table  ,
5443       poemline  = line   ,
5444     } ,
5445     counterresetby =
5446     {
5447       subfigure = figure ,
5448       subtable  = table  ,
5449     } ,
5450   }
```

Support for subcaption references.

```
5451   \zref@newprop { subcaption }
5452   { \cs_if_exist_use:c { @thesub \@capttype } }
5453   \AddToHook{ memoir/subcaption/aftercounter }
5454   { \zref@localaddprop \ZREF@mainlist { subcaption } }
```

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

```
5455   \__zrefclever_zcsetup:n
5456   {
5457     countertype =
5458     {
5459       sidefootnote = footnote ,
5460       pagenote     = endnote  ,
5461     } ,
5462   }
5463   \msg_info:nnn { zref-clever } { compat-class } { memoir }
5464 }
5465 }
```

## 9.4 amsmath

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

```
5466 \__zrefclever_compat_module:nn { amsmath }
5467 {
```

```

5468     \_zrefclever_if_package_loaded:nT { amsmath }
5469     {

```

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).

```

5470     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5471     \AddToHook { env / subequations / begin }
5472     {
5473       \_zrefclever_zcsetup:e
5474       {
5475         counterresetby =
5476         {
5477           parentequation =
5478             \_zrefclever_counter_reset_by:n { equation } ,
5479           equation = parentequation ,
5480         } ,
5481         currentcounter = parentequation ,
5482         countertype = { parentequation = equation } ,
5483       }
5484     \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5485     }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout and supposedly sets `\@currentcounter` for `\tags` (I’m not sure if it works in all environments, though. Once I tried to remove the explicit `currentcounter` setting and several labels to `\tags` ended up with type `section`. But I didn’t investigate this further). 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.

```

5486     \zref@newprop { subeq } { \alph { equation } }
5487     \clist_map_inline:nn
5488     {
5489       equation ,
5490       equation* ,
5491       align ,
5492       align* ,
5493       alignat ,

```

```

5494     alignat* ,
5495     flalign ,
5496     flalign* ,
5497     xalignat ,
5498     xalignat* ,
5499     gather ,
5500     gather* ,
5501     multiline ,
5502     multiline* ,
5503   }
5504   {
5505     \AddToHook { env / #1 / begin }
5506     {
5507       \__zrefclever_zcsetup:n { currentcounter = equation }
5508       \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5509         { \zref@localaddprop \ZREF@mainlist { subeq } }
5510     }
5511   }
5512   \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5513 }
5514 }

```

## 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.

Note that this support comes at a little cost. `showonlyrefs` works by setting a special `\MT@newlabel` for each label referenced with `\eqref`. Now, `\eqref` is a specialized reference command, only used to refer to equations, so it sets `\MT@newlabel` unconditionally on the first run. `\zcref`, on the other hand, is a general purpose reference command, used to reference labels of any type. But we wouldn't want to set `\MT@newlabel` indiscriminately for all referenced labels in the document, so we need to test for its type. Alas, the label must exist before its type can be tested, thus we cannot set `\MT@newlabel` on the first run, only on the second. In sum, since `\eqref` requires 3 runs to work, `\zcref` needs 4.

```

5515 \bool_new:N \l__zrefclever_mathtools_loaded_bool
5516 \__zrefclever_compat_module:nn { mathtools }
5517 {
5518   \__zrefclever_if_package_loaded:nT { mathtools }
5519   {
5520     \bool_set_true:N \l__zrefclever_mathtools_loaded_bool
5521     \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5522     {
5523       \seq_map_inline:Nn #1
5524       {

```

```

5525         \tl_set:Nc \l__zrefclever_tmpa_tl
5526             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5527         \bool_lazy_or:nnT
5528             { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { equation } }
5529             { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { parentequation } }
5530             { \noeqref {##1} }
5531     }
5532 }
5533 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5534 }
5535 }

```

## 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.

```

5536 \__zrefclever_compat_module:nn { breqn }
5537 {
5538     \__zrefclever_if_package_loaded:nT { breqn }
5539     {

```

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>).

```

5540     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5541     \AddToHook { env / dgroup / begin }
5542     {
5543         \__zrefclever_zcsetup:e
5544         {
5545             counterresetby =
5546             {
5547                 parentequation =
5548                 \__zrefclever_counter_reset_by:n { equation } ,
5549                 equation = parentequation ,
5550             } ,
5551             currentcounter = parentequation ,
5552             countertype = { parentequation = equation } ,
5553         }
5554     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5555 }
5556 \zref@ifpropundefined { subeq }
5557 { \zref@newprop { subeq } { \alph { equation } } }
5558 { }
5559 \clist_map_inline:nn
5560 {
5561     dmath ,
5562     dseries ,
5563     darray ,
5564 }

```

```

5565     {
5566         \AddToHook { env / #1 / begin }
5567         {
5568             \__zrefclever_zcsetup:n { currentcounter = equation }
5569             \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5570             { \zref@localaddprop \ZREF@mainlist { subeq } }
5571         }
5572     }
5573     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5574 }
5575 }

```

## 9.7 listings

```

5576 \__zrefclever_compat_module:nn { listings }
5577 {
5578     \__zrefclever_if_package_loaded:nT { listings }
5579     {
5580         \__zrefclever_zcsetup:n
5581         {
5582             countertype =
5583             {
5584                 lstlisting = listing ,
5585                 lstnumber = line ,
5586             } ,
5587             counterresetby = { lstnumber = lstlisting } ,
5588         }

```

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.

```

5589     \lst@AddToHook { Init }
5590     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5591     \msg_info:nnn { zref-clever } { compat-package } { listings }
5592 }
5593 }

```

## 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.

```

5594 \__zrefclever_compat_module:nn { enumitem }
5595 {
5596   \__zrefclever_if_package_loaded:nT { enumitem }
5597   {
5598     \int_set:Nn \l__zrefclever_tmpa_int { 5 }
5599     \bool_while_do:nn
5600     {
5601       \cs_if_exist_p:c
5602       { c@ enum \int_to_roman:n { \l__zrefclever_tmpa_int } }
5603     }
5604     {
5605       \__zrefclever_zcsetup:e
5606       {
5607         counterresetby =
5608         {
5609           enum \int_to_roman:n { \l__zrefclever_tmpa_int } =
5610           enum \int_to_roman:n { \l__zrefclever_tmpa_int - 1 }
5611         } ,
5612         countertype =
5613         { enum \int_to_roman:n { \l__zrefclever_tmpa_int } = item } ,
5614       }
5615       \int_incr:N \l__zrefclever_tmpa_int
5616     }
5617     \int_compare:nNnT { \l__zrefclever_tmpa_int } > { 5 }
5618     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5619   }
5620 }

```

## 9.9 subcaption

```

5621 \__zrefclever_compat_module:nn { subcaption }
5622 {
5623   \__zrefclever_if_package_loaded:nT { subcaption }
5624   {
5625     \__zrefclever_zcsetup:n
5626     {
5627       countertype =
5628       {
5629         subfigure = figure ,
5630         subtable = table ,
5631       } ,
5632       counterresetby =
5633       {
5634         subfigure = figure ,
5635         subtable = table ,
5636       } ,
5637     }

```

Support for subref reference.

```

5638   \zref@newprop { subref }
5639   { \cs_if_exist_use:c { thesub \@capttype } }

```



```

5640     \tl_put_right:Nn \caption@subtypehook
5641     { \zref@localaddprop \ZREF@mainlist { subref } }
5642   }
5643 }

```

## 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.

```

5644 \__zrefclever_compat_module:nm { subfig }
5645 {
5646   \__zrefclever_if_package_loaded:nT { subfig }
5647   {
5648     \__zrefclever_zcsetup:n
5649     {
5650       countertype =
5651       {
5652         subfigure = figure ,
5653         subtable = table ,
5654       } ,
5655       counterresetby =
5656       {
5657         subfigure = figure ,
5658         subtable = table ,
5659       } ,
5660     }
5661   }
5662 }

```

## 9.11 beamer

**FIXME** When `beamer` releases fixes for these issues, remove this compatibility module. See <https://github.com/josephwright/beamer/issues/917>.

`beamer` does some really atypical things with regard to cross-references. To start with, it redefines `\label` to receive an optional `<<overlay specification>>` argument. Then, presumably to support overlays, it goes on and hijacks `hyperref`'s anchoring system, sets anchors (`\hypertargets`) to each `label` in the `.snm` file, while letting every standard label's anchor in the `.aux` file default to `Doc-Start`. Of course, having rendered useless `hyperref`'s anchoring, it has to redefine `\ref` so that it uses its own `.snm` provided "label anchors" to make hyperlinks. In particular, from our perspective, there is no support at all for `zref` provided by `beamer`. Which is specially unfortunate since the above procedures also appear to break `cleveref`. See, for example, <https://tex.stackexchange.com/q/266080>, <https://tex.stackexchange.com/q/668998>, and <https://github.com/josephwright/beamer/issues/750>. The work-around provided at <https://tex.stackexchange.com/a/266109> is not general enough since it breaks `cleveref`'s ability to receive a list of labels as argument. Finally, `beamer` also does not set `\@currentcounter` for the frames, making it hard for `zref-clever` to assign the proper type to labels set in that scope.

The technique to set proper anchors is thanks to Ulrike Fischer at <https://tex.stackexchange.com/a/730792>.

```

5663 \__zrefclever_compat_module:nm { beamer }
5664 {

```

```

5665 \__zrefclever_if_class_loaded:nT { beamer }
5666 {
5667   \AddToHookWithArguments { label } [ zref-clever/compat/beamer ]
5668   { \xdef\@currentHref{#1} }
5669   \DeclareHookRule { label }
5670   { zref-clever/compat/beamer } { before } { zref-clever }
5671   \AddToHookWithArguments { cmd/refcounter/before }
5672   [ zref-clever/compat/beamer ]
5673   { \edef\@currentcounter{#1} }
5674 }
5675 }
5676 </package>

```

## 10 Language files

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

### 10.1 Localization guidelines

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

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

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

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

of ‘Paradise Lost’.” It is this third meaning which the `book` reference type is meant to support: a major subdivision of a work, much like `\part`. Even if it does not exist in the standard classes, it may exist elsewhere, in particular, it is provided by `memoir`.

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

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

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

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

**Completeness and abbreviated forms:** Ideally, the language file should be as complete as possible. “Complete” meaning it contains: i) the defaults for all basic sepa-

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

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

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

**Precedence rule for options in the language files:** Any option given twice or more times has to have some precedence rule. Normally, the language files should not contain options in duplicity, but they may happen when setting some “group” `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 `\language` name. But we also don't want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

**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.

```
5677 (*package)
5678 \zcDeclareLanguage { english }
5679 \zcDeclareLanguageAlias { american } { english }
5680 \zcDeclareLanguageAlias { australian } { english }
5681 \zcDeclareLanguageAlias { british } { english }
5682 \zcDeclareLanguageAlias { canadian } { english }
5683 \zcDeclareLanguageAlias { newzealand } { english }
5684 \zcDeclareLanguageAlias { UKenglish } { english }
5685 \zcDeclareLanguageAlias { USenglish } { english }
5686 \endpackage)

5687 (*lang-english)

5688 namesep = {\nobreakspace} ,
5689 pairsep = {\and\nobreakspace} ,
5690 listsep = {,~} ,
5691 lastsep = {\and\nobreakspace} ,
5692 tpairsep = {\and\nobreakspace} ,
5693 tlistsep = {,~} ,
5694 tlastsep = {,~\and\nobreakspace} ,
5695 notesep = {~} ,
5696 rangesep = {\to\nobreakspace} ,
5697
5698 type = book ,
5699   Name-sg = Book ,
5700   name-sg = book ,
5701   Name-pl = Books ,
5702   name-pl = books ,
5703
5704 type = part ,
5705   Name-sg = Part ,
5706   name-sg = part ,
5707   Name-pl = Parts ,
5708   name-pl = parts ,
5709
5710 type = chapter ,
5711   Name-sg = Chapter ,
5712   name-sg = chapter ,
5713   Name-pl = Chapters ,
5714   name-pl = chapters ,
5715
5716 type = section ,
5717   Name-sg = Section ,
5718   name-sg = section ,
5719   Name-pl = Sections ,
5720   name-pl = sections ,
5721
5722 type = paragraph ,
5723   Name-sg = Paragraph ,
5724   name-sg = paragraph ,
5725   Name-pl = Paragraphs ,
5726   name-pl = paragraphs ,
```

```

5727 Name-sg-ab = Par. ,
5728 name-sg-ab = par. ,
5729 Name-pl-ab = Par. ,
5730 name-pl-ab = par. ,
5731
5732 type = appendix ,
5733 Name-sg = Appendix ,
5734 name-sg = appendix ,
5735 Name-pl = Appendices ,
5736 name-pl = appendices ,
5737
5738 type = page ,
5739 Name-sg = Page ,
5740 name-sg = page ,
5741 Name-pl = Pages ,
5742 name-pl = pages ,
5743 rangesep = {\textendash} ,
5744 rangetopair = false ,
5745
5746 type = line ,
5747 Name-sg = Line ,
5748 name-sg = line ,
5749 Name-pl = Lines ,
5750 name-pl = lines ,
5751
5752 type = figure ,
5753 Name-sg = Figure ,
5754 name-sg = figure ,
5755 Name-pl = Figures ,
5756 name-pl = figures ,
5757 Name-sg-ab = Fig. ,
5758 name-sg-ab = fig. ,
5759 Name-pl-ab = Figs. ,
5760 name-pl-ab = figs. ,
5761
5762 type = table ,
5763 Name-sg = Table ,
5764 name-sg = table ,
5765 Name-pl = Tables ,
5766 name-pl = tables ,
5767
5768 type = item ,
5769 Name-sg = Item ,
5770 name-sg = item ,
5771 Name-pl = Items ,
5772 name-pl = items ,
5773
5774 type = footnote ,
5775 Name-sg = Footnote ,
5776 name-sg = footnote ,
5777 Name-pl = Footnotes ,
5778 name-pl = footnotes ,
5779
5780 type = endnote ,

```

```

5781 Name-sg = Note ,
5782 name-sg = note ,
5783 Name-pl = Notes ,
5784 name-pl = notes ,
5785
5786 type = note ,
5787 Name-sg = Note ,
5788 name-sg = note ,
5789 Name-pl = Notes ,
5790 name-pl = notes ,
5791
5792 type = equation ,
5793 Name-sg = Equation ,
5794 name-sg = equation ,
5795 Name-pl = Equations ,
5796 name-pl = equations ,
5797 Name-sg-ab = Eq. ,
5798 name-sg-ab = eq. ,
5799 Name-pl-ab = Eqs. ,
5800 name-pl-ab = eqs. ,
5801 refbounds-first-sg = {,(,)}, ,
5802 refbounds = {(,,)} ,
5803
5804 type = theorem ,
5805 Name-sg = Theorem ,
5806 name-sg = theorem ,
5807 Name-pl = Theorems ,
5808 name-pl = theorems ,
5809
5810 type = lemma ,
5811 Name-sg = Lemma ,
5812 name-sg = lemma ,
5813 Name-pl = Lemmas ,
5814 name-pl = lemmas ,
5815
5816 type = corollary ,
5817 Name-sg = Corollary ,
5818 name-sg = corollary ,
5819 Name-pl = Corollaries ,
5820 name-pl = corollaries ,
5821
5822 type = proposition ,
5823 Name-sg = Proposition ,
5824 name-sg = proposition ,
5825 Name-pl = Propositions ,
5826 name-pl = propositions ,
5827
5828 type = definition ,
5829 Name-sg = Definition ,
5830 name-sg = definition ,
5831 Name-pl = Definitions ,
5832 name-pl = definitions ,
5833
5834 type = proof ,

```

```

5835   Name-sg = Proof ,
5836   name-sg = proof ,
5837   Name-pl = Proofs ,
5838   name-pl = proofs ,
5839
5840 type = result ,
5841   Name-sg = Result ,
5842   name-sg = result ,
5843   Name-pl = Results ,
5844   name-pl = results ,
5845
5846 type = remark ,
5847   Name-sg = Remark ,
5848   name-sg = remark ,
5849   Name-pl = Remarks ,
5850   name-pl = remarks ,
5851
5852 type = example ,
5853   Name-sg = Example ,
5854   name-sg = example ,
5855   Name-pl = Examples ,
5856   name-pl = examples ,
5857
5858 type = algorithm ,
5859   Name-sg = Algorithm ,
5860   name-sg = algorithm ,
5861   Name-pl = Algorithms ,
5862   name-pl = algorithms ,
5863
5864 type = listing ,
5865   Name-sg = Listing ,
5866   name-sg = listing ,
5867   Name-pl = Listings ,
5868   name-pl = listings ,
5869
5870 type = exercise ,
5871   Name-sg = Exercise ,
5872   name-sg = exercise ,
5873   Name-pl = Exercises ,
5874   name-pl = exercises ,
5875
5876 type = solution ,
5877   Name-sg = Solution ,
5878   name-sg = solution ,
5879   Name-pl = Solutions ,
5880   name-pl = solutions ,
5881 </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`.



```

5882 (*package)
5883 \zcDeclareLanguage
5884 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5885 { german }
5886 \zcDeclareLanguageAlias { ngerman      } { german }
5887 \zcDeclareLanguageAlias { austrian     } { german }
5888 \zcDeclareLanguageAlias { naustrian    } { german }
5889 \zcDeclareLanguageAlias { swissgerman  } { german }
5890 \zcDeclareLanguageAlias { nswissgerman } { german }
5891 \end{package}

5892 (*lang-german)

5893 namesep = {\nobreakspace} ,
5894 pairsep  = {\simand\nobreakspace} ,
5895 listsep  = { , \sim } ,
5896 lastsep  = {\simand\nobreakspace} ,
5897 tpairsep = {\simand\nobreakspace} ,
5898 tlistsep = { , \sim } ,
5899 tlastsep = {\simand\nobreakspace} ,
5900 notesep  = { \sim } ,
5901 rangesep = {\simbis\nobreakspace} ,
5902
5903 type = book ,
5904 gender = n ,
5905 case = N ,
5906 Name-sg = Buch ,
5907 Name-pl = Bücher ,
5908 case = A ,
5909 Name-sg = Buch ,
5910 Name-pl = Bücher ,
5911 case = D ,
5912 Name-sg = Buch ,
5913 Name-pl = Büchern ,
5914 case = G ,
5915 Name-sg = Buches ,
5916 Name-pl = Bücher ,
5917
5918 type = part ,
5919 gender = m ,
5920 case = N ,
5921 Name-sg = Teil ,
5922 Name-pl = Teile ,
5923 case = A ,
5924 Name-sg = Teil ,
5925 Name-pl = Teile ,
5926 case = D ,
5927 Name-sg = Teil ,
5928 Name-pl = Teilen ,
5929 case = G ,
5930 Name-sg = Teiles ,
5931 Name-pl = Teile ,
5932
5933 type = chapter ,
5934 gender = n ,

```

```

5935 case = N ,
5936     Name-sg = Kapitel ,
5937     Name-pl = Kapitel ,
5938 case = A ,
5939     Name-sg = Kapitel ,
5940     Name-pl = Kapitel ,
5941 case = D ,
5942     Name-sg = Kapitel ,
5943     Name-pl = Kapiteln ,
5944 case = G ,
5945     Name-sg = Kapitels ,
5946     Name-pl = Kapitel ,
5947
5948 type = section ,
5949     gender = m ,
5950     case = N ,
5951         Name-sg = Abschnitt ,
5952         Name-pl = Abschnitte ,
5953     case = A ,
5954         Name-sg = Abschnitt ,
5955         Name-pl = Abschnitte ,
5956     case = D ,
5957         Name-sg = Abschnitt ,
5958         Name-pl = Abschnitten ,
5959     case = G ,
5960         Name-sg = Abschnitts ,
5961         Name-pl = Abschnitte ,
5962
5963 type = paragraph ,
5964     gender = m ,
5965     case = N ,
5966         Name-sg = Absatz ,
5967         Name-pl = Absätze ,
5968     case = A ,
5969         Name-sg = Absatz ,
5970         Name-pl = Absätze ,
5971     case = D ,
5972         Name-sg = Absatz ,
5973         Name-pl = Absätzen ,
5974     case = G ,
5975         Name-sg = Absatzes ,
5976         Name-pl = Absätze ,
5977
5978 type = appendix ,
5979     gender = m ,
5980     case = N ,
5981         Name-sg = Anhang ,
5982         Name-pl = Anhänge ,
5983     case = A ,
5984         Name-sg = Anhang ,
5985         Name-pl = Anhänge ,
5986     case = D ,
5987         Name-sg = Anhang ,
5988         Name-pl = Anhängen ,

```

```

5989 case = G ,
5990     Name-sg = Anhangs ,
5991     Name-pl = Anhänge ,
5992
5993 type = page ,
5994     gender = f ,
5995     case = N ,
5996         Name-sg = Seite ,
5997         Name-pl = Seiten ,
5998     case = A ,
5999         Name-sg = Seite ,
6000         Name-pl = Seiten ,
6001     case = D ,
6002         Name-sg = Seite ,
6003         Name-pl = Seiten ,
6004     case = G ,
6005         Name-sg = Seite ,
6006         Name-pl = Seiten ,
6007     rangesep = {\textendash} ,
6008     rangetopair = false ,
6009
6010 type = line ,
6011     gender = f ,
6012     case = N ,
6013         Name-sg = Zeile ,
6014         Name-pl = Zeilen ,
6015     case = A ,
6016         Name-sg = Zeile ,
6017         Name-pl = Zeilen ,
6018     case = D ,
6019         Name-sg = Zeile ,
6020         Name-pl = Zeilen ,
6021     case = G ,
6022         Name-sg = Zeile ,
6023         Name-pl = Zeilen ,
6024
6025 type = figure ,
6026     gender = f ,
6027     case = N ,
6028         Name-sg = Abbildung ,
6029         Name-pl = Abbildungen ,
6030         Name-sg-ab = Abb. ,
6031         Name-pl-ab = Abb. ,
6032     case = A ,
6033         Name-sg = Abbildung ,
6034         Name-pl = Abbildungen ,
6035         Name-sg-ab = Abb. ,
6036         Name-pl-ab = Abb. ,
6037     case = D ,
6038         Name-sg = Abbildung ,
6039         Name-pl = Abbildungen ,
6040         Name-sg-ab = Abb. ,
6041         Name-pl-ab = Abb. ,
6042     case = G ,

```

```

6043     Name-sg = Abbildung ,
6044     Name-pl = Abbildungen ,
6045     Name-sg-ab = Abb. ,
6046     Name-pl-ab = Abb. ,
6047
6048 type = table ,
6049     gender = f ,
6050     case = N ,
6051     Name-sg = Tabelle ,
6052     Name-pl = Tabellen ,
6053     case = A ,
6054     Name-sg = Tabelle ,
6055     Name-pl = Tabellen ,
6056     case = D ,
6057     Name-sg = Tabelle ,
6058     Name-pl = Tabellen ,
6059     case = G ,
6060     Name-sg = Tabelle ,
6061     Name-pl = Tabellen ,
6062
6063 type = item ,
6064     gender = m ,
6065     case = N ,
6066     Name-sg = Punkt ,
6067     Name-pl = Punkte ,
6068     case = A ,
6069     Name-sg = Punkt ,
6070     Name-pl = Punkte ,
6071     case = D ,
6072     Name-sg = Punkt ,
6073     Name-pl = Punkten ,
6074     case = G ,
6075     Name-sg = Punktes ,
6076     Name-pl = Punkte ,
6077
6078 type = footnote ,
6079     gender = f ,
6080     case = N ,
6081     Name-sg = Fußnote ,
6082     Name-pl = Fußnoten ,
6083     case = A ,
6084     Name-sg = Fußnote ,
6085     Name-pl = Fußnoten ,
6086     case = D ,
6087     Name-sg = Fußnote ,
6088     Name-pl = Fußnoten ,
6089     case = G ,
6090     Name-sg = Fußnote ,
6091     Name-pl = Fußnoten ,
6092
6093 type = endnote ,
6094     gender = f ,
6095     case = N ,
6096     Name-sg = Endnote ,

```

```

6097     Name-pl = Endnoten ,
6098     case = A ,
6099     Name-sg = Endnote ,
6100     Name-pl = Endnoten ,
6101     case = D ,
6102     Name-sg = Endnote ,
6103     Name-pl = Endnoten ,
6104     case = G ,
6105     Name-sg = Endnote ,
6106     Name-pl = Endnoten ,
6107
6108 type = note ,
6109     gender = f ,
6110     case = N ,
6111     Name-sg = Anmerkung ,
6112     Name-pl = Anmerkungen ,
6113     case = A ,
6114     Name-sg = Anmerkung ,
6115     Name-pl = Anmerkungen ,
6116     case = D ,
6117     Name-sg = Anmerkung ,
6118     Name-pl = Anmerkungen ,
6119     case = G ,
6120     Name-sg = Anmerkung ,
6121     Name-pl = Anmerkungen ,
6122
6123 type = equation ,
6124     gender = f ,
6125     case = N ,
6126     Name-sg = Gleichung ,
6127     Name-pl = Gleichungen ,
6128     case = A ,
6129     Name-sg = Gleichung ,
6130     Name-pl = Gleichungen ,
6131     case = D ,
6132     Name-sg = Gleichung ,
6133     Name-pl = Gleichungen ,
6134     case = G ,
6135     Name-sg = Gleichung ,
6136     Name-pl = Gleichungen ,
6137     refbounds-first-sg = {,(,)}, ,
6138     refbounds = {(,,)} ,
6139
6140 type = theorem ,
6141     gender = n ,
6142     case = N ,
6143     Name-sg = Theorem ,
6144     Name-pl = Theoreme ,
6145     case = A ,
6146     Name-sg = Theorem ,
6147     Name-pl = Theoreme ,
6148     case = D ,
6149     Name-sg = Theorem ,
6150     Name-pl = Theoremen ,

```

```

6151 case = G ,
6152     Name-sg = Theorems ,
6153     Name-pl = Theoreme ,
6154
6155 type = lemma ,
6156     gender = n ,
6157     case = N ,
6158         Name-sg = Lemma ,
6159         Name-pl = Lemmata ,
6160     case = A ,
6161         Name-sg = Lemma ,
6162         Name-pl = Lemmata ,
6163     case = D ,
6164         Name-sg = Lemma ,
6165         Name-pl = Lemmata ,
6166     case = G ,
6167         Name-sg = Lemmas ,
6168         Name-pl = Lemmata ,
6169
6170 type = corollary ,
6171     gender = n ,
6172     case = N ,
6173         Name-sg = Korollar ,
6174         Name-pl = Korollare ,
6175     case = A ,
6176         Name-sg = Korollar ,
6177         Name-pl = Korollare ,
6178     case = D ,
6179         Name-sg = Korollar ,
6180         Name-pl = Korollaren ,
6181     case = G ,
6182         Name-sg = Korollars ,
6183         Name-pl = Korollare ,
6184
6185 type = proposition ,
6186     gender = m ,
6187     case = N ,
6188         Name-sg = Satz ,
6189         Name-pl = Sätze ,
6190     case = A ,
6191         Name-sg = Satz ,
6192         Name-pl = Sätze ,
6193     case = D ,
6194         Name-sg = Satz ,
6195         Name-pl = Sätzen ,
6196     case = G ,
6197         Name-sg = Satzes ,
6198         Name-pl = Sätze ,
6199
6200 type = definition ,
6201     gender = f ,
6202     case = N ,
6203         Name-sg = Definition ,
6204         Name-pl = Definitionen ,

```

```

6205 case = A ,
6206     Name-sg = Definition ,
6207     Name-pl = Definitionen ,
6208 case = D ,
6209     Name-sg = Definition ,
6210     Name-pl = Definitionen ,
6211 case = G ,
6212     Name-sg = Definition ,
6213     Name-pl = Definitionen ,
6214
6215 type = proof ,
6216     gender = m ,
6217     case = N ,
6218         Name-sg = Beweis ,
6219         Name-pl = Beweise ,
6220 case = A ,
6221     Name-sg = Beweis ,
6222     Name-pl = Beweise ,
6223 case = D ,
6224     Name-sg = Beweis ,
6225     Name-pl = Beweisen ,
6226 case = G ,
6227     Name-sg = Beweises ,
6228     Name-pl = Beweise ,
6229
6230 type = result ,
6231     gender = n ,
6232     case = N ,
6233         Name-sg = Ergebnis ,
6234         Name-pl = Ergebnisse ,
6235 case = A ,
6236     Name-sg = Ergebnis ,
6237     Name-pl = Ergebnisse ,
6238 case = D ,
6239     Name-sg = Ergebnis ,
6240     Name-pl = Ergebnissen ,
6241 case = G ,
6242     Name-sg = Ergebnisses ,
6243     Name-pl = Ergebnisse ,
6244
6245 type = remark ,
6246     gender = f ,
6247     case = N ,
6248         Name-sg = Bemerkung ,
6249         Name-pl = Bemerkungen ,
6250 case = A ,
6251     Name-sg = Bemerkung ,
6252     Name-pl = Bemerkungen ,
6253 case = D ,
6254     Name-sg = Bemerkung ,
6255     Name-pl = Bemerkungen ,
6256 case = G ,
6257     Name-sg = Bemerkung ,
6258     Name-pl = Bemerkungen ,

```

```

6259
6260 type = example ,
6261     gender = n ,
6262     case = N ,
6263     Name-sg = Beispiel ,
6264     Name-pl = Beispiele ,
6265     case = A ,
6266     Name-sg = Beispiel ,
6267     Name-pl = Beispiele ,
6268     case = D ,
6269     Name-sg = Beispiel ,
6270     Name-pl = Beispielen ,
6271     case = G ,
6272     Name-sg = Beispiels ,
6273     Name-pl = Beispiele ,
6274
6275 type = algorithm ,
6276     gender = m ,
6277     case = N ,
6278     Name-sg = Algorithmus ,
6279     Name-pl = Algorithmen ,
6280     case = A ,
6281     Name-sg = Algorithmus ,
6282     Name-pl = Algorithmen ,
6283     case = D ,
6284     Name-sg = Algorithmus ,
6285     Name-pl = Algorithmen ,
6286     case = G ,
6287     Name-sg = Algorithmus ,
6288     Name-pl = Algorithmen ,
6289
6290 type = listing ,
6291     gender = n ,
6292     case = N ,
6293     Name-sg = Listing ,
6294     Name-pl = Listings ,
6295     case = A ,
6296     Name-sg = Listing ,
6297     Name-pl = Listings ,
6298     case = D ,
6299     Name-sg = Listing ,
6300     Name-pl = Listings ,
6301     case = G ,
6302     Name-sg = Listings ,
6303     Name-pl = Listings ,
6304
6305 type = exercise ,
6306     gender = f ,
6307     case = N ,
6308     Name-sg = Übungsaufgabe ,
6309     Name-pl = Übungsaufgaben ,
6310     case = A ,
6311     Name-sg = Übungsaufgabe ,
6312     Name-pl = Übungsaufgaben ,

```



```

6313 case = D ,
6314     Name-sg = Übungsaufgabe ,
6315     Name-pl = Übungsaufgaben ,
6316 case = G ,
6317     Name-sg = Übungsaufgabe ,
6318     Name-pl = Übungsaufgaben ,
6319
6320 type = solution ,
6321     gender = f ,
6322     case = N ,
6323     Name-sg = Lösung ,
6324     Name-pl = Lösungen ,
6325     case = A ,
6326     Name-sg = Lösung ,
6327     Name-pl = Lösungen ,
6328     case = D ,
6329     Name-sg = Lösung ,
6330     Name-pl = Lösungen ,
6331     case = G ,
6332     Name-sg = Lösung ,
6333     Name-pl = Lösungen ,
6334 </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  $\TeX$  (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 `francais`, `frenchb`, and `canadien`, but they are deprecated as options and, if used, they fall back to either `french` or `acadian`.

```

6335 (*package)
6336 \zcDeclareLanguage [ gender = { f , m } ] { french }
6337 \zcDeclareLanguageAlias { acadian } { french }
6338 </package>
6339 (*lang-french)
6340 namesep = {\nobreakspace} ,
6341 pairsep = {\et\nobreakspace} ,
6342 listsep = {,~} ,
6343 lastsep = {\et\nobreakspace} ,
6344 tpairsep = {\et\nobreakspace} ,
6345 tlistsep = {,~} ,
6346 tlastsep = {\et\nobreakspace} ,
6347 notesep = {~} ,
6348 rangeseq = {\à\nobreakspace} ,
6349
6350 type = book ,
6351     gender = m ,
6352     Name-sg = Livre ,
6353     name-sg = livre ,
6354     Name-pl = Livres ,

```

```

6355 name-pl = livres ,
6356
6357 type = part ,
6358     gender = f ,
6359     Name-sg = Partie ,
6360     name-sg = partie ,
6361     Name-pl = Parties ,
6362     name-pl = parties ,
6363
6364 type = chapter ,
6365     gender = m ,
6366     Name-sg = Chapitre ,
6367     name-sg = chapitre ,
6368     Name-pl = Chapitres ,
6369     name-pl = chapitres ,
6370
6371 type = section ,
6372     gender = f ,
6373     Name-sg = Section ,
6374     name-sg = section ,
6375     Name-pl = Sections ,
6376     name-pl = sections ,
6377
6378 type = paragraph ,
6379     gender = m ,
6380     Name-sg = Paragraphe ,
6381     name-sg = paragraphe ,
6382     Name-pl = Paragraphes ,
6383     name-pl = paragraphes ,
6384
6385 type = appendix ,
6386     gender = f ,
6387     Name-sg = Annexe ,
6388     name-sg = annexe ,
6389     Name-pl = Annexes ,
6390     name-pl = annexes ,
6391
6392 type = page ,
6393     gender = f ,
6394     Name-sg = Page ,
6395     name-sg = page ,
6396     Name-pl = Pages ,
6397     name-pl = pages ,
6398     rangesep = {-} ,
6399     rangetopair = false ,
6400
6401 type = line ,
6402     gender = f ,
6403     Name-sg = Ligne ,
6404     name-sg = ligne ,
6405     Name-pl = Lignes ,
6406     name-pl = lignes ,
6407
6408 type = figure ,

```

```

6409   gender = f ,
6410   Name-sg = Figure ,
6411   name-sg = figure ,
6412   Name-pl = Figures ,
6413   name-pl = figures ,
6414
6415   type = table ,
6416   gender = f ,
6417   Name-sg = Table ,
6418   name-sg = table ,
6419   Name-pl = Tables ,
6420   name-pl = tables ,
6421
6422   type = item ,
6423   gender = m ,
6424   Name-sg = Point ,
6425   name-sg = point ,
6426   Name-pl = Points ,
6427   name-pl = points ,
6428
6429   type = footnote ,
6430   gender = f ,
6431   Name-sg = Note ,
6432   name-sg = note ,
6433   Name-pl = Notes ,
6434   name-pl = notes ,
6435
6436   type = endnote ,
6437   gender = f ,
6438   Name-sg = Note ,
6439   name-sg = note ,
6440   Name-pl = Notes ,
6441   name-pl = notes ,
6442
6443   type = note ,
6444   gender = f ,
6445   Name-sg = Note ,
6446   name-sg = note ,
6447   Name-pl = Notes ,
6448   name-pl = notes ,
6449
6450   type = equation ,
6451   gender = f ,
6452   Name-sg = Équation ,
6453   name-sg = équation ,
6454   Name-pl = Équations ,
6455   name-pl = équations ,
6456   refbounds-first-sg = {,(,)}, ,
6457   refbounds = {(,,)} ,
6458
6459   type = theorem ,
6460   gender = m ,
6461   Name-sg = Théorème ,
6462   name-sg = théorème ,

```

```

6463 Name-pl = Théorèmes ,
6464 name-pl = théorèmes ,
6465
6466 type = lemma ,
6467 gender = m ,
6468 Name-sg = Lemme ,
6469 name-sg = lemme ,
6470 Name-pl = Lemmes ,
6471 name-pl = lemmes ,
6472
6473 type = corollary ,
6474 gender = m ,
6475 Name-sg = Corollaire ,
6476 name-sg = corollaire ,
6477 Name-pl = Corollaires ,
6478 name-pl = corollaires ,
6479
6480 type = proposition ,
6481 gender = f ,
6482 Name-sg = Proposition ,
6483 name-sg = proposition ,
6484 Name-pl = Propositions ,
6485 name-pl = propositions ,
6486
6487 type = definition ,
6488 gender = f ,
6489 Name-sg = Définition ,
6490 name-sg = définition ,
6491 Name-pl = Définitions ,
6492 name-pl = définitions ,
6493
6494 type = proof ,
6495 gender = f ,
6496 Name-sg = Démonstration ,
6497 name-sg = démonstration ,
6498 Name-pl = Démonstrations ,
6499 name-pl = démonstrations ,
6500
6501 type = result ,
6502 gender = m ,
6503 Name-sg = Résultat ,
6504 name-sg = résultat ,
6505 Name-pl = Résultats ,
6506 name-pl = résultats ,
6507
6508 type = remark ,
6509 gender = f ,
6510 Name-sg = Remarque ,
6511 name-sg = remarque ,
6512 Name-pl = Remarques ,
6513 name-pl = remarques ,
6514
6515 type = example ,
6516 gender = m ,

```

```

6517 Name-sg = Exemple ,
6518 name-sg = exemple ,
6519 Name-pl = Exemples ,
6520 name-pl = exemples ,
6521
6522 type = algorithm ,
6523   gender = m ,
6524   Name-sg = Algorithmme ,
6525   name-sg = algorithmme ,
6526   Name-pl = Algorithmmes ,
6527   name-pl = algorithmmes ,
6528
6529 type = listing ,
6530   gender = m ,
6531   Name-sg = Listing ,
6532   name-sg = listing ,
6533   Name-pl = Listings ,
6534   name-pl = listings ,
6535
6536 type = exercise ,
6537   gender = m ,
6538   Name-sg = Exercice ,
6539   name-sg = exercice ,
6540   Name-pl = Exercices ,
6541   name-pl = exercices ,
6542
6543 type = solution ,
6544   gender = f ,
6545   Name-sg = Solution ,
6546   name-sg = solution ,
6547   Name-pl = Solutions ,
6548   name-pl = solutions ,
6549 
```

## 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.

```

6550 (*package)
6551 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6552 \zcDeclareLanguageAlias { brazilian } { portuguese }
6553 \zcDeclareLanguageAlias { brazil } { portuguese }
6554 \zcDeclareLanguageAlias { portuges } { portuguese }
6555 
```

```

6556 (*lang-portuguese)

6557 namesep = {\nobreakspace} ,
6558 pairsep = {\~e\nobreakspace} ,
6559 listsep = { , ~ } ,
6560 lastsep = {\~e\nobreakspace} ,
6561 tpairsep = {\~e\nobreakspace} ,
6562 tlistsep = { , ~ } ,

```

```

6563 tlastsep = {~e\nobreakspace} ,
6564 notesep = {~} ,
6565 rangesep = {~a\nobreakspace} ,
6566
6567 type = book ,
6568     gender = m ,
6569     Name-sg = Livro ,
6570     name-sg = livro ,
6571     Name-pl = Livros ,
6572     name-pl = livros ,
6573
6574 type = part ,
6575     gender = f ,
6576     Name-sg = Parte ,
6577     name-sg = parte ,
6578     Name-pl = Partes ,
6579     name-pl = partes ,
6580
6581 type = chapter ,
6582     gender = m ,
6583     Name-sg = Capítulo ,
6584     name-sg = capítulo ,
6585     Name-pl = Capítulos ,
6586     name-pl = capítulos ,
6587
6588 type = section ,
6589     gender = f ,
6590     Name-sg = Seção ,
6591     name-sg = seção ,
6592     Name-pl = Seções ,
6593     name-pl = seções ,
6594
6595 type = paragraph ,
6596     gender = m ,
6597     Name-sg = Parágrafo ,
6598     name-sg = parágrafo ,
6599     Name-pl = Parágrafos ,
6600     name-pl = parágrafos ,
6601     Name-sg-ab = Par. ,
6602     name-sg-ab = par. ,
6603     Name-pl-ab = Par. ,
6604     name-pl-ab = par. ,
6605
6606 type = appendix ,
6607     gender = m ,
6608     Name-sg = Apêndice ,
6609     name-sg = apêndice ,
6610     Name-pl = Apêndices ,
6611     name-pl = apêndices ,
6612
6613 type = page ,
6614     gender = f ,
6615     Name-sg = Página ,
6616     name-sg = página ,

```

```

6617 Name-pl = Páginas ,
6618 name-pl = páginas ,
6619 rangesep = {\textendash} ,
6620 rangetopair = false ,
6621
6622 type = line ,
6623 gender = f ,
6624 Name-sg = Linha ,
6625 name-sg = linha ,
6626 Name-pl = Linhas ,
6627 name-pl = linhas ,
6628
6629 type = figure ,
6630 gender = f ,
6631 Name-sg = Figura ,
6632 name-sg = figura ,
6633 Name-pl = Figuras ,
6634 name-pl = figuras ,
6635 Name-sg-ab = Fig. ,
6636 name-sg-ab = fig. ,
6637 Name-pl-ab = Figs. ,
6638 name-pl-ab = figs. ,
6639
6640 type = table ,
6641 gender = f ,
6642 Name-sg = Tabela ,
6643 name-sg = tabela ,
6644 Name-pl = Tabelas ,
6645 name-pl = tabelas ,
6646
6647 type = item ,
6648 gender = m ,
6649 Name-sg = Item ,
6650 name-sg = item ,
6651 Name-pl = Itens ,
6652 name-pl = itens ,
6653
6654 type = footnote ,
6655 gender = f ,
6656 Name-sg = Nota ,
6657 name-sg = nota ,
6658 Name-pl = Notas ,
6659 name-pl = notas ,
6660
6661 type = endnote ,
6662 gender = f ,
6663 Name-sg = Nota ,
6664 name-sg = nota ,
6665 Name-pl = Notas ,
6666 name-pl = notas ,
6667
6668 type = note ,
6669 gender = f ,
6670 Name-sg = Nota ,

```

```

6671 name-sg = nota ,
6672 Name-pl = Notas ,
6673 name-pl = notas ,
6674
6675 type = equation ,
6676 gender = f ,
6677 Name-sg = Equação ,
6678 name-sg = equação ,
6679 Name-pl = Equações ,
6680 name-pl = equações ,
6681 Name-sg-ab = Eq. ,
6682 name-sg-ab = eq. ,
6683 Name-pl-ab = Eqs. ,
6684 name-pl-ab = eqs. ,
6685 refbounds-first-sg = {,(,)}, ,
6686 refbounds = {(,,)} ,
6687
6688 type = theorem ,
6689 gender = m ,
6690 Name-sg = Teorema ,
6691 name-sg = teorema ,
6692 Name-pl = Teoremas ,
6693 name-pl = teoremas ,
6694
6695 type = lemma ,
6696 gender = m ,
6697 Name-sg = Lema ,
6698 name-sg = lema ,
6699 Name-pl = Lemas ,
6700 name-pl = lemas ,
6701
6702 type = corollary ,
6703 gender = m ,
6704 Name-sg = Corolário ,
6705 name-sg = corolário ,
6706 Name-pl = Corolários ,
6707 name-pl = corolários ,
6708
6709 type = proposition ,
6710 gender = f ,
6711 Name-sg = Proposição ,
6712 name-sg = proposição ,
6713 Name-pl = Proposições ,
6714 name-pl = proposições ,
6715
6716 type = definition ,
6717 gender = f ,
6718 Name-sg = Definição ,
6719 name-sg = definição ,
6720 Name-pl = Definições ,
6721 name-pl = definições ,
6722
6723 type = proof ,
6724 gender = f ,

```



```

6725 Name-sg = Demonstração ,
6726 name-sg = demonstração ,
6727 Name-pl = Demonstrações ,
6728 name-pl = demonstrações ,
6729
6730 type = result ,
6731 gender = m ,
6732 Name-sg = Resultado ,
6733 name-sg = resultado ,
6734 Name-pl = Resultados ,
6735 name-pl = resultados ,
6736
6737 type = remark ,
6738 gender = f ,
6739 Name-sg = Observação ,
6740 name-sg = observação ,
6741 Name-pl = Observações ,
6742 name-pl = observações ,
6743
6744 type = example ,
6745 gender = m ,
6746 Name-sg = Exemplo ,
6747 name-sg = exemplo ,
6748 Name-pl = Exemplos ,
6749 name-pl = exemplos ,
6750
6751 type = algorithm ,
6752 gender = m ,
6753 Name-sg = Algoritmo ,
6754 name-sg = algoritmo ,
6755 Name-pl = Algoritmos ,
6756 name-pl = algoritmos ,
6757
6758 type = listing ,
6759 gender = f ,
6760 Name-sg = Listagem ,
6761 name-sg = listagem ,
6762 Name-pl = Listagens ,
6763 name-pl = listagens ,
6764
6765 type = exercise ,
6766 gender = m ,
6767 Name-sg = Exercício ,
6768 name-sg = exercício ,
6769 Name-pl = Exercícios ,
6770 name-pl = exercícios ,
6771
6772 type = solution ,
6773 gender = f ,
6774 Name-sg = Solução ,
6775 name-sg = solução ,
6776 Name-pl = Soluções ,
6777 name-pl = soluções ,
6778 </lang-portuguese>

```

## 10.6 Spanish

Spanish language file has been initially provided by the author.

```
6779 (*package)
6780 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6781 \end{package}

6782 (*lang-spanish)

6783 namesep = {\nobreakspace} ,
6784 pairsep = {\~y\nobreakspace} ,
6785 listsep = {,~} ,
6786 lastsep = {\~y\nobreakspace} ,
6787 tpairsep = {\~y\nobreakspace} ,
6788 tlistsep = {,~} ,
6789 tlastsep = {\~y\nobreakspace} ,
6790 notesep = {\~} ,
6791 rangesep = {\~a\nobreakspace} ,
6792
6793 type = book ,
6794   gender = m ,
6795   Name-sg = Libro ,
6796   name-sg = libro ,
6797   Name-pl = Libros ,
6798   name-pl = libros ,
6799
6800 type = part ,
6801   gender = f ,
6802   Name-sg = Parte ,
6803   name-sg = parte ,
6804   Name-pl = Partes ,
6805   name-pl = partes ,
6806
6807 type = chapter ,
6808   gender = m ,
6809   Name-sg = Capítulo ,
6810   name-sg = capítulo ,
6811   Name-pl = Capítulos ,
6812   name-pl = capítulos ,
6813
6814 type = section ,
6815   gender = f ,
6816   Name-sg = Sección ,
6817   name-sg = sección ,
6818   Name-pl = Secciones ,
6819   name-pl = secciones ,
6820
6821 type = paragraph ,
6822   gender = m ,
6823   Name-sg = Párrafo ,
6824   name-sg = párrafo ,
6825   Name-pl = Párrafos ,
6826   name-pl = párrafos ,
6827
6828 type = appendix ,
```

```

6829     gender = m ,
6830     Name-sg = Apéndice ,
6831     name-sg = apéndice ,
6832     Name-pl = Apéndices ,
6833     name-pl = apéndices ,
6834
6835     type = page ,
6836     gender = f ,
6837     Name-sg = Página ,
6838     name-sg = página ,
6839     Name-pl = Páginas ,
6840     name-pl = páginas ,
6841     rangeseq = {\textendash} ,
6842     rangetopair = false ,
6843
6844     type = line ,
6845     gender = f ,
6846     Name-sg = Línea ,
6847     name-sg = línea ,
6848     Name-pl = Líneas ,
6849     name-pl = líneas ,
6850
6851     type = figure ,
6852     gender = f ,
6853     Name-sg = Figura ,
6854     name-sg = figura ,
6855     Name-pl = Figuras ,
6856     name-pl = figuras ,
6857
6858     type = table ,
6859     gender = m ,
6860     Name-sg = Cuadro ,
6861     name-sg = cuadro ,
6862     Name-pl = Cuadros ,
6863     name-pl = cuadros ,
6864
6865     type = item ,
6866     gender = m ,
6867     Name-sg = Punto ,
6868     name-sg = punto ,
6869     Name-pl = Puntos ,
6870     name-pl = puntos ,
6871
6872     type = footnote ,
6873     gender = f ,
6874     Name-sg = Nota ,
6875     name-sg = nota ,
6876     Name-pl = Notas ,
6877     name-pl = notas ,
6878
6879     type = endnote ,
6880     gender = f ,
6881     Name-sg = Nota ,
6882     name-sg = nota ,

```

```

6883 Name-pl = Notas ,
6884 name-pl = notas ,
6885
6886 type = note ,
6887 gender = f ,
6888 Name-sg = Nota ,
6889 name-sg = nota ,
6890 Name-pl = Notas ,
6891 name-pl = notas ,
6892
6893 type = equation ,
6894 gender = f ,
6895 Name-sg = Ecuación ,
6896 name-sg = ecuación ,
6897 Name-pl = Ecuaciones ,
6898 name-pl = ecuaciones ,
6899 refbounds-first-sg = {,(,)}, ,
6900 refbounds = {(,,)} ,
6901
6902 type = theorem ,
6903 gender = m ,
6904 Name-sg = Teorema ,
6905 name-sg = teorema ,
6906 Name-pl = Teoremas ,
6907 name-pl = teoremas ,
6908
6909 type = lemma ,
6910 gender = m ,
6911 Name-sg = Lema ,
6912 name-sg = lema ,
6913 Name-pl = Lemas ,
6914 name-pl = lemas ,
6915
6916 type = corollary ,
6917 gender = m ,
6918 Name-sg = Corolario ,
6919 name-sg = corolario ,
6920 Name-pl = Corolarios ,
6921 name-pl = corolarios ,
6922
6923 type = proposition ,
6924 gender = f ,
6925 Name-sg = Proposición ,
6926 name-sg = proposición ,
6927 Name-pl = Proposiciones ,
6928 name-pl = proposiciones ,
6929
6930 type = definition ,
6931 gender = f ,
6932 Name-sg = Definición ,
6933 name-sg = definición ,
6934 Name-pl = Definiciones ,
6935 name-pl = definiciones ,
6936

```

```

6937 type = proof ,
6938     gender = f ,
6939     Name-sg = Demostración ,
6940     name-sg = demostración ,
6941     Name-pl = Demostraciones ,
6942     name-pl = demostraciones ,
6943
6944 type = result ,
6945     gender = m ,
6946     Name-sg = Resultado ,
6947     name-sg = resultado ,
6948     Name-pl = Resultados ,
6949     name-pl = resultados ,
6950
6951 type = remark ,
6952     gender = f ,
6953     Name-sg = Observación ,
6954     name-sg = observación ,
6955     Name-pl = Observaciones ,
6956     name-pl = observaciones ,
6957
6958 type = example ,
6959     gender = m ,
6960     Name-sg = Ejemplo ,
6961     name-sg = ejemplo ,
6962     Name-pl = Ejemplos ,
6963     name-pl = ejemplos ,
6964
6965 type = algorithm ,
6966     gender = m ,
6967     Name-sg = Algoritmo ,
6968     name-sg = algoritmo ,
6969     Name-pl = Algoritmos ,
6970     name-pl = algoritmos ,
6971
6972 type = listing ,
6973     gender = m ,
6974     Name-sg = Listado ,
6975     name-sg = listado ,
6976     Name-pl = Listados ,
6977     name-pl = listados ,
6978
6979 type = exercise ,
6980     gender = m ,
6981     Name-sg = Ejercicio ,
6982     name-sg = ejercicio ,
6983     Name-pl = Ejercicios ,
6984     name-pl = ejercicios ,
6985
6986 type = solution ,
6987     gender = f ,
6988     Name-sg = Solución ,
6989     name-sg = solución ,
6990     Name-pl = Soluciones ,

```

```

6991 name-pl = soluciones ,
6992 </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.

```

6993 <*package>
6994 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6995 </package>
6996 <*lang-dutch>
6997 namesep = {\nobreakspace} ,
6998 pairsep = {\en\nobreakspace} ,
6999 listsep = {,~} ,
7000 lastsep = {\en\nobreakspace} ,
7001 tpairsep = {\en\nobreakspace} ,
7002 tlistsep = {,~} ,
7003 tlastsep = {,~en\nobreakspace} ,
7004 notesep = {~} ,
7005 rangesep = {\t/m\nobreakspace} ,
7006
7007 type = book ,
7008 gender = n ,
7009 Name-sg = Boek ,
7010 name-sg = boek ,
7011 Name-pl = Boeken ,
7012 name-pl = boeken ,
7013
7014 type = part ,
7015 gender = n ,
7016 Name-sg = Deel ,
7017 name-sg = deel ,
7018 Name-pl = Delen ,
7019 name-pl = delen ,
7020
7021 type = chapter ,
7022 gender = n ,
7023 Name-sg = Hoofdstuk ,
7024 name-sg = hoofdstuk ,
7025 Name-pl = Hoofdstukken ,
7026 name-pl = hoofdstukken ,
7027
7028 type = section ,
7029 gender = m ,
7030 Name-sg = Paragraaf ,
7031 name-sg = paragraaf ,
7032 Name-pl = Paragrafen ,
7033 name-pl = paragrafen ,
7034
7035 type = paragraph ,
7036 gender = f ,
7037 Name-sg = Alinea ,

```

```

7038 name-sg = alinea ,
7039 Name-pl = Alinea's ,
7040 name-pl = alinea's ,
7041

```

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.

```

7042 type = appendix ,
7043   gender = { f , m } ,
7044   Name-sg = Bijlage ,
7045   name-sg = bijlage ,
7046   Name-pl = Bijlagen ,
7047   name-pl = bijlagen ,
7048
7049 type = page ,
7050   gender = { f , m } ,
7051   Name-sg = Pagina ,
7052   name-sg = pagina ,
7053   Name-pl = Pagina's ,
7054   name-pl = pagina's ,
7055   rangesep = {\textendash} ,
7056   rangetopair = false ,
7057
7058 type = line ,
7059   gender = m ,
7060   Name-sg = Regel ,
7061   name-sg = regel ,
7062   Name-pl = Regels ,
7063   name-pl = regels ,
7064
7065 type = figure ,
7066   gender = { n , f , m } ,
7067   Name-sg = Figuur ,
7068   name-sg = figuur ,
7069   Name-pl = Figuren ,
7070   name-pl = figuren ,
7071
7072 type = table ,
7073   gender = { f , m } ,
7074   Name-sg = Tabel ,
7075   name-sg = tabel ,
7076   Name-pl = Tabellen ,
7077   name-pl = tabellen ,
7078
7079 type = item ,
7080   gender = n ,
7081   Name-sg = Punt ,
7082   name-sg = punt ,
7083   Name-pl = Punten ,
7084   name-pl = punten ,
7085
7086 type = footnote ,
7087   gender = { f , m } ,

```

```

7088 Name-sg = Voetnoot ,
7089 name-sg = voetnoot ,
7090 Name-pl = Voetnoten ,
7091 name-pl = voetnoten ,
7092
7093 type = endnote ,
7094   gender = { f , m } ,
7095 Name-sg = Eindnoot ,
7096 name-sg = eindnoot ,
7097 Name-pl = Eindnoten ,
7098 name-pl = eindnoten ,
7099
7100 type = note ,
7101   gender = f ,
7102 Name-sg = Opmerking ,
7103 name-sg = opmerking ,
7104 Name-pl = Opmerkingen ,
7105 name-pl = opmerkingen ,
7106
7107 type = equation ,
7108   gender = f ,
7109 Name-sg = Vergelijking ,
7110 name-sg = vergelijking ,
7111 Name-pl = Vergelijkingen ,
7112 name-pl = vergelijkingen ,
7113 Name-sg-ab = Vgl. ,
7114 name-sg-ab = vgl. ,
7115 Name-pl-ab = Vgl.'s ,
7116 name-pl-ab = vgl.'s ,
7117 refbounds-first-sg = { ,(,) } ,
7118 refbounds = { (,,) } ,
7119
7120 type = theorem ,
7121   gender = f ,
7122 Name-sg = Stelling ,
7123 name-sg = stelling ,
7124 Name-pl = Stellingen ,
7125 name-pl = stellingen ,
7126

```

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".

```

7127 type = lemma ,
7128   gender = n ,
7129 Name-sg = Lemma ,
7130 name-sg = lemma ,
7131 Name-pl = Lemma's ,
7132 name-pl = lemma's ,
7133
7134 type = corollary ,
7135   gender = n ,
7136 Name-sg = Gevolg ,
7137 name-sg = gevolg ,

```



```

7138 Name-pl = Gevolgen ,
7139 name-pl = gevolgen ,
7140
7141 type = proposition ,
7142 gender = f ,
7143 Name-sg = Propositie ,
7144 name-sg = propositie ,
7145 Name-pl = Proposities ,
7146 name-pl = proposities ,
7147
7148 type = definition ,
7149 gender = f ,
7150 Name-sg = Definitie ,
7151 name-sg = definitie ,
7152 Name-pl = Definities ,
7153 name-pl = definities ,
7154
7155 type = proof ,
7156 gender = n ,
7157 Name-sg = Bewijs ,
7158 name-sg = bewijs ,
7159 Name-pl = Bewijzen ,
7160 name-pl = bewijzen ,
7161
7162 type = result ,
7163 gender = n ,
7164 Name-sg = Resultaat ,
7165 name-sg = resultaat ,
7166 Name-pl = Resultaten ,
7167 name-pl = resultaten ,
7168
7169 type = remark ,
7170 gender = f ,
7171 Name-sg = Opmerking ,
7172 name-sg = opmerking ,
7173 Name-pl = Opmerkingen ,
7174 name-pl = opmerkingen ,
7175
7176 type = example ,
7177 gender = n ,
7178 Name-sg = Voorbeeld ,
7179 name-sg = voorbeeld ,
7180 Name-pl = Voorbeelden ,
7181 name-pl = voorbeelden ,
7182

```

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".

```

7183 type = algorithm ,
7184 gender = { n , f , m } ,
7185 Name-sg = Algoritme ,
7186 name-sg = algoritme ,
7187 Name-pl = Algoritmen ,
7188 name-pl = algoritmen ,

```

7189

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

```
7190 type = listing ,
7191   gender = m ,
7192   Name-sg = Listing ,
7193   name-sg = listing ,
7194   Name-pl = Listings ,
7195   name-pl = listings ,
7196
7197 type = exercise ,
7198   gender = { f , m } ,
7199   Name-sg = Opgave ,
7200   name-sg = opgave ,
7201   Name-pl = Opgaven ,
7202   name-pl = opgaven ,
7203
7204 type = solution ,
7205   gender = f ,
7206   Name-sg = Oplossing ,
7207   name-sg = oplossing ,
7208   Name-pl = Oplossingen ,
7209   name-pl = oplossingen ,
7210 </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->

```
7211  $\$ *package)
7212 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7213  $\$ /package)
7214  $\$ *lang-italian)
7215 namesep = {\nobreakspace} ,
7216 pairsep = {\~e\nobreakspace} ,
7217 listsep = { , ~ } ,
7218 lastsep = {\~e\nobreakspace} ,
7219 tpairsep = {\~e\nobreakspace} ,
7220 tlistsep = { , ~ } ,
7221 tlastsep = { , ~e\nobreakspace} ,
7222 notesep = { ~ } ,
7223 rangesep = {\~a\nobreakspace} ,
7224 +refbounds-rb = {da\nobreakspace,,} ,
7225
7226 type = book ,
7227   gender = m ,
7228   Name-sg = Libro ,
7229   name-sg = libro ,
7230   Name-pl = Libri ,
7231   name-pl = libri ,
```

```

7232
7233 type = part ,
7234     gender = f ,
7235     Name-sg = Parte ,
7236     name-sg = parte ,
7237     Name-pl = Parti ,
7238     name-pl = parti ,
7239
7240 type = chapter ,
7241     gender = m ,
7242     Name-sg = Capitolo ,
7243     name-sg = capitolo ,
7244     Name-pl = Capitoli ,
7245     name-pl = capitoli ,
7246
7247 type = section ,
7248     gender = m ,
7249     Name-sg = Paragrafo ,
7250     name-sg = paragrafo ,
7251     Name-pl = Paragrafi ,
7252     name-pl = paragrafi ,
7253
7254 type = paragraph ,
7255     gender = m ,
7256     Name-sg = Capoverso ,
7257     name-sg = capoverso ,
7258     Name-pl = Capoversi ,
7259     name-pl = capoversi ,
7260
7261 type = appendix ,
7262     gender = f ,
7263     Name-sg = Appendice ,
7264     name-sg = appendice ,
7265     Name-pl = Appendici ,
7266     name-pl = appendici ,
7267
7268 type = page ,
7269     gender = f ,
7270     Name-sg = Pagina ,
7271     name-sg = pagina ,
7272     Name-pl = Pagine ,
7273     name-pl = pagine ,
7274     Name-sg-ab = Pag. ,
7275     name-sg-ab = pag. ,
7276     Name-pl-ab = Pag. ,
7277     name-pl-ab = pag. ,
7278     rangesep = {\textendash} ,
7279     rangetopair = false ,
7280     +refbounds-rb = {,,} ,
7281
7282 type = line ,
7283     gender = f ,
7284     Name-sg = Riga ,
7285     name-sg = riga ,

```

```

7286 Name-pl = Righe ,
7287 name-pl = righe ,
7288
7289 type = figure ,
7290 gender = f ,
7291 Name-sg = Figura ,
7292 name-sg = figura ,
7293 Name-pl = Figure ,
7294 name-pl = figure ,
7295 Name-sg-ab = Fig. ,
7296 name-sg-ab = fig. ,
7297 Name-pl-ab = Fig. ,
7298 name-pl-ab = fig. ,
7299
7300 type = table ,
7301 gender = f ,
7302 Name-sg = Tabella ,
7303 name-sg = tabella ,
7304 Name-pl = Tabelle ,
7305 name-pl = tabelle ,
7306 Name-sg-ab = Tab. ,
7307 name-sg-ab = tab. ,
7308 Name-pl-ab = Tab. ,
7309 name-pl-ab = tab. ,
7310
7311 type = item ,
7312 gender = m ,
7313 Name-sg = Punto ,
7314 name-sg = punto ,
7315 Name-pl = Punti ,
7316 name-pl = punti ,
7317
7318 type = footnote ,
7319 gender = f ,
7320 Name-sg = Nota ,
7321 name-sg = nota ,
7322 Name-pl = Note ,
7323 name-pl = note ,
7324
7325 type = endnote ,
7326 gender = f ,
7327 Name-sg = Nota ,
7328 name-sg = nota ,
7329 Name-pl = Note ,
7330 name-pl = note ,
7331
7332 type = note ,
7333 gender = f ,
7334 Name-sg = Nota ,
7335 name-sg = nota ,
7336 Name-pl = Note ,
7337 name-pl = note ,
7338
7339 type = equation ,

```

```

7340 gender = f ,
7341 Name-sg = Equazione ,
7342 name-sg = equazione ,
7343 Name-pl = Equazioni ,
7344 name-pl = equazioni ,
7345 Name-sg-ab = Eq. ,
7346 name-sg-ab = eq. ,
7347 Name-pl-ab = Eq. ,
7348 name-pl-ab = eq. ,
7349 +refbounds-rb = {da\nobreakspace(,,)} ,
7350 refbounds-first-sg = {,(,)} ,
7351 refbounds = {(,,)} ,
7352
7353 type = theorem ,
7354 gender = m ,
7355 Name-sg = Teorema ,
7356 name-sg = teorema ,
7357 Name-pl = Teoremi ,
7358 name-pl = teoremi ,
7359
7360 type = lemma ,
7361 gender = m ,
7362 Name-sg = Lemma ,
7363 name-sg = lemma ,
7364 Name-pl = Lemmi ,
7365 name-pl = lemmi ,
7366
7367 type = corollary ,
7368 gender = m ,
7369 Name-sg = Corollario ,
7370 name-sg = corollario ,
7371 Name-pl = Corollari ,
7372 name-pl = corollari ,
7373
7374 type = proposition ,
7375 gender = f ,
7376 Name-sg = Proposizione ,
7377 name-sg = proposizione ,
7378 Name-pl = Proposizioni ,
7379 name-pl = proposizioni ,
7380
7381 type = definition ,
7382 gender = f ,
7383 Name-sg = Definizione ,
7384 name-sg = definizione ,
7385 Name-pl = Definizioni ,
7386 name-pl = definizioni ,
7387
7388 type = proof ,
7389 gender = f ,
7390 Name-sg = Dimostrazione ,
7391 name-sg = dimostrazione ,
7392 Name-pl = Dimostrazioni ,
7393 name-pl = dimostrazioni ,

```

```

7394
7395 type = result ,
7396     gender = m ,
7397     Name-sg = Risultato ,
7398     name-sg = risultato ,
7399     Name-pl = Risultati ,
7400     name-pl = risultati ,
7401
7402 type = remark ,
7403     gender = f ,
7404     Name-sg = Osservazione ,
7405     name-sg = osservazione ,
7406     Name-pl = Osservazioni ,
7407     name-pl = osservazioni ,
7408
7409 type = example ,
7410     gender = m ,
7411     Name-sg = Esempio ,
7412     name-sg = esempio ,
7413     Name-pl = Esempi ,
7414     name-pl = esempi ,
7415
7416 type = algorithm ,
7417     gender = m ,
7418     Name-sg = Algoritmo ,
7419     name-sg = algoritmo ,
7420     Name-pl = Algoritmi ,
7421     name-pl = algoritmi ,
7422
7423 type = listing ,
7424     gender = m ,
7425     Name-sg = Listato ,
7426     name-sg = listato ,
7427     Name-pl = Listati ,
7428     name-pl = listati ,
7429
7430 type = exercise ,
7431     gender = m ,
7432     Name-sg = Esercizio ,
7433     name-sg = esercizio ,
7434     Name-pl = Esercizi ,
7435     name-pl = esercizi ,
7436
7437 type = solution ,
7438     gender = f ,
7439     Name-sg = Soluzione ,
7440     name-sg = soluzione ,
7441     Name-pl = Soluzioni ,
7442     name-pl = soluzioni ,
7443 </lang-italian>

```

## 10.9 Russian

Russian language file initially contributed by Sergey Slyusarev ‘jemmybutton’ (PR #29). Russian localization is consistent with that of cleveref, with the following exceptions: “equation” is translated as “уравнение”, rather than “formula”, “proposition” is translated as “предложение”, rather than “утверждение”; several abbreviations are replaced with more common ones, e.g. abbreviated plural of “item” is “шт.”, not “п.п.”.

```
7444 \package
7445 \zcDeclareLanguage
7446 [ declension = { n , a , g , d , i , p } , gender = { f , m , n } ]
7447 { russian }
7448 \package

7449 \*lang-russian)

7450 namesep = {\nobreakspace} ,
7451 pairsep = {\~\nobreakspace} ,
7452 listsep = { ,\~ } ,
7453 lastsep = {\~\nobreakspace} ,
7454 tpairsep = {\~\nobreakspace} ,
7455 tlistsep = { ,\~ } ,
7456 tlastsep = { ,\~\nobreakspace} ,
7457 notesep = { } ,
7458 rangesep = {\~по\nobreakspace} ,
7459 +refbounds-rb = {c\nobreakspace,,} ,
7460
7461 type = book ,
7462 gender = f ,
7463 case = n ,
7464 Name-sg = Книга ,
7465 name-sg = книга ,
7466 Name-pl = Книги ,
7467 name-pl = книги ,
7468 case = a ,
7469 Name-sg = Книгу ,
7470 name-sg = книгу ,
7471 Name-pl = Книги ,
7472 name-pl = книги ,
7473 case = g ,
7474 Name-sg = Книги ,
7475 name-sg = книги ,
7476 Name-pl = Книг ,
7477 name-pl = книг ,
7478 case = d ,
7479 Name-sg = Книге ,
7480 name-sg = книге ,
7481 Name-pl = Книгам ,
7482 name-pl = книгам ,
7483 case = i ,
7484 Name-sg = Книгой ,
7485 name-sg = книгой ,
7486 Name-pl = Книгами ,
7487 name-pl = книгами ,
7488 case = p ,
7489 Name-sg = Книге ,
```

7490 name-sg = книге ,  
7491 Name-pl = Книгах ,  
7492 name-pl = книгах ,  
7493  
7494 type = part ,  
7495 gender = f ,  
7496 case = n ,  
7497 Name-sg = Часть ,  
7498 name-sg = часть ,  
7499 Name-pl = Части ,  
7500 name-pl = части ,  
7501 Name-sg-ab = Ч. ,  
7502 name-sg-ab = ч. ,  
7503 Name-pl-ab = Чч. ,  
7504 name-pl-ab = чч. ,  
7505 case = a ,  
7506 Name-sg = Часть ,  
7507 name-sg = часть ,  
7508 Name-pl = Части ,  
7509 name-pl = части ,  
7510 Name-sg-ab = Ч. ,  
7511 name-sg-ab = ч. ,  
7512 Name-pl-ab = Чч. ,  
7513 name-pl-ab = чч. ,  
7514 case = g ,  
7515 Name-sg = Части ,  
7516 name-sg = части ,  
7517 Name-pl = Частей ,  
7518 name-pl = частей ,  
7519 Name-sg-ab = Ч. ,  
7520 name-sg-ab = ч. ,  
7521 Name-pl-ab = Чч. ,  
7522 name-pl-ab = чч. ,  
7523 case = d ,  
7524 Name-sg = Части ,  
7525 name-sg = части ,  
7526 Name-pl = Частям ,  
7527 name-pl = частям ,  
7528 Name-sg-ab = Ч. ,  
7529 name-sg-ab = ч. ,  
7530 Name-pl-ab = Чч. ,  
7531 name-pl-ab = чч. ,  
7532 case = i ,  
7533 Name-sg = Частью ,  
7534 name-sg = частью ,  
7535 Name-pl = Частями ,  
7536 name-pl = частями ,  
7537 Name-sg-ab = Ч. ,  
7538 name-sg-ab = ч. ,  
7539 Name-pl-ab = Чч. ,  
7540 name-pl-ab = чч. ,  
7541 case = p ,  
7542 Name-sg = Части ,  
7543 name-sg = части ,



7544 Name-pl = Частях ,  
7545 name-pl = частях ,  
7546 Name-sg-ab = Ч. ,  
7547 name-sg-ab = ч. ,  
7548 Name-pl-ab = Чч. ,  
7549 name-pl-ab = чч. ,  
7550  
7551 type = chapter ,  
7552 gender = f ,  
7553 case = n ,  
7554 Name-sg = Глава ,  
7555 name-sg = глава ,  
7556 Name-pl = Главы ,  
7557 name-pl = главы ,  
7558 Name-sg-ab = Гл. ,  
7559 name-sg-ab = гл. ,  
7560 Name-pl-ab = Гл. ,  
7561 name-pl-ab = гл. ,  
7562 case = a ,  
7563 Name-sg = Главу ,  
7564 name-sg = главу ,  
7565 Name-pl = Главы ,  
7566 name-pl = главы ,  
7567 Name-sg-ab = Гл. ,  
7568 name-sg-ab = гл. ,  
7569 Name-pl-ab = Гл. ,  
7570 name-pl-ab = гл. ,  
7571 case = g ,  
7572 Name-sg = Главы ,  
7573 name-sg = главы ,  
7574 Name-pl = Глав ,  
7575 name-pl = глав ,  
7576 Name-sg-ab = Гл. ,  
7577 name-sg-ab = гл. ,  
7578 Name-pl-ab = Гл. ,  
7579 name-pl-ab = гл. ,  
7580 case = d ,  
7581 Name-sg = Главе ,  
7582 name-sg = главе ,  
7583 Name-pl = Главам ,  
7584 name-pl = главам ,  
7585 Name-sg-ab = Гл. ,  
7586 name-sg-ab = гл. ,  
7587 Name-pl-ab = Гл. ,  
7588 name-pl-ab = гл. ,  
7589 case = i ,  
7590 Name-sg = Главой ,  
7591 name-sg = главой ,  
7592 Name-pl = Главами ,  
7593 name-pl = главами ,  
7594 Name-sg-ab = Гл. ,  
7595 name-sg-ab = гл. ,  
7596 Name-pl-ab = Гл. ,  
7597 name-pl-ab = гл. ,

```

7598 case = p ,
7599     Name-sg = Главе ,
7600     name-sg = главе ,
7601     Name-pl = Главах ,
7602     name-pl = главах ,
7603     Name-sg-ab = Гл. ,
7604     name-sg-ab = гл. ,
7605     Name-pl-ab = Гл. ,
7606     name-pl-ab = гл. ,
7607
7608 type = section ,
7609     gender = m ,
7610     case = n ,
7611         Name-sg = Раздел ,
7612         name-sg = раздел ,
7613         Name-pl = Разделы ,
7614         name-pl = разделы ,
7615     case = a ,
7616         Name-sg = Раздел ,
7617         name-sg = раздел ,
7618         Name-pl = Разделы ,
7619         name-pl = разделы ,
7620     case = g ,
7621         Name-sg = Раздела ,
7622         name-sg = раздела ,
7623         Name-pl = Разделов ,
7624         name-pl = разделов ,
7625     case = d ,
7626         Name-sg = Разделу ,
7627         name-sg = разделу ,
7628         Name-pl = Разделам ,
7629         name-pl = разделам ,
7630     case = i ,
7631         Name-sg = Разделом ,
7632         name-sg = разделом ,
7633         Name-pl = Разделами ,
7634         name-pl = разделами ,
7635     case = p ,
7636         Name-sg = Разделе ,
7637         name-sg = разделе ,
7638         Name-pl = Разделах ,
7639         name-pl = разделах ,
7640
7641 type = paragraph ,
7642     gender = m ,
7643     case = n ,
7644         Name-sg = Абзац ,
7645         name-sg = абзац ,
7646         Name-pl = Абзацы ,
7647         name-pl = абзацы ,
7648     case = a ,
7649         Name-sg = Абзац ,
7650         name-sg = абзац ,
7651         Name-pl = Абзацы ,

```

7652 name-pl = абзацы ,  
7653 case = g ,  
7654 Name-sg = Абзаца ,  
7655 name-sg = абзаца ,  
7656 Name-pl = Абзацев ,  
7657 name-pl = абзацев ,  
7658 case = d ,  
7659 Name-sg = Абзацу ,  
7660 name-sg = абзацу ,  
7661 Name-pl = Абзацам ,  
7662 name-pl = абзацам ,  
7663 case = i ,  
7664 Name-sg = Абзацем ,  
7665 name-sg = абзацем ,  
7666 Name-pl = Абзацами ,  
7667 name-pl = абзацами ,  
7668 case = p ,  
7669 Name-sg = Абзаце ,  
7670 name-sg = абзаце ,  
7671 Name-pl = Абзацах ,  
7672 name-pl = абзацах ,  
7673  
7674 type = appendix ,  
7675 gender = n ,  
7676 case = n ,  
7677 Name-sg = Приложение ,  
7678 name-sg = приложение ,  
7679 Name-pl = Приложения ,  
7680 name-pl = приложения ,  
7681 case = a ,  
7682 Name-sg = Приложение ,  
7683 name-sg = приложение ,  
7684 Name-pl = Приложения ,  
7685 name-pl = приложения ,  
7686 case = g ,  
7687 Name-sg = Приложения ,  
7688 name-sg = приложения ,  
7689 Name-pl = Приложений ,  
7690 name-pl = приложений ,  
7691 case = d ,  
7692 Name-sg = Приложению ,  
7693 name-sg = приложению ,  
7694 Name-pl = Приложениям ,  
7695 name-pl = приложениям ,  
7696 case = i ,  
7697 Name-sg = Приложением ,  
7698 name-sg = приложением ,  
7699 Name-pl = Приложениями ,  
7700 name-pl = приложениями ,  
7701 case = p ,  
7702 Name-sg = Приложении ,  
7703 name-sg = приложении ,  
7704 Name-pl = Приложениях ,  
7705 name-pl = приложениях ,

7706  
7707 type = page ,  
7708 gender = f ,  
7709 case = n ,  
7710 Name-sg = Страница ,  
7711 name-sg = страница ,  
7712 Name-pl = Страницы ,  
7713 name-pl = страницы ,  
7714 Name-sg-ab = С. ,  
7715 name-sg-ab = с. ,  
7716 Name-pl-ab = Сс. ,  
7717 name-pl-ab = сс. ,  
7718 case = a ,  
7719 Name-sg = Страницу ,  
7720 name-sg = страницу ,  
7721 Name-pl = Страницы ,  
7722 name-pl = страницы ,  
7723 Name-sg-ab = С. ,  
7724 name-sg-ab = с. ,  
7725 Name-pl-ab = Сс. ,  
7726 name-pl-ab = сс. ,  
7727 case = g ,  
7728 Name-sg = Страницы ,  
7729 name-sg = страницы ,  
7730 Name-pl = Страниц ,  
7731 name-pl = страниц ,  
7732 Name-sg-ab = С. ,  
7733 name-sg-ab = с. ,  
7734 Name-pl-ab = Сс. ,  
7735 name-pl-ab = сс. ,  
7736 case = d ,  
7737 Name-sg = Странице ,  
7738 name-sg = странице ,  
7739 Name-pl = Страницам ,  
7740 name-pl = страницам ,  
7741 Name-sg-ab = С. ,  
7742 name-sg-ab = с. ,  
7743 Name-pl-ab = Сс. ,  
7744 name-pl-ab = сс. ,  
7745 case = i ,  
7746 Name-sg = Страницей ,  
7747 name-sg = страницей ,  
7748 Name-pl = Страницами ,  
7749 name-pl = страницами ,  
7750 Name-sg-ab = С. ,  
7751 name-sg-ab = с. ,  
7752 Name-pl-ab = Сс. ,  
7753 name-pl-ab = сс. ,  
7754 case = p ,  
7755 Name-sg = Странице ,  
7756 name-sg = странице ,  
7757 Name-pl = Страницах ,  
7758 name-pl = страницах ,  
7759 Name-sg-ab = С. ,

```

7760     name-sg-ab = с. ,
7761     Name-pl-ab = Сс. ,
7762     name-pl-ab = сс. ,
7763     rangesep = {\textendash} ,
7764     rangetopair = false ,
7765     +refbounds-rb = {,,} ,
7766
7767 type = line ,
7768     gender = f ,
7769     case = n ,
7770     Name-sg = Строка ,
7771     name-sg = строка ,
7772     Name-pl = Строки ,
7773     name-pl = строки ,
7774     case = a ,
7775     Name-sg = Строку ,
7776     name-sg = строку ,
7777     Name-pl = Строки ,
7778     name-pl = строки ,
7779     case = g ,
7780     Name-sg = Строки ,
7781     name-sg = строки ,
7782     Name-pl = Строк ,
7783     name-pl = строк ,
7784     case = d ,
7785     Name-sg = Строке ,
7786     name-sg = строке ,
7787     Name-pl = Строкам ,
7788     name-pl = строкам ,
7789     case = i ,
7790     Name-sg = Строкой ,
7791     name-sg = строкой ,
7792     Name-pl = Строками ,
7793     name-pl = строками ,
7794     case = p ,
7795     Name-sg = Строке ,
7796     name-sg = строке ,
7797     Name-pl = Строках ,
7798     name-pl = строках ,
7799
7800 type = figure ,
7801     gender = m ,
7802     case = n ,
7803     Name-sg = Рисунок ,
7804     name-sg = рисунок ,
7805     Name-pl = Рисунки ,
7806     name-pl = рисунки ,
7807     Name-sg-ab = Рис. ,
7808     name-sg-ab = рис. ,
7809     Name-pl-ab = Рис. ,
7810     name-pl-ab = рис. ,
7811     case = a ,
7812     Name-sg = Рисунок ,
7813     name-sg = рисунок ,

```

7814 Name-pl = Рисунки ,  
7815 name-pl = рисунки ,  
7816 Name-sg-ab = Рис. ,  
7817 name-sg-ab = рис. ,  
7818 Name-pl-ab = Рис. ,  
7819 name-pl-ab = рис. ,  
7820 case = g ,  
7821 Name-sg = Рисунок ,  
7822 name-sg = рисунок ,  
7823 Name-pl = Рисунков ,  
7824 name-pl = рисунков ,  
7825 Name-sg-ab = Рис. ,  
7826 name-sg-ab = рис. ,  
7827 Name-pl-ab = Рис. ,  
7828 name-pl-ab = рис. ,  
7829 case = d ,  
7830 Name-sg = Рисунку ,  
7831 name-sg = рисунку ,  
7832 Name-pl = Рисункам ,  
7833 name-pl = рисункам ,  
7834 Name-sg-ab = Рис. ,  
7835 name-sg-ab = рис. ,  
7836 Name-pl-ab = Рис. ,  
7837 name-pl-ab = рис. ,  
7838 case = i ,  
7839 Name-sg = Рисунком ,  
7840 name-sg = рисунком ,  
7841 Name-pl = Рисунками ,  
7842 name-pl = рисунками ,  
7843 Name-sg-ab = Рис. ,  
7844 name-sg-ab = рис. ,  
7845 Name-pl-ab = Рис. ,  
7846 name-pl-ab = рис. ,  
7847 case = p ,  
7848 Name-sg = Рисунке ,  
7849 name-sg = рисунке ,  
7850 Name-pl = Рисунках ,  
7851 name-pl = рисунках ,  
7852 Name-sg-ab = Рис. ,  
7853 name-sg-ab = рис. ,  
7854 Name-pl-ab = Рис. ,  
7855 name-pl-ab = рис. ,  
7856  
7857 type = table ,  
7858 gender = f ,  
7859 case = n ,  
7860 Name-sg = Таблица ,  
7861 name-sg = таблица ,  
7862 Name-pl = Таблицы ,  
7863 name-pl = таблицы ,  
7864 Name-sg-ab = Табл. ,  
7865 name-sg-ab = табл. ,  
7866 Name-pl-ab = Табл. ,  
7867 name-pl-ab = табл. ,

7868 case = a ,  
7869 Name-sg = Таблицу ,  
7870 name-sg = таблицу ,  
7871 Name-pl = Таблицы ,  
7872 name-pl = таблицы ,  
7873 Name-sg-ab = Табл. ,  
7874 name-sg-ab = табл. ,  
7875 Name-pl-ab = Табл. ,  
7876 name-pl-ab = табл. ,  
7877 case = g ,  
7878 Name-sg = Таблицы ,  
7879 name-sg = таблицы ,  
7880 Name-pl = Таблиц ,  
7881 name-pl = таблиц ,  
7882 Name-sg-ab = Табл. ,  
7883 name-sg-ab = табл. ,  
7884 Name-pl-ab = Табл. ,  
7885 name-pl-ab = табл. ,  
7886 case = d ,  
7887 Name-sg = Таблице ,  
7888 name-sg = таблице ,  
7889 Name-pl = Таблицам ,  
7890 name-pl = таблицам ,  
7891 Name-sg-ab = Табл. ,  
7892 name-sg-ab = табл. ,  
7893 Name-pl-ab = Табл. ,  
7894 name-pl-ab = табл. ,  
7895 case = i ,  
7896 Name-sg = Таблицей ,  
7897 name-sg = таблицей ,  
7898 Name-pl = Таблицами ,  
7899 name-pl = таблицами ,  
7900 Name-sg-ab = Табл. ,  
7901 name-sg-ab = табл. ,  
7902 Name-pl-ab = Табл. ,  
7903 name-pl-ab = табл. ,  
7904 case = p ,  
7905 Name-sg = Таблице ,  
7906 name-sg = таблице ,  
7907 Name-pl = Таблицах ,  
7908 name-pl = таблицах ,  
7909 Name-sg-ab = Табл. ,  
7910 name-sg-ab = табл. ,  
7911 Name-pl-ab = Табл. ,  
7912 name-pl-ab = табл. ,  
7913  
7914 type = item ,  
7915 gender = m ,  
7916 case = n ,  
7917 Name-sg = Пункт ,  
7918 name-sg = пункт ,  
7919 Name-pl = Пункты ,  
7920 name-pl = пункты ,  
7921 Name-sg-ab = П. ,

7922 name-sg-ab = п. ,  
7923 Name-pl-ab = Пп. ,  
7924 name-pl-ab = пп. ,  
7925 case = a ,  
7926 Name-sg = Пункт ,  
7927 name-sg = пункт ,  
7928 Name-pl = Пункты ,  
7929 name-pl = пункты ,  
7930 Name-sg-ab = П. ,  
7931 name-sg-ab = п. ,  
7932 Name-pl-ab = Пп. ,  
7933 name-pl-ab = пп. ,  
7934 case = g ,  
7935 Name-sg = Пункта ,  
7936 name-sg = пункта ,  
7937 Name-pl = Пунктов ,  
7938 name-pl = пунктов ,  
7939 Name-sg-ab = П. ,  
7940 name-sg-ab = п. ,  
7941 Name-pl-ab = Пп. ,  
7942 name-pl-ab = пп. ,  
7943 case = d ,  
7944 Name-sg = Пункту ,  
7945 name-sg = пункту ,  
7946 Name-pl = Пунктам ,  
7947 name-pl = пунктам ,  
7948 Name-sg-ab = П. ,  
7949 name-sg-ab = п. ,  
7950 Name-pl-ab = Пп. ,  
7951 name-pl-ab = пп. ,  
7952 case = i ,  
7953 Name-sg = Пунктом ,  
7954 name-sg = пунктом ,  
7955 Name-pl = Пунктами ,  
7956 name-pl = пунктами ,  
7957 Name-sg-ab = П. ,  
7958 name-sg-ab = п. ,  
7959 Name-pl-ab = Пп. ,  
7960 name-pl-ab = пп. ,  
7961 case = p ,  
7962 Name-sg = Пункте ,  
7963 name-sg = пункте ,  
7964 Name-pl = Пунктах ,  
7965 name-pl = пунктах ,  
7966 Name-sg-ab = П. ,  
7967 name-sg-ab = п. ,  
7968 Name-pl-ab = Пп. ,  
7969 name-pl-ab = пп. ,  
7970  
7971 type = footnote ,  
7972 gender = f ,  
7973 case = n ,  
7974 Name-sg = Сноска ,  
7975 name-sg = сноска ,



7976 Name-pl = Сноски ,  
7977 name-pl = сноски ,  
7978 case = a ,  
7979 Name-sg = Сноску ,  
7980 name-sg = сноску ,  
7981 Name-pl = Сноски ,  
7982 name-pl = сноски ,  
7983 case = g ,  
7984 Name-sg = Сноски ,  
7985 name-sg = сноски ,  
7986 Name-pl = Сносок ,  
7987 name-pl = сносок ,  
7988 case = d ,  
7989 Name-sg = Сноске ,  
7990 name-sg = сноске ,  
7991 Name-pl = Сноскам ,  
7992 name-pl = сноскам ,  
7993 case = i ,  
7994 Name-sg = Сноской ,  
7995 name-sg = сноской ,  
7996 Name-pl = Сносками ,  
7997 name-pl = сносками ,  
7998 case = p ,  
7999 Name-sg = Сноске ,  
8000 name-sg = сноске ,  
8001 Name-pl = Сносках ,  
8002 name-pl = сносках ,  
8003  
8004 type = endnote ,  
8005 gender = f ,  
8006 case = n ,  
8007 Name-sg = Сноска ,  
8008 name-sg = сноска ,  
8009 Name-pl = Сноски ,  
8010 name-pl = сноски ,  
8011 case = a ,  
8012 Name-sg = Сноску ,  
8013 name-sg = сноску ,  
8014 Name-pl = Сноски ,  
8015 name-pl = сноски ,  
8016 case = g ,  
8017 Name-sg = Сноски ,  
8018 name-sg = сноски ,  
8019 Name-pl = Сносок ,  
8020 name-pl = сносок ,  
8021 case = d ,  
8022 Name-sg = Сноске ,  
8023 name-sg = сноске ,  
8024 Name-pl = Сноскам ,  
8025 name-pl = сноскам ,  
8026 case = i ,  
8027 Name-sg = Сноской ,  
8028 name-sg = сноской ,  
8029 Name-pl = Сносками ,

```

8030     name-pl = сносками ,
8031     case = p ,
8032     Name-sg = Сноске ,
8033     name-sg = сноске ,
8034     Name-pl = Сносках ,
8035     name-pl = сносках ,
8036
8037     type = note ,
8038     gender = f ,
8039     case = n ,
8040     Name-sg = Заметка ,
8041     name-sg = заметка ,
8042     Name-pl = Заметки ,
8043     name-pl = заметки ,
8044     case = a ,
8045     Name-sg = Заметку ,
8046     name-sg = заметку ,
8047     Name-pl = Заметки ,
8048     name-pl = заметки ,
8049     case = g ,
8050     Name-sg = Заметки ,
8051     name-sg = заметки ,
8052     Name-pl = Заметок ,
8053     name-pl = заметок ,
8054     case = d ,
8055     Name-sg = Заметке ,
8056     name-sg = заметке ,
8057     Name-pl = Заметкам ,
8058     name-pl = заметкам ,
8059     case = i ,
8060     Name-sg = Заметкой ,
8061     name-sg = заметкой ,
8062     Name-pl = Заметками ,
8063     name-pl = заметками ,
8064     case = p ,
8065     Name-sg = Заметке ,
8066     name-sg = заметке ,
8067     Name-pl = Заметках ,
8068     name-pl = заметках ,
8069
8070     type = equation ,
8071     gender = n ,
8072     case = n ,
8073     Name-sg = Уравнение ,
8074     name-sg = уравнение ,
8075     Name-pl = Уравнения ,
8076     name-pl = уравнения ,
8077     Name-sg-ab = Ур. ,
8078     name-sg-ab = ур. ,
8079     Name-pl-ab = Ур. ,
8080     name-pl-ab = ур. ,
8081     case = a ,
8082     Name-sg = Уравнение ,
8083     name-sg = уравнение ,

```

```

8084     Name-pl = Уравнения ,
8085     name-pl = уравнения ,
8086     Name-sg-ab = Ур. ,
8087     name-sg-ab = ур. ,
8088     Name-pl-ab = Ур. ,
8089     name-pl-ab = ур. ,
8090     case = g ,
8091     Name-sg = Уравнения ,
8092     name-sg = уравнения ,
8093     Name-pl = Уравнений ,
8094     name-pl = уравнений ,
8095     Name-sg-ab = Ур. ,
8096     name-sg-ab = ур. ,
8097     Name-pl-ab = Ур. ,
8098     name-pl-ab = ур. ,
8099     case = d ,
8100     Name-sg = Уравнению ,
8101     name-sg = уравнению ,
8102     Name-pl = Уравнениям ,
8103     name-pl = уравнениям ,
8104     Name-sg-ab = Ур. ,
8105     name-sg-ab = ур. ,
8106     Name-pl-ab = Ур. ,
8107     name-pl-ab = ур. ,
8108     case = i ,
8109     Name-sg = Уравнением ,
8110     name-sg = уравнением ,
8111     Name-pl = Уравнениями ,
8112     name-pl = уравнениями ,
8113     Name-sg-ab = Ур. ,
8114     name-sg-ab = ур. ,
8115     Name-pl-ab = Ур. ,
8116     name-pl-ab = ур. ,
8117     case = p ,
8118     Name-sg = Уравнении ,
8119     name-sg = уравнении ,
8120     Name-pl = Уравнениях ,
8121     name-pl = уравнениях ,
8122     Name-sg-ab = Ур. ,
8123     name-sg-ab = ур. ,
8124     Name-pl-ab = Ур. ,
8125     name-pl-ab = ур. ,
8126     +refbounds-rb = {c\nobreakspace(,,)} ,
8127     refbounds-first-sg = {(,)}, ,
8128     refbounds = {(,,)} ,
8129
8130     type = theorem ,
8131     gender = f ,
8132     case = n ,
8133     Name-sg = Теорема ,
8134     name-sg = теорема ,
8135     Name-pl = Теоремы ,
8136     name-pl = теоремы ,
8137     Name-sg-ab = Теор. ,

```

8138 name-sg-ab = теор. ,  
8139 Name-pl-ab = Теор. ,  
8140 name-pl-ab = теор. ,  
8141 case = a ,  
8142 Name-sg = Теорему ,  
8143 name-sg = теорему ,  
8144 Name-pl = Теоремы ,  
8145 name-pl = теоремы ,  
8146 Name-sg-ab = Теор. ,  
8147 name-sg-ab = теор. ,  
8148 Name-pl-ab = Теор. ,  
8149 name-pl-ab = теор. ,  
8150 case = g ,  
8151 Name-sg = Теоремы ,  
8152 name-sg = теоремы ,  
8153 Name-pl = Теорем ,  
8154 name-pl = теорем ,  
8155 Name-sg-ab = Теор. ,  
8156 name-sg-ab = теор. ,  
8157 Name-pl-ab = Теор. ,  
8158 name-pl-ab = теор. ,  
8159 case = d ,  
8160 Name-sg = Теореме ,  
8161 name-sg = теореме ,  
8162 Name-pl = Теоремам ,  
8163 name-pl = теоремам ,  
8164 Name-sg-ab = Теор. ,  
8165 name-sg-ab = теор. ,  
8166 Name-pl-ab = Теор. ,  
8167 name-pl-ab = теор. ,  
8168 case = i ,  
8169 Name-sg = Теоремой ,  
8170 name-sg = теоремой ,  
8171 Name-pl = Теоремами ,  
8172 name-pl = теоремами ,  
8173 Name-sg-ab = Теор. ,  
8174 name-sg-ab = теор. ,  
8175 Name-pl-ab = Теор. ,  
8176 name-pl-ab = теор. ,  
8177 case = p ,  
8178 Name-sg = Теореме ,  
8179 name-sg = теореме ,  
8180 Name-pl = Теоремах ,  
8181 name-pl = теоремах ,  
8182 Name-sg-ab = Теор. ,  
8183 name-sg-ab = теор. ,  
8184 Name-pl-ab = Теор. ,  
8185 name-pl-ab = теор. ,  
8186  
8187 type = lemma ,  
8188 gender = f ,  
8189 case = n ,  
8190 Name-sg = Лемма ,  
8191 name-sg = лемма ,

8192 Name-pl = Леммы ,  
8193 name-pl = леммы ,  
8194 case = a ,  
8195 Name-sg = Лемму ,  
8196 name-sg = лемму ,  
8197 Name-pl = Леммы ,  
8198 name-pl = леммы ,  
8199 case = g ,  
8200 Name-sg = Леммы ,  
8201 name-sg = леммы ,  
8202 Name-pl = Лемм ,  
8203 name-pl = лемм ,  
8204 case = d ,  
8205 Name-sg = Лемме ,  
8206 name-sg = лемме ,  
8207 Name-pl = Леммам ,  
8208 name-pl = леммам ,  
8209 case = i ,  
8210 Name-sg = Леммой ,  
8211 name-sg = леммой ,  
8212 Name-pl = Леммами ,  
8213 name-pl = леммами ,  
8214 case = p ,  
8215 Name-sg = Лемме ,  
8216 name-sg = лемме ,  
8217 Name-pl = Леммах ,  
8218 name-pl = леммах ,  
8219  
8220 type = corollary ,  
8221 gender = m ,  
8222 case = n ,  
8223 Name-sg = Вывод ,  
8224 name-sg = вывод ,  
8225 Name-pl = Выводы ,  
8226 name-pl = выводы ,  
8227 case = a ,  
8228 Name-sg = Вывод ,  
8229 name-sg = вывод ,  
8230 Name-pl = Выводы ,  
8231 name-pl = выводы ,  
8232 case = g ,  
8233 Name-sg = Вывода ,  
8234 name-sg = вывода ,  
8235 Name-pl = Выводов ,  
8236 name-pl = выводов ,  
8237 case = d ,  
8238 Name-sg = Выводу ,  
8239 name-sg = выводу ,  
8240 Name-pl = Выводам ,  
8241 name-pl = выводам ,  
8242 case = i ,  
8243 Name-sg = Выводом ,  
8244 name-sg = выводом ,  
8245 Name-pl = Выводами ,

8246       name-pl = выводами ,  
8247       case = p ,  
8248       Name-sg = Выводе ,  
8249       name-sg = выводе ,  
8250       Name-pl = Выводах ,  
8251       name-pl = выводах ,  
8252  
8253       type = proposition ,  
8254       gender = n ,  
8255       case = n ,  
8256       Name-sg = Предложение ,  
8257       name-sg = предложение ,  
8258       Name-pl = Предложения ,  
8259       name-pl = предложения ,  
8260       Name-sg-ab = Предл. ,  
8261       name-sg-ab = предл. ,  
8262       Name-pl-ab = Предл. ,  
8263       name-pl-ab = предл. ,  
8264       case = a ,  
8265       Name-sg = Предложение ,  
8266       name-sg = предложение ,  
8267       Name-pl = Предложения ,  
8268       name-pl = предложения ,  
8269       Name-sg-ab = Предл. ,  
8270       name-sg-ab = предл. ,  
8271       Name-pl-ab = Предл. ,  
8272       name-pl-ab = предл. ,  
8273       case = g ,  
8274       Name-sg = Предложения ,  
8275       name-sg = предложения ,  
8276       Name-pl = Предложений ,  
8277       name-pl = предложений ,  
8278       Name-sg-ab = Предл. ,  
8279       name-sg-ab = предл. ,  
8280       Name-pl-ab = Предл. ,  
8281       name-pl-ab = предл. ,  
8282       case = d ,  
8283       Name-sg = Предложению ,  
8284       name-sg = предложению ,  
8285       Name-pl = Предложениям ,  
8286       name-pl = предложениям ,  
8287       Name-sg-ab = Предл. ,  
8288       name-sg-ab = предл. ,  
8289       Name-pl-ab = Предл. ,  
8290       name-pl-ab = предл. ,  
8291       case = i ,  
8292       Name-sg = Предложением ,  
8293       name-sg = предложением ,  
8294       Name-pl = Предложениями ,  
8295       name-pl = предложениями ,  
8296       Name-sg-ab = Предл. ,  
8297       name-sg-ab = предл. ,  
8298       Name-pl-ab = Предл. ,  
8299       name-pl-ab = предл. ,

8300 case = p ,  
8301 Name-sg = Предложенияи ,  
8302 name-sg = предложенияи ,  
8303 Name-pl = Предложенияих ,  
8304 name-pl = предложенияих ,  
8305 Name-sg-ab = Предл. ,  
8306 name-sg-ab = предл. ,  
8307 Name-pl-ab = Предл. ,  
8308 name-pl-ab = предл. ,  
8309  
8310 type = definition ,  
8311 gender = n ,  
8312 case = n ,  
8313 Name-sg = Определение ,  
8314 name-sg = определение ,  
8315 Name-pl = Определения ,  
8316 name-pl = определения ,  
8317 Name-sg-ab = Опр. ,  
8318 name-sg-ab = опр. ,  
8319 Name-pl-ab = Опр. ,  
8320 name-pl-ab = опр. ,  
8321 case = a ,  
8322 Name-sg = Определение ,  
8323 name-sg = определение ,  
8324 Name-pl = Определения ,  
8325 name-pl = определения ,  
8326 Name-sg-ab = Опр. ,  
8327 name-sg-ab = опр. ,  
8328 Name-pl-ab = Опр. ,  
8329 name-pl-ab = опр. ,  
8330 case = g ,  
8331 Name-sg = Определения ,  
8332 name-sg = определений ,  
8333 Name-pl = Определений ,  
8334 name-pl = определений ,  
8335 Name-sg-ab = Опр. ,  
8336 name-sg-ab = опр. ,  
8337 Name-pl-ab = Опр. ,  
8338 name-pl-ab = опр. ,  
8339 case = d ,  
8340 Name-sg = Определению ,  
8341 name-sg = определению ,  
8342 Name-pl = Определениям ,  
8343 name-pl = определениям ,  
8344 Name-sg-ab = Опр. ,  
8345 name-sg-ab = опр. ,  
8346 Name-pl-ab = Опр. ,  
8347 name-pl-ab = опр. ,  
8348 case = i ,  
8349 Name-sg = Определением ,  
8350 name-sg = определением ,  
8351 Name-pl = Определениями ,  
8352 name-pl = определениями ,  
8353 Name-sg-ab = Опр. ,

8354 name-sg-ab = опр. ,  
8355 Name-pl-ab = Опр. ,  
8356 name-pl-ab = опр. ,  
8357 case = p ,  
8358 Name-sg = Определении ,  
8359 name-sg = определении ,  
8360 Name-pl = Определениях ,  
8361 name-pl = определениях ,  
8362 Name-sg-ab = Опр. ,  
8363 name-sg-ab = опр. ,  
8364 Name-pl-ab = Опр. ,  
8365 name-pl-ab = опр. ,  
8366  
8367 type = proof ,  
8368 gender = n ,  
8369 case = n ,  
8370 Name-sg = Доказательство ,  
8371 name-sg = доказательство ,  
8372 Name-pl = Доказательства ,  
8373 name-pl = доказательства ,  
8374 case = a ,  
8375 Name-sg = Доказательство ,  
8376 name-sg = доказательство ,  
8377 Name-pl = Доказательства ,  
8378 name-pl = доказательства ,  
8379 case = g ,  
8380 Name-sg = Доказательства ,  
8381 name-sg = доказательства ,  
8382 Name-pl = Доказательств ,  
8383 name-pl = доказательств ,  
8384 case = d ,  
8385 Name-sg = Доказательству ,  
8386 name-sg = доказательству ,  
8387 Name-pl = Доказательствам ,  
8388 name-pl = доказательствам ,  
8389 case = i ,  
8390 Name-sg = Доказательством ,  
8391 name-sg = доказательством ,  
8392 Name-pl = Доказательствами ,  
8393 name-pl = доказательствами ,  
8394 case = p ,  
8395 Name-sg = Доказательстве ,  
8396 name-sg = доказательстве ,  
8397 Name-pl = Доказательствах ,  
8398 name-pl = доказательствах ,  
8399  
8400 type = result ,  
8401 gender = m ,  
8402 case = n ,  
8403 Name-sg = Результат ,  
8404 name-sg = результат ,  
8405 Name-pl = Результаты ,  
8406 name-pl = результаты ,  
8407 case = a ,



8408 Name-sg = Результат ,  
 8409 name-sg = результат ,  
 8410 Name-pl = Результаты ,  
 8411 name-pl = результаты ,  
 8412 case = g ,  
 8413 Name-sg = Результата ,  
 8414 name-sg = результата ,  
 8415 Name-pl = Результатов ,  
 8416 name-pl = результатов ,  
 8417 case = d ,  
 8418 Name-sg = Результату ,  
 8419 name-sg = результату ,  
 8420 Name-pl = Результатам ,  
 8421 name-pl = результатам ,  
 8422 case = i ,  
 8423 Name-sg = Результатом ,  
 8424 name-sg = результатом ,  
 8425 Name-pl = Результатами ,  
 8426 name-pl = результатами ,  
 8427 case = p ,  
 8428 Name-sg = Результате ,  
 8429 name-sg = результате ,  
 8430 Name-pl = Результатах ,  
 8431 name-pl = результатах ,  
 8432  
 8433 type = remark ,  
 8434 gender = n ,  
 8435 case = n ,  
 8436 Name-sg = Примечание ,  
 8437 name-sg = примечание ,  
 8438 Name-pl = Примечания ,  
 8439 name-pl = примечания ,  
 8440 Name-sg-ab = Прим. ,  
 8441 name-sg-ab = прим. ,  
 8442 Name-pl-ab = Прим. ,  
 8443 name-pl-ab = прим. ,  
 8444 case = a ,  
 8445 Name-sg = Примечание ,  
 8446 name-sg = примечание ,  
 8447 Name-pl = Примечания ,  
 8448 name-pl = примечания ,  
 8449 Name-sg-ab = Прим. ,  
 8450 name-sg-ab = прим. ,  
 8451 Name-pl-ab = Прим. ,  
 8452 name-pl-ab = прим. ,  
 8453 case = g ,  
 8454 Name-sg = Примечания ,  
 8455 name-sg = примечания ,  
 8456 Name-pl = Примечаний ,  
 8457 name-pl = примечаний ,  
 8458 Name-sg-ab = Прим. ,  
 8459 name-sg-ab = прим. ,  
 8460 Name-pl-ab = Прим. ,  
 8461 name-pl-ab = прим. ,

```

8462 case = d ,
8463     Name-sg = Примечанию ,
8464     name-sg = примечанию ,
8465     Name-pl = Примечаниям ,
8466     name-pl = примечаниям ,
8467     Name-sg-ab = Прим. ,
8468     name-sg-ab = прим. ,
8469     Name-pl-ab = Прим. ,
8470     name-pl-ab = прим. ,
8471 case = i ,
8472     Name-sg = Примечанием ,
8473     name-sg = примечанием ,
8474     Name-pl = Примечаниями ,
8475     name-pl = примечаниями ,
8476     Name-sg-ab = Прим. ,
8477     name-sg-ab = прим. ,
8478     Name-pl-ab = Прим. ,
8479     name-pl-ab = прим. ,
8480 case = p ,
8481     Name-sg = Примечании ,
8482     name-sg = примечании ,
8483     Name-pl = Примечаниях ,
8484     name-pl = примечаниях ,
8485     Name-sg-ab = Прим. ,
8486     name-sg-ab = прим. ,
8487     Name-pl-ab = Прим. ,
8488     name-pl-ab = прим. ,
8489
8490 type = example ,
8491     gender = m ,
8492     case = n ,
8493     Name-sg = Пример ,
8494     name-sg = пример ,
8495     Name-pl = Примеры ,
8496     name-pl = примеры ,
8497     case = a ,
8498     Name-sg = Пример ,
8499     name-sg = пример ,
8500     Name-pl = Примеры ,
8501     name-pl = примеры ,
8502     case = g ,
8503     Name-sg = Примера ,
8504     name-sg = примера ,
8505     Name-pl = Примеров ,
8506     name-pl = примеров ,
8507     case = d ,
8508     Name-sg = Примеру ,
8509     name-sg = примеру ,
8510     Name-pl = Примерам ,
8511     name-pl = примерам ,
8512     case = i ,
8513     Name-sg = Примером ,
8514     name-sg = примером ,
8515     Name-pl = Примерами ,

```

```

8516     name-pl = примерами ,
8517     case = p ,
8518     Name-sg = Примере ,
8519     name-sg = примере ,
8520     Name-pl = Примерах ,
8521     name-pl = примерах ,
8522
8523 type = algorithm ,
8524     gender = m ,
8525     case = n ,
8526     Name-sg = Алгоритм ,
8527     name-sg = алгоритм ,
8528     Name-pl = Алгоритмы ,
8529     name-pl = алгоритмы ,
8530     case = a ,
8531     Name-sg = Алгоритм ,
8532     name-sg = алгоритм ,
8533     Name-pl = Алгоритмы ,
8534     name-pl = алгоритмы ,
8535     case = g ,
8536     Name-sg = Алгоритма ,
8537     name-sg = алгоритма ,
8538     Name-pl = Алгоритмов ,
8539     name-pl = алгоритмов ,
8540     case = d ,
8541     Name-sg = Алгоритму ,
8542     name-sg = алгоритму ,
8543     Name-pl = Алгоритмам ,
8544     name-pl = алгоритмам ,
8545     case = i ,
8546     Name-sg = Алгоритмом ,
8547     name-sg = алгоритмом ,
8548     Name-pl = Алгоритмами ,
8549     name-pl = алгоритмами ,
8550     case = p ,
8551     Name-sg = Алгоритме ,
8552     name-sg = алгоритме ,
8553     Name-pl = Алгоритмах ,
8554     name-pl = алгоритмах ,
8555
8556 type = listing ,
8557     gender = m ,
8558     case = n ,
8559     Name-sg = Листинг ,
8560     name-sg = листинг ,
8561     Name-pl = Листинги ,
8562     name-pl = листинги ,
8563     case = a ,
8564     Name-sg = Листинг ,
8565     name-sg = листинг ,
8566     Name-pl = Листинги ,
8567     name-pl = листинги ,
8568     case = g ,
8569     Name-sg = Листинга ,

```

8570 name-sg = листинга ,  
8571 Name-pl = Листингов ,  
8572 name-pl = листингов ,  
8573 case = d ,  
8574 Name-sg = Листингу ,  
8575 name-sg = листингу ,  
8576 Name-pl = Листингам ,  
8577 name-pl = листингам ,  
8578 case = i ,  
8579 Name-sg = Листингом ,  
8580 name-sg = листингм ,  
8581 Name-pl = Листингами ,  
8582 name-pl = листингами ,  
8583 case = p ,  
8584 Name-sg = Листинге ,  
8585 name-sg = листинге ,  
8586 Name-pl = Листингах ,  
8587 name-pl = листингах ,  
8588  
8589 type = exercise ,  
8590 gender = n ,  
8591 case = n ,  
8592 Name-sg = Упражнение ,  
8593 name-sg = упражнение ,  
8594 Name-pl = Упражнения ,  
8595 name-pl = упражнения ,  
8596 Name-sg-ab = Упр. ,  
8597 name-sg-ab = упр. ,  
8598 Name-pl-ab = Упр. ,  
8599 name-pl-ab = упр. ,  
8600 case = a ,  
8601 Name-sg = Упражнение ,  
8602 name-sg = упражнение ,  
8603 Name-pl = Упражнения ,  
8604 name-pl = упражнения ,  
8605 Name-sg-ab = Упр. ,  
8606 name-sg-ab = упр. ,  
8607 Name-pl-ab = Упр. ,  
8608 name-pl-ab = упр. ,  
8609 case = g ,  
8610 Name-sg = Упражнения ,  
8611 name-sg = упражнения ,  
8612 Name-pl = Упражнений ,  
8613 name-pl = упражнений ,  
8614 Name-sg-ab = Упр. ,  
8615 name-sg-ab = упр. ,  
8616 Name-pl-ab = Упр. ,  
8617 name-pl-ab = упр. ,  
8618 case = d ,  
8619 Name-sg = Упражнению ,  
8620 name-sg = упражнению ,  
8621 Name-pl = Упражнениям ,  
8622 name-pl = упражнениям ,  
8623 Name-sg-ab = Упр. ,

8624 name-sg-ab = упр. ,  
8625 Name-pl-ab = Упр. ,  
8626 name-pl-ab = упр. ,  
8627 case = i ,  
8628 Name-sg = Упражнением ,  
8629 name-sg = упражнением ,  
8630 Name-pl = Упражнениями ,  
8631 name-pl = упражнениями ,  
8632 Name-sg-ab = Упр. ,  
8633 name-sg-ab = упр. ,  
8634 Name-pl-ab = Упр. ,  
8635 name-pl-ab = упр. ,  
8636 case = p ,  
8637 Name-sg = Упражнении ,  
8638 name-sg = упражнении ,  
8639 Name-pl = Упражнениях ,  
8640 name-pl = упражнениях ,  
8641 Name-sg-ab = Упр. ,  
8642 name-sg-ab = упр. ,  
8643 Name-pl-ab = Упр. ,  
8644 name-pl-ab = упр. ,  
8645  
8646 type = solution ,  
8647 gender = n ,  
8648 case = n ,  
8649 Name-sg = Решение ,  
8650 name-sg = решение ,  
8651 Name-pl = Решения ,  
8652 name-pl = решения ,  
8653 case = a ,  
8654 Name-sg = Решение ,  
8655 name-sg = решение ,  
8656 Name-pl = Решения ,  
8657 name-pl = решения ,  
8658 case = g ,  
8659 Name-sg = Решения ,  
8660 name-sg = решения ,  
8661 Name-pl = Решений ,  
8662 name-pl = решений ,  
8663 case = d ,  
8664 Name-sg = Решению ,  
8665 name-sg = решению ,  
8666 Name-pl = Решениям ,  
8667 name-pl = решениям ,  
8668 case = i ,  
8669 Name-sg = Решением ,  
8670 name-sg = решением ,  
8671 Name-pl = Решениями ,  
8672 name-pl = решениями ,  
8673 case = p ,  
8674 Name-sg = Решении ,  
8675 name-sg = решении ,  
8676 Name-pl = Решениях ,  
8677 name-pl = решениях ,

## 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**
- \AddToHook ..... 120, 2051, 2095,  
2118, 2148, 2150, 2187, 2260, 2302,  
2456, 2469, 2477, 5352, 5373, 5389,  
5413, 5428, 5453, 5471, 5505, 5541, 5566
  - \AddToHookWithArguments . 2431, 5667, 5671
  - \alph ..... 5486, 5557
  - \appendix ..... 2, 127, 128, 131, 138
  - \appendixname ..... 127, 140
  - \AtEndOfPackage ..... 2467
- B**
- \babelname ..... 2105
  - \babelprovide ..... 30, 55
  - bool commands:
    - \bool\_gset\_false:N ..... 559
    - \bool\_gset\_true:N ..... 552, 2443
    - \bool\_if:NTF ..... 403, 480, 518,  
576, 1769, 1820, 2055, 2059, 2433,  
2479, 3450, 3845, 3983, 4109, 4145,  
4179, 4245, 4258, 4270, 4320, 4336,  
4346, 4351, 4395, 4398, 4454, 4477,  
4484, 4490, 4501, 4507, 4535, 4580,  
4602, 4631, 4780, 4947, 4949, 5508, 5569
    - \bool\_if:nTF ..... 86,  
3561, 3571, 3594, 3611, 3626, 3691,  
3699, 4329, 4701, 4742, 4823, 4840, 4891
    - \bool\_if\_exist:NTF .... 357, 367,  
391, 401, 451, 468, 478, 503, 506,  
514, 516, 530, 533, 540, 543, 550, 557
    - \bool\_lazy\_all:nTF ..... 4422, 4959
    - \bool\_lazy\_and:nnTF .. 49, 54, 2603,  
3421, 3442, 4213, 4283, 4654, 4930, 4974
    - \bool\_lazy\_any:nTF ..... 5110, 5119
    - \bool\_lazy\_or:nnTF 1391, 1424, 1976,  
2549, 2812, 3309, 3334, 3425, 4917, 5527
    - \bool\_new:N . 138, 358, 368, 393, 452,  
470, 508, 531, 534, 541, 544, 551,  
558, 818, 1588, 1589, 1615, 1639,  
1999, 2006, 2013, 2026, 2027, 2195,  
2196, 2197, 2198, 2199, 2295, 2296,  
2424, 2436, 3458, 3473, 3731, 3732,  
3743, 3744, 3751, 3753, 3754, 3767,  
3768, 3769, 3781, 3782, 5470, 5515, 5540
    - \bool\_set:Nn ..... 3418
    - \bool\_set\_eq:NN ..... 566
    - \bool\_set\_false:N .. 392, 469, 505,  
507, 542, 1601, 1605, 1772, 1870,  
1918, 1963, 2034, 2043, 2044, 2061,  
2068, 2207, 2211, 2218, 2226, 2227,  
2228, 2320, 2332, 3463, 3554, 3798,  
3799, 3814, 3851, 3862, 4257, 4444,  
4445, 4452, 4453, 4684, 4898, 5117, 5134
    - \bool\_set\_true:N .....  
..... 359, 369, 453, 532, 535,  
545, 1595, 1596, 1600, 1606, 1794,  
1816, 1882, 1884, 1922, 1924, 1936,  
1938, 1967, 1969, 1988, 1990, 2033,  
2038, 2039, 2205, 2212, 2217, 2234,  
2236, 2238, 2241, 2242, 2243, 2308,  
2313, 3567, 3577, 3581, 3602, 3617,  
3632, 3656, 3822, 3846, 3852, 3856,  
3863, 4006, 4016, 4027, 4077, 4113,  
4149, 4183, 4200, 4280, 4314, 4479,  
4539, 4585, 4607, 4636, 4899, 4905,  
4912, 5036, 5095, 5133, 5171, 5178,  
5179, 5197, 5214, 5216, 5484, 5520, 5554
    - \bool\_until\_do:Nn .....  
..... 1871, 1925, 1970, 3592, 3815
    - \bool\_while\_do:nn ..... 5599
- C**
- \chaptername ..... 140
  - clist commands:
    - \clist\_map\_inline:nn ... 651, 699,  
716, 1033, 2229, 2445, 2949, 5487, 5559
  - \counterwithin ..... 5, 62
  - \counterwithout ..... 62
  - \crefstripprefix ..... 46
  - cs commands:
    - \cs\_generate\_variant:Nn ..... 82,  
83, 300, 306, 313, 324, 335, 346,  
361, 371, 378, 385, 396, 420, 430,  
455, 462, 473, 511, 537, 547, 554,  
561, 989, 1853, 1892, 1946, 1998,  
2611, 4736, 4774, 5154, 5258, 5291, 5324
    - \cs\_if\_exist:NTF .....  
... 24, 27, 66, 75, 96, 5328, 5340, 5392
    - \cs\_if\_exist\_p:N .....  
..... 51, 56, 4215, 4285, 4656, 5601

<code>\cs_if_exist_use:N</code> . . . . .	5452, 5639		
<code>\cs_new:Npn</code> . . . . .	64, 73, 84, 94, 105, 301, 307, 309, 311, 314, 325, 336, 348, 350, 738, 4697, 4737, 4775, 5144		
<code>\cs_new_protected:Npn</code> . . . . .	295, 352, 362, 372, 379, 386, 411, 421, 442, 444, 446, 456, 463, 501, 528, 538, 548, 555, 831, 933, 1537, 1556, 1755, 1854, 1902, 1947, 2475, 2609, 3413, 3477, 3519, 3530, 3542, 3669, 3721, 3783, 3990, 4448, 4693, 4695, 4889, 5138, 5155, 5226, 5259, 5292, 5521		
<code>\cs_set_eq:NN</code> . . . . .	740		
<code>\cs_to_str:N</code> . . . . .	351		
<b>D</b>			
<code>\DeclareHookRule</code> . . . . .	5669		
<code>\def</code> . . . . .	3		
<b>E</b>			
<code>\edef</code> . . . . .	5673		
<code>\eqref</code> . . . . .	133		
exp commands:			
<code>\exp_args:Ne</code> . . . . .	943		
<code>\exp_args:Nee</code> . . . . .	1773, 1785, 1797, 1807, 1873, 1927, 1972, 5157, 5163, 5185, 5189, 5204		
<code>\exp_args:NNe</code> . . . . .	36, 39		
<code>\exp_args:NNNo</code> . . . . .	297		
<code>\exp_args:NNNV</code> . . . . .	1834, 1886, 1940, 1992		
<code>\exp_args:NNo</code> . . . . .	297, 303		
<code>\exp_args:No</code> . . . . .	303		
<code>\exp_not:N</code> . . . . .	114, 4353, 4366, 4369, 4372, 4713, 4717, 4725, 4729, 4754, 4757, 4765, 4768, 4796, 4799, 4803, 4808, 4814, 4817, 4829, 4832, 4860, 4865, 4875, 4880		
<code>\exp_not:n</code> . . . . .	304, 4021, 4044, 4052, 4070, 4083, 4087, 4122, 4131, 4153, 4161, 4168, 4192, 4206, 4223, 4233, 4274, 4297, 4307, 4354, 4365, 4370, 4371, 4523, 4546, 4558, 4592, 4614, 4623, 4643, 4664, 4674, 4714, 4726, 4755, 4756, 4766, 4767, 4797, 4798, 4800, 4804, 4815, 4816, 4818, 4826, 4830, 4831, 4834, 4861, 4876		
<code>\ExplSyntaxOn</code> . . . . .	31, 949		
<b>F</b>			
<code>\figurename</code> . . . . .	140		
file commands:			
<code>\file_get:nnNTF</code> . . . . .	943		
<code>\fmtversion</code> . . . . .	5		
<code>\footnote</code> . . . . .	2		
		<b>G</b>	
		group commands:	
		<code>\group_begin:</code> . . . . .	754, 935, 1771, 1858, 1906, 1951, 2875, 3415, 3429, 3461, 4353, 4369, 4713, 4725, 4754, 4765, 4796, 4803, 4814, 4829, 4860, 4875
		<code>\group_end:</code> . . . . .	766, 987, 1835, 1887, 1941, 1993, 2903, 3432, 3455, 3465, 4366, 4372, 4717, 4729, 4757, 4768, 4799, 4808, 4817, 4832, 4865, 4880
		<b>H</b>	
		<code>\hyperlink</code> . . . . .	5141
		<code>\hypertarget</code> . . . . .	137
		<b>I</b>	
		<code>\IfBooleanT</code> . . . . .	3462
		<code>\IfClassLoadedTF</code> . . . . .	133
		<code>\ifdraft</code> . . . . .	2210
		<code>\IfFormatAtLeastTF</code> . . . . .	5, 6
		<code>\ifoptionfinal</code> . . . . .	2216
		<code>\IfPackageAtLeastTF</code> . . . . .	2306
		<code>\IfPackageLoadedTF</code> . . . . .	131
		<code>\input</code> . . . . .	30
		int commands:	
		<code>\int_case:nnTF</code> . . . . .	3993, 4034, 4096, 4401, 4514, 4571
		<code>\int_compare:nnNTF</code> . . . . .	3603, 3618, 3633, 3645, 3657, 3677, 3679, 3723, 3891, 4012, 4040, 4062, 4117, 4187, 4388, 4390, 4464, 4493, 4552, 5167, 5173, 5193, 5199, 5617
		<code>\int_compare_p:nnN</code> . . . . .	1394, 1427, 1977, 1979, 2552, 2815, 3312, 3337, 3693, 3701, 4426, 4921, 4933, 4964, 5130
		<code>\int_gincr:N</code> . . . . .	124
		<code>\int_incr:N</code> . . . . .	4441, 4472, 4483, 4485, 4500, 4502, 4506, 4508, 4520, 4543, 4555, 4589, 4611, 4620, 4640, 4691, 5615
		<code>\int_new:N</code> . . . . .	118, 139, 3474, 3475, 3733, 3734, 3735, 3748, 3749
		<code>\int_rand:n</code> . . . . .	322, 333, 344
		<code>\int_set:Nn</code> . . . . .	3678, 3680, 3684, 3687, 5598
		<code>\int_to_roman:n</code> . . . . .	5602, 5609, 5610, 5613
		<code>\int_use:N</code> . . . . .	52, 57, 62, 77, 128
		<code>\int_zero:N</code> . . . . .	3671, 3672, 3793, 3794, 3795, 3796, 3797, 4439, 4440, 4442, 4443, 4686, 4687
		iow commands:	
		<code>\iow_char:N</code> . . . . .	143, 158, 159, 164, 165, 170, 171, 176, 177, 229, 246, 293, 2277, 2286
		<code>\iow_newline:</code> . . . . .	287

	<b>K</b>	
keys commands:		2276, 2285, 2354, 2510, 2516, 2522, 2528, 2561, 2770, 2776, 2782, 2788, 2827, 2918, 2925, 2954, 3234, 3240, 3246, 3252, 3325, 3350, 3884, 5037, 5097
\l_keys_choice_tl	823	
\keys_define:nn		\msg_warning:nnnnn 884, 923, 2938, 4988 \msg_warning:nnnnnn . . . . . 4995
	21, 659, 705, 722, 783, 990, 1077, 1102, 1130, 1336, 1375, 1453, 1563, 1590, 1616, 1625, 1640, 1649, 2000, 2007, 2014, 2020, 2028, 2063, 2072, 2086, 2114, 2152, 2182, 2189, 2201, 2262, 2269, 2271, 2280, 2290, 2297, 2309, 2321, 2333, 2341, 2348, 2377, 2394, 2418, 2425, 2438, 2458, 2487, 2505, 2535, 2572, 2595, 2621, 2633, 2657, 2765, 2795, 2838, 2906, 2975, 2999, 3026, 3229, 3259, 3299, 3361	
\keys_set:nn	28, 31, 57, 59, 85, 763, 911, 974, 2314, 2610, 2615, 2900, 3416	
keyval commands:		
\keyval_parse:nnn	1542, 2352, 2398	
	<b>L</b>	
\label	2, 61, 64, 134, 137	
\labelformat	3	
\language	25, 140, 2099	
	<b>M</b>	
\mainbabelname	25, 2106	
msg commands:		
\msg_info:nnn	977, 1028, 1093, 1283, 1289, 1343, 5430, 5463, 5512, 5533, 5573, 5591, 5618	
\msg_info:nnnn	1002, 1009, 1038, 1407, 1441	
\msg_info:nnnnn	1022	
\msg_line_context:	142, 148, 152, 154, 157, 163, 169, 175, 181, 186, 191, 196, 201, 207, 212, 215, 218, 223, 227, 234, 239, 244, 251, 260, 265, 270, 274, 276, 278, 280, 287	
\msg_new:nnn	140, 146, 151, 153, 155, 161, 167, 173, 179, 184, 189, 194, 199, 204, 209, 214, 216, 221, 226, 228, 230, 232, 237, 242, 248, 250, 252, 257, 263, 268, 273, 275, 277, 279, 281, 283, 285, 290	
\msg_warning:nn	2060, 2066, 2336, 2606, 4428	
\msg_warning:nnn	758, 779, 1569, 1576, 1727, 1733, 2137, 2173, 2185, 2247, 2258, 2300, 2325, 2400, 2462, 2472, 2625, 2735, 2741, 2902, 2944, 2990, 3179, 3185, 3266, 3876, 4252, 4953, 4969	
\msg_warning:nnnn	847, 864, 898, 916, 2091,	
	<b>N</b>	
\NeedsTeXFormat	4	
\newcounter	5, 5327, 5391	
\NewDocumentCommand	752, 769, 2607, 2612, 2873, 3411, 3459	
\newfloat	131	
\NewHook	1648	
\newsfloat	131	
\newtheorem	139	
\nobreakspace	1550, 5688, 5689, 5691, 5692, 5694, 5696, 5893, 5894, 5896, 5897, 5899, 5901, 6340, 6341, 6343, 6344, 6346, 6348, 6557, 6558, 6560, 6561, 6563, 6565, 6783, 6784, 6786, 6787, 6789, 6791, 6997, 6998, 7000, 7001, 7003, 7005, 7215, 7216, 7218, 7219, 7221, 7223, 7224, 7349, 7450, 7451, 7453, 7454, 7456, 7458, 7459, 8126	
\noeqref	5530	
\NumCheckSetup	46	
\NumsCheckSetup	46	
	<b>P</b>	
\PackageError	9	
\pagename	140	
\pagenote	131	
\pagenumbering	7	
\pageref	86	
\PagesCheckSetup	46	
\paragraph	62	
\part	139	
\partname	140	
prg commands:		
\prg_generate_conditional_		
variant:Nnn	440, 488, 499, 526, 571, 579, 748, 1900	
\prg_new_conditional:Npnn	130, 132, 397, 474, 512, 573, 742	
\prg_new_protected_conditional:Npnn	431, 490, 562, 1893	
\prg_return_false:	131, 133, 405, 409, 438, 482, 486, 497, 520, 524, 569, 576, 577, 746, 1898	
\prg_return_true:	131, 133, 404, 407, 436, 481, 484, 495, 519, 522, 567, 576, 745, 1897	
\prg_set_eq_conditional:NNn	750	



prop commands:	
\prop_if_in:NnTF	36
\prop_if_in_p:Nn	87
\prop_item:Nn	39, 88
\prop_new:N	2347, 2393
\prop_put:Nnn	1560
\prop_remove:Nn	1559
\providecommand	5
\ProvidesExplPackage	14
\ProvidesFile	30
<b>R</b>	
\ref	137
\refstepcounter	3, 4, 132, 134, 135
regex commands:	
\regex_match:nnTF	1896
\renewlist	135
\RequirePackage	16, 17, 18, 19, 2056
\restoreapp	129
\roman	7
<b>S</b>	
\scantokens	129
seq commands:	
\seq_clear:N	467, 842, 879, 960, 973, 1032, 1636, 2546, 2809, 2886, 2899, 2948, 3479, 5285
\seq_const_from_clist:Nn	22
\seq_count:N	1395, 1409, 1428, 1443, 2553, 2563, 2816, 2829, 3313, 3327, 3338, 3352
\seq_gclear:N	1388, 1421, 3306, 3331
\seq_gconcat:NNN	644, 648
\seq_get_left:NN	857, 868, 964, 1011, 2890, 2927, 3825
\seq_gput_right:Nn	975, 981, 2449
\seq_gremove_all:Nn	2481
\seq_gset_eq:NN	460, 1060
\seq_gset_from_clist:Nn	587, 596, 608, 623, 631, 792, 807
\seq_gset_split:Nnn	445
\seq_if_empty:NTF	843, 880, 961, 1000, 1020, 2887, 2916, 2936, 3819, 4986
\seq_if_exist:NTF	448, 458, 465, 476
\seq_if_in:NnTF	861, 895, 939, 1006, 1035, 2447, 2480, 2922, 2951, 3523, 4982
\seq_item:Nn	4707, 4712, 4718, 4720, 4723, 4724, 4730, 4731, 4748, 4753, 4758, 4760, 4763, 4764, 4769, 4770, 4801, 4802, 4809, 4811, 4846, 4858, 4866, 4869, 4873, 4874, 4881, 4882
\seq_map_break:n	108, 3712, 3715
\seq_map_function:NN	3482
\seq_map_indexed_inline:Nn	45, 3673
\seq_map_inline:Nn	1074, 1099, 1333, 1372, 1450, 2471, 2484, 2532, 2569, 2618, 2630, 2792, 2835, 2972, 2996, 3256, 3296, 3358, 3709, 5523
\seq_map_tokens:Nn	90
\seq_new:N	136, 137, 449, 459, 584, 585, 586, 595, 607, 622, 630, 643, 647, 787, 802, 932, 1052, 1624, 2376, 2437, 3457, 3476, 3730, 3747, 3770, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778, 3779, 3780
\seq_pop_left:NN	3817
\seq_put_right:Nn	1036, 2952, 3526
\seq_reverse:N	1630
\seq_set_eq:NN	450, 494, 3785, 4004, 4014, 4025, 4075, 4111, 4147, 4181, 4197, 4278, 4312, 4322, 4537, 4582, 4604, 4633
\seq_set_from_clist:Nn	1629, 2380, 3417
\seq_set_split:Nnn	443
\seq_sort:Nn	88, 3485
\seq_use:Nn	5000
\setcounter	5354, 5375, 5390, 5415, 5429
\sidefootnote	131
sort commands:	
\sort_return_same:	88, 92, 3492, 3497, 3568, 3588, 3608, 3623, 3637, 3662, 3697, 3712, 3728
\sort_return_swapped:	88, 92, 3505, 3578, 3587, 3607, 3622, 3638, 3661, 3705, 3715, 3727
\space	11
\stepcounter	134
str commands:	
\str_case:n	45
\str_case:nnTF	1135, 1653, 2120, 2156, 2231, 2661, 3031
\str_compare:nNnTF	3584
\str_if_eq:nnTF	107, 4739
\str_if_eq_p:nn	5115, 5121, 5123, 5127, 5528, 5529
\str_new:N	2071
\str_set:Nn	2076, 2078, 2080, 2082
\subparagraph	138
\subref	137
\subsections	138
\subsubsection	62
\subsubsections	138
\subsubsubsection	62
<b>T</b>	
\tablename	140
>tag	124, 132, 134

TeX and L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> commands:

<code>\@Alpha</code>	127
<code>\@addtoreset</code>	5
<code>\@bsphack</code>	936
<code>\@capttype</code>	5452, 5639
<code>\@chapapp</code>	127
<code>\@currenthref</code>	5668
<code>\@currentcounter</code>	2–5, 64, 132, 135, 137, 27, 28, 55, 56, 57, 2421, 2422, 5673
<code>\@currentlabel</code>	3, 124, 135
<code>\@elt</code>	5
<code>\@esphack</code>	986
<code>\@ifl@t@r</code>	5
<code>\@onlypreamble</code>	768, 782, 2905
<code>\bbl@loaded</code>	55
<code>\bbl@main@language</code>	25, 2100
<code>\c@lstnumber</code>	135
<code>\c@page</code>	7
<code>\caption@subtyphook</code>	5640
<code>\hyper@@link</code>	113, 123
<code>\hyper@linkfile</code>	5142
<code>\lst@AddToHook</code>	5589
<code>\ltx@gobble</code>	64
<code>\MT@newlabel</code>	133
<code>\zref@addprop</code>	63, 21, 31, 46, 61, 63, 115, 129
<code>\zref@default</code>	113, 114, 4694, 4696
<code>\zref@extractdefault</code>	11, 12, 123, 298, 304, 308
<code>\zref@ifpropundefined</code>	43, 1287, 1574, 1731, 2739, 3183, 5146, 5556
<code>\zref@ifrefcontainsprop</code>	43, 45, 1759, 1767, 1826, 1844, 1856, 1904, 1949, 3879, 4699, 4782, 4836, 5149
<code>\zref@ifrefundefined</code>	3487, 3489, 3501, 3848, 3850, 3855, 3871, 4249, 4260, 4456, 4777, 4902
<code>\zref@label</code>	64, 2434
<code>\zref@localaddprop</code>	5454, 5509, 5570, 5641
<code>\ZREF@mainlist</code>	21, 31, 46, 61, 63, 115, 129, 5454, 5509, 5570, 5641
<code>\zref@newprop</code>	6, 8, 20, 22, 32, 47, 62, 110, 116, 128, 5451, 5486, 5557, 5638
<code>\zref@refused</code>	3869
<code>\zref@wrapper@babel</code>	64, 84, 2434, 3412
<code>\zrefclever@required@kernel</code>	3, 4, 6, 11
<code>\textendash</code>	1554, 5743, 6007, 6619, 6841, 7055, 7278, 7763
<code>\thechapter</code>	127, 129
<code>\thelstnumber</code>	135
<code>\thepage</code>	7, 8, 122, 125
<code>\thesection</code>	127

tl commands:

<code>\c_novalue_tl</code>	707, 708, 709, 710, 711, 712, 713, 2489, 2537, 2635, 2797
<code>\tl_clear:N</code>	366, 390, 851, 889, 902, 919, 928, 953, 962, 995, 2616, 2878, 2888, 2911, 3787, 3788, 3789, 3790, 3791, 3792, 3821, 4434, 4435, 4436, 4437, 4438, 4482, 4499, 4897, 4904, 4911, 4942, 5035, 5094, 5252
<code>\tl_const:Nn</code>	1539
<code>\tl_gc_clear:N</code>	383, 427
<code>\tl_gset:Nn</code>	376, 417, 761, 776
<code>\tl_gset_eq:NN</code>	125
<code>\tl_head:N</code>	3621, 3634, 3646, 3648, 3658, 3660
<code>\tl_head:n</code>	1874, 1875, 1928, 1929, 1973, 1974, 1980
<code>\tl_if_empty:NTF</code>	34, 98, 845, 855, 882, 893, 914, 921, 1026, 1082, 1107, 1139, 1173, 1209, 1245, 1293, 1341, 1347, 1380, 1458, 1496, 1757, 1881, 1935, 2942, 2980, 3004, 3035, 3069, 3105, 3141, 3189, 3264, 3270, 3304, 3366, 3387, 3433, 4247, 4833, 4927, 4951, 5008, 5019, 5062
<code>\tl_if_empty:nTF</code>	755, 771, 994, 1281, 1558, 1567, 1725, 2442, 2733, 2910, 3177, 5140
<code>\tl_if_empty_p:N</code>	50, 55, 2605, 4214, 4284, 4655, 4962, 4976, 5114, 5124, 5128
<code>\tl_if_empty_p:n</code>	1392, 1425, 2550, 2813, 3310, 3335, 3563, 3564, 3573, 3574, 3598, 3599, 3614, 3629
<code>\tl_if_eq:NNTF</code>	122, 3536, 3557, 3859
<code>\tl_if_eq:NnTF</code>	1783, 3480, 3512, 3683, 3686, 3711, 3714, 3828, 3874, 4908, 5161
<code>\tl_if_eq:nnTF</code>	1773, 1785, 1797, 1807, 1873, 1927, 1972, 3675, 5157, 5163, 5185, 5189, 5204
<code>\tl_if_exist:NTF</code>	354, 364, 374, 381, 388, 399, 415, 425, 744
<code>\tl_if_exist_p:N</code>	2604
<code>\tl_if_novalue:NTF</code>	2492, 2540, 2638, 2800
<code>\tl_map_break:n</code>	108
<code>\tl_map_tokens:Nn</code>	100
<code>\tl_new:N</code>	119, 134, 135, 355, 365, 375, 382, 416, 426, 581, 582, 583, 733, 734, 735, 736, 760, 775, 1562, 2181, 2200, 2268, 2289, 2340, 2417, 3467, 3468, 3469, 3470, 3471, 3472, 3736, 3737, 3738, 3739, 3740, 3741, 3742, 3745, 3746, 3750,

3752, 3755, 3756, 3757, 3758, 3759,  
3760, 3761, 3762, 3763, 3764, 3765, 3766  
\l\_put\_left:Nn . . . . . 4332, 4339,  
4382, 5021, 5022, 5064, 5066, 5068, 5070  
\l\_put\_right:Nn . . . . . 4019, 4042,  
4050, 4068, 4081, 4120, 4129, 4151,  
4159, 4166, 4190, 4204, 4221, 4231,  
4521, 4544, 4556, 4590, 4612, 4621,  
4641, 4662, 4672, 4928, 4929, 4940, 5640  
\l\_reverse:N . . . . . 3546, 3549  
\l\_set:Nn . . . . . 297, 356, 737, 762, 952,  
996, 1007, 1571, 1578, 1580, 1764,  
1832, 1836, 1849, 1860, 1865, 1877,  
1879, 1888, 1890, 1908, 1913, 1931,  
1933, 1942, 1944, 1953, 1958, 1983,  
1985, 1994, 1996, 2099, 2100, 2105,  
2106, 2109, 2110, 2124, 2129, 2134,  
2160, 2165, 2170, 2614, 2879, 2912,  
2923, 3650, 3652, 3830, 3831, 4000,  
4002, 4272, 4295, 4305, 4349, 4468,  
4470, 4480, 4497, 4923, 4924, 4938, 5525  
\l\_set\_eq:NN . . . . . 435, 4432  
\l\_show:N . . . . . 4396  
\l\_tail:N . . . . . 3651, 3653  
\l\_tail:n . . . . .  
. . . 1878, 1880, 1932, 1934, 1984, 1986  
\l\_use:N . . . . . 319,  
330, 341, 777, 941, 946, 976, 978, 982

**U**

use commands:  
\use:N 25, 28, 823, 4217, 4291, 4658, 5318  
\UseHook . . . . . 1859, 1907, 1952

**X**

\xdef . . . . . 5668

**Z**

\zcDeclareLanguage . . . 13, 26, 752, 5678,  
5883, 6336, 6551, 6780, 6994, 7212, 7445  
\zcDeclareLanguageAlias . . . . .  
. . . . . 26, 769, 5679, 5680, 5681,  
5682, 5683, 5684, 5685, 5886, 5887,  
5888, 5889, 5890, 6337, 6552, 6553, 6554  
\zcLanguageSetup . . . . .  
. . . . . 21, 30, 31, 69, 74, 75, 140, 2873  
\zcpageref . . . . . 86, 3459  
\zceref . . . . . 21, 66, 68,  
84, 86–88, 94, 96, 129, 133, 3411, 3464  
\zcRefTypeSetup . . . . . 21, 69, 2612  
\zcsetup . . . . . 21, 55, 66, 68, 2607  
\zlabel . . . . . 2, 64, 130, 132, 134  
zrefcheck commands:  
\zrefcheck\_zceref\_beg\_label: . . . 3424

\zrefcheck\_zceref\_end\_label\_-  
maybe: . . . . . 3446  
\zrefcheck\_zceref\_run\_checks\_on\_-  
labels:n . . . . . 3447  
zrefclever commands:  
\zrefclever\_language\_if\_declared:n  
. . . . . 750  
\zrefclever\_language\_if\_declared:nTF  
. . . . . 750  
\zrefclever\_language\_varname:n .  
. . . . . 740, 740  
\l\_zrefclever\_ref\_language\_tl . . 736  
zrefclever internal commands:  
\l\_\_zrefclever\_abbrev\_bool . . . . .  
. . . . . 3755, 3936, 4931  
\l\_\_zrefclever\_amsmath\_subequations\_-  
bool . . . . . 5470, 5484, 5508  
\l\_\_zrefclever\_breqn\_dgroup\_bool  
. . . . . 5540, 5554, 5569  
\l\_\_zrefclever\_cap\_bool . . . . .  
. . . . . 3755, 3932, 4918  
\l\_\_zrefclever\_capfirst\_bool . . . . .  
. . . . . 2006, 2009, 4920  
\\_\_zrefclever\_compat\_module:nn 65,  
2475, 2475, 5325, 5369, 5433, 5466,  
5516, 5536, 5576, 5594, 5621, 5644, 5663  
\\_\_zrefclever\_counter\_reset\_by:n  
6, 62, 63, 66, 68, 70, 75, 77, 79, 84,  
84, 5334, 5346, 5398, 5408, 5478, 5548  
\\_\_zrefclever\_counter\_reset\_by\_-  
aux:nn . . . . . 91, 94  
\\_\_zrefclever\_counter\_reset\_by\_-  
auxi:nnn . . . . . 101, 105  
\l\_\_zrefclever\_counter\_resetby\_-  
prop . . . . . 6, 63, 87, 88, 2393, 2405  
\l\_\_zrefclever\_counter\_resettters\_-  
seq . . . . . 5, 6, 62, 63, 90, 2376, 2380  
\l\_\_zrefclever\_counter\_type\_prop  
. . . . . 4, 61, 36, 39, 2347, 2359  
\l\_\_zrefclever\_current\_counter\_-  
tl . . . . . 3, 5, 64, 20, 24, 25, 37,  
40, 42, 50, 51, 52, 113, 117, 2417, 2420  
\l\_\_zrefclever\_current\_language\_-  
tl . . . . . 25,  
55, 734, 2099, 2105, 2109, 2125, 2161  
\l\_\_zrefclever\_endrangefunc\_tl .  
. . . . . 3755, 3924, 4214, 4215,  
4217, 4284, 4285, 4291, 4655, 4656, 4658  
\l\_\_zrefclever\_endrangeprop\_tl .  
. . . . . 48, 1757, 1767, 3755, 3928  
\\_\_zrefclever\_extract:nnn . . . . .  
. . . . . 12, 307, 307, 1862,  
1867, 1910, 1915, 1955, 1960, 3604,

3606, 3619, 3636, 3724, 3726, 5168,  
 5170, 5174, 5176, 5194, 5196, 5200, 5202  
 \\_zrefclever\_extract\_default:Nnn  
 ..... 12, 295, 295, 300, 1761,  
 1822, 1829, 1839, 1846, 3521, 3532,  
 3534, 3544, 3547, 3550, 3552, 3834, 3837  
 \\_zrefclever\_extract\_unexp:nnn  
 ..... 12, 123, 301, 301, 306, 1775,  
 1779, 1787, 1791, 1799, 1803, 1809,  
 1813, 4361, 4710, 4715, 4727, 4751,  
 4792, 4805, 4854, 4862, 4877, 5147,  
 5150, 5151, 5158, 5159, 5164, 5165,  
 5186, 5187, 5190, 5191, 5206, 5210, 5526  
 \\_zrefclever\_extract\_url\_-  
 unexp:n ..... 4357,  
 4709, 4750, 4788, 4850, 5144, 5144, 5154  
 \\_zrefclever\_get\_enclosing\_-  
 counters:n ... 6, 64, 64, 69, 82, 117  
 \\_zrefclever\_get\_enclosing\_-  
 counters\_value:n 6, 64, 73, 78, 83, 112  
 \\_zrefclever\_get\_endrange\_-  
 pagecomp:nnN ..... 1902, 1946  
 \\_zrefclever\_get\_endrange\_-  
 pagecomptwo:nnN ..... 1947, 1998  
 \\_zrefclever\_get\_endrange\_-  
 property:nnN ..... 45, 1755, 1853  
 \\_zrefclever\_get\_endrange\_-  
 stripprefix:nnN ..... 1854, 1892  
 \\_zrefclever\_get\_ref:nN .....  
 ..... 113, 114, 4022, 4045, 4053,  
 4071, 4084, 4088, 4123, 4132, 4154,  
 4162, 4169, 4193, 4207, 4234, 4275,  
 4308, 4341, 4524, 4547, 4559, 4593,  
 4615, 4624, 4644, 4675, 4697, 4697, 4736  
 \\_zrefclever\_get\_ref\_endrange:nnN  
 ..... 46, 114,  
 115, 4224, 4298, 4665, 4737, 4737, 4774  
 \\_zrefclever\_get\_ref\_first: ...  
 . 113, 114, 118, 4333, 4383, 4775, 4775  
 \\_zrefclever\_get\_rf\_opt\_bool:nN 126  
 \\_zrefclever\_get\_rf\_opt\_-  
 bool:nnnnN ..... 21,  
 3929, 3933, 3937, 5292, 5292, 5324  
 \\_zrefclever\_get\_rf\_opt\_-  
 seq:nnnN ..... 21, 126, 3941,  
 3945, 3949, 3953, 3957, 3961, 3965,  
 3969, 3973, 3977, 4978, 5259, 5259, 5291  
 \\_zrefclever\_get\_rf\_opt\_tl:nnnN  
 . 21, 23, 46, 125, 3435, 3801, 3805,  
 3809, 3893, 3897, 3901, 3905, 3909,  
 3913, 3917, 3921, 3925, 5226, 5226, 5258  
 \\_zrefclever\_hyperlink:nnn . 123,  
 4355, 4708, 4749, 4786, 4848, 5138, 5138  
 \l\_zrefclever\_hyperlink\_bool ..  
 .... 2026, 2033, 2038, 2043, 2055,  
 2061, 2068, 3463, 4703, 4744, 4842, 5112  
 \l\_zrefclever\_hyperref\_warn\_-  
 bool .. 2027, 2034, 2039, 2044, 2059  
 \\_zrefclever\_if\_class\_loaded:n  
 ..... 130, 132  
 \\_zrefclever\_if\_class\_loaded:nTF  
 ..... 5435, 5665  
 \\_zrefclever\_if\_package\_-  
 loaded:n ..... 130, 130  
 \\_zrefclever\_if\_package\_-  
 loaded:nTF .....  
 .... 2053, 2097, 2103, 2304, 5371,  
 5468, 5518, 5538, 5578, 5596, 5623, 5646  
 \\_zrefclever\_is\_integer\_rgx:n .  
 ..... 1893, 1894, 1901  
 \\_zrefclever\_is\_integer\_rgx:nTF  
 ..... 1919, 1921, 1964, 1966  
 \l\_zrefclever\_label\_a\_tl .....  
 ..... 93, 3736, 3818, 3836, 3848,  
 3869, 3871, 3877, 3880, 3886, 4001,  
 4022, 4045, 4053, 4088, 4154, 4169,  
 4219, 4225, 4234, 4265, 4275, 4293,  
 4299, 4308, 4456, 4460, 4469, 4481,  
 4498, 4524, 4560, 4624, 4660, 4666, 4675  
 \l\_zrefclever\_label\_b\_tl ... 93,  
 3736, 3821, 3826, 3839, 3850, 3855, 4460  
 \l\_zrefclever\_label\_count\_int .  
 ..... 94, 3733,  
 3793, 3891, 3993, 4439, 4464, 4691, 4965  
 \l\_zrefclever\_label\_enclval\_a\_-  
 tl ..... 3467, 3544, 3546,  
 3598, 3614, 3634, 3646, 3650, 3651, 3658  
 \l\_zrefclever\_label\_enclval\_b\_-  
 tl ..... 3467, 3547, 3549,  
 3599, 3621, 3629, 3648, 3652, 3653, 3660  
 \l\_zrefclever\_label\_extdoc\_a\_tl  
 ... 3467, 3550, 3558, 3563, 3573, 3586  
 \l\_zrefclever\_label\_extdoc\_b\_tl  
 ... 3467, 3552, 3559, 3564, 3574, 3585  
 \l\_zrefclever\_label\_type\_a\_tl .  
 ..... 3436, 3467, 3522,  
 3524, 3527, 3533, 3537, 3683, 3711,  
 3802, 3806, 3810, 3830, 3835, 3860,  
 3874, 3894, 3898, 3902, 3906, 3910,  
 3914, 3918, 3922, 3926, 3930, 3934,  
 3938, 3942, 3946, 3950, 3954, 3958,  
 3962, 3966, 3970, 3974, 3978, 4003, 4471  
 \l\_zrefclever\_label\_type\_b\_tl .  
 ..... 3467,  
 3535, 3538, 3686, 3714, 3831, 3838, 3861  
 \\_zrefclever\_label\_type\_put\_-  
 new\_right:n 87, 88, 3483, 3519, 3519

<code>\l_zrefclever_label_types_seq</code> .	4929, 4939, 4940, 5014, 5021, 5022,
..... 88, 3476, 3479, 3523, 3526, 3709	5030, 5038, 5048, 5065, 5066, 5079, 5099
<code>\l_zrefclever_labelhook_bool</code> ..	<code>\l_zrefclever_name_in_link_bool</code>
..... 2424, 2427, 2433	..... 116, 118,
<code>\_zrefclever_labels_in_sequence:mn</code>	3742, 4351, 4780, 4898, 5117, 5133, 5134
... 48, 94, 123, 4263, 4459, 5155, 5155	<code>\l_zrefclever_namefont_tl</code> ....
<code>\l_zrefclever_lang_decl_case_tl</code>	3755, 3916, 4354, 4370, 4797, 4815, 4830
.. 581, 962, 965, 1007, 1012, 1347,	<code>\l_zrefclever_nameinlink_str</code> ..
1364, 2888, 2891, 2923, 2928, 3270, 3287	..... 2071, 2076,
<code>\l_zrefclever_lang_declension_-</code>	2078, 2080, 2082, 5115, 5121, 5123, 5127
seq .....	<code>\l_zrefclever_namesep_tl</code> .....
581,	... 3755, 3896, 4800, 4818, 4826, 4834
841, 842, 843, 857, 861, 868, 959,	<code>\l_zrefclever_next_is_same_bool</code>
960, 961, 964, 1000, 1006, 1011,	..... 94, 123,
2885, 2886, 2887, 2890, 2916, 2922, 2927	3748, 4453, 4484, 4501, 4507, 5179, 5217
<code>\l_zrefclever_lang_gender_seq</code> .	<code>\l_zrefclever_next_maybe_range_-</code>
..... 581, 878, 879, 880, 895, 972,	bool .. 94, 123, 3748, 4257, 4270,
973, 1020, 1035, 2898, 2899, 2936, 2951	4452, 4477, 4490, 5171, 5178, 5197, 5215
<code>\_zrefclever_language_if_-</code>	<code>\l_zrefclever_noabbrev_first_-</code>
declared:n .....	bool .....
26, 742, 749, 751	2013, 2016, 4935
<code>\_zrefclever_language_if_-</code>	<code>\g_zrefclever_nocompat_bool</code> ...
declared:n(TF) .....	..... 2436, 2443, 2479
25	<code>\l_zrefclever_nocompat_bool</code> ....
<code>\_zrefclever_language_if_-</code>	65
declared:nTF ... 316, 327, 338,	<code>\g_zrefclever_nocompat_modules_-</code>
742, 757, 773, 833, 937, 2135, 2171, 2876	seq 2437, 2447, 2450, 2471, 2480, 2481
<code>\_zrefclever_language_varname:n</code>	<code>\l_zrefclever_nocompat_modules_-</code>
..... 25, 319, 330,	seq .....
341, 738, 738, 741, 744, 760, 761,	65
775, 776, 777, 941, 946, 976, 978, 982	<code>\l_zrefclever_nudge_comptosing_-</code>
<code>\l_zrefclever_last_of_type_bool</code>	bool .. 2197, 2227, 2236, 2242, 4961
..... 93, 3730,	<code>\l_zrefclever_nudge_enabled_-</code>
3846, 3851, 3852, 3856, 3862, 3863, 3983	bool .....
<code>\l_zrefclever_lastsep_tl</code> .....	2195, 2205,
3755, 3908, 4052, 4087, 4131, 4168, 4206	2207, 2211, 2212, 2217, 2218, 4424, 4947
<code>\l_zrefclever_link_star_bool</code> ..	<code>\l_zrefclever_nudge_gender_bool</code>
... 3418, 3457, 4704, 4745, 4843, 5113	..... 2199, 2228, 2238, 2243, 4975
<code>\l_zrefclever_listsep_tl</code> .....	<code>\l_zrefclever_nudge_multitype_-</code>
..... 3755, 3904, 4083, 4161,	bool .. 2196, 2226, 2234, 2241, 4425
4523, 4546, 4558, 4592, 4614, 4623, 4643	<code>\l_zrefclever_nudge_singular_-</code>
<code>\g_zrefclever_loaded_langfiles_-</code>	bool .....
seq .....	2198, 2254, 4949
932, 940, 975, 981	<code>\_zrefclever_opt_bool_get:NN</code> ..
<code>\l_zrefclever_main_language_tl</code>	..... 562, 572
25, 55, 735, 2100, 2106, 2110, 2130, 2166	<code>\_zrefclever_opt_bool_get:NN(TF)</code>
<code>\l_zrefclever_mathtools_loaded_-</code>	..... 20
bool .....	<code>\_zrefclever_opt_bool_get:NNTF</code>
3450, 5515, 5520	... 562, 5295, 5300, 5305, 5310, 5315
<code>\_zrefclever_mathtools_showonlyrefs:n</code>	<code>\_zrefclever_opt_bool_gset_-</code>
..... 3452, 5521	false:N .....
<code>\_zrefclever_name_default:</code> ....	19,
..... 4693, 4695, 4825	528, 555, 561, 1505, 1522, 3389, 3397
<code>\l_zrefclever_name_format_-</code>	<code>\_zrefclever_opt_bool_gset_-</code>
fallback_tl .....	true:N .....
3742,	19,
4938, 4942, 5008, 5057, 5069, 5071, 5089	528, 548, 554, 1467, 1484, 3368, 3376
<code>\l_zrefclever_name_format_tl</code> ..	<code>\_zrefclever_opt_bool_if:N</code> . 573, 580
..... 3742, 4923, 4924, 4928,	<code>\_zrefclever_opt_bool_if:N(TF)</code> . 20
	<code>\_zrefclever_opt_bool_if:NTF</code> ..
	..... 573, 906

```

\__zrefclever_opt_bool_if_set:N
    ..... 512, 527
\__zrefclever_opt_bool_if_-
  set:N(TF) ..... 19
\__zrefclever_opt_bool_if_-
  set:N(TF) .....
    . 512, 564, 575, 1460, 1476, 1498, 1514
\__zrefclever_opt_bool_set_-
  false:N 19, 528, 538, 547, 2582, 2852
\__zrefclever_opt_bool_set_-
  true:N . 19, 528, 528, 537, 2577, 2843
\__zrefclever_opt_bool_unset:N .
    ..... 18, 501, 501, 511, 2587, 2861
\__zrefclever_opt_seq_get:NN 490, 500
\__zrefclever_opt_seq_get:NN(TF) 18
\__zrefclever_opt_seq_get:NNTF .
    ..... 490, 836, 873, 954, 967,
    2880, 2893, 5262, 5267, 5272, 5277, 5282
\__zrefclever_opt_seq_gset_-
  clist_split:Nn .....
    .. 17, 442, 444, 1389, 1422, 3307, 3332
\__zrefclever_opt_seq_gset_eq:NN
    ..... 17, 442,
    456, 462, 1398, 1431, 2959, 3316, 3341
\__zrefclever_opt_seq_if_set:N .
    ..... 474, 489
\__zrefclever_opt_seq_if_-
  set:N(TF) ..... 18
\__zrefclever_opt_seq_if_set:N(TF)
    ..... 474, 492, 1043, 1382, 1414
\__zrefclever_opt_seq_set_clist_-
  split:Nn ... 17, 442, 442, 2547, 2810
\__zrefclever_opt_seq_set_eq:NN
    ..... 17, 442, 446, 455, 2556, 2819
\__zrefclever_opt_seq_unset:N ..
    ..... 17, 463, 463, 473, 2542, 2802
\__zrefclever_opt_tl_clear:N ...
    15, 352, 362, 371, 1657, 1662, 1676,
    1690, 1704, 2665, 2670, 2684, 2698, 2712
\__zrefclever_opt_tl_cset_-
  fallback:nn ..... 1537, 1544
\__zrefclever_opt_tl_gclear:N ..
    15, 352, 379, 385, 3037, 3043, 3051,
    3058, 3078, 3094, 3114, 3130, 3150, 3166
\__zrefclever_opt_tl_gclear_if_-
  new:N ..... 16,
    411, 421, 430, 1141, 1147, 1155,
    1162, 1182, 1198, 1218, 1234, 1254, 1270
\__zrefclever_opt_tl_get:NN . 431, 441
\__zrefclever_opt_tl_get:NN(TF) . 17
\__zrefclever_opt_tl_get:NNTF ..
    ..... 431, 5010, 5025, 5044, 5053,
    5074, 5084, 5229, 5234, 5239, 5244, 5249
\__zrefclever_opt_tl_gset:N ..... 15
\__zrefclever_opt_tl_gset:Nn ...
    ..... 352, 372, 378, 2982, 3006,
    3014, 3071, 3086, 3107, 3122, 3143,
    3158, 3191, 3198, 3207, 3215, 3272, 3282
\__zrefclever_opt_tl_gset_if_-
  new:Nn ..... 16,
    411, 411, 420, 1084, 1109, 1118,
    1175, 1190, 1211, 1226, 1247, 1262,
    1295, 1302, 1311, 1319, 1349, 1359
\__zrefclever_opt_tl_if_set:N .. 397
\__zrefclever_opt_tl_if_set:N(TF)
    ..... 16
\__zrefclever_opt_tl_if_set:N(TF)
    ..... 397, 413, 423, 433
\__zrefclever_opt_tl_set:N ..... 15
\__zrefclever_opt_tl_set:Nn ....
    ..... 352, 352,
    361, 1670, 1684, 1698, 1737, 1743,
    2498, 2647, 2678, 2692, 2706, 2745, 2752
\__zrefclever_opt_tl_unset:N ...
    ..... 15, 386, 386,
    396, 1712, 1717, 2494, 2640, 2720, 2725
\__zrefclever_opt_var_set_bool:n
    ..... 14, 15, 350, 350, 357,
    358, 359, 367, 368, 369, 391, 392,
    393, 401, 403, 451, 452, 453, 468,
    469, 470, 478, 480, 506, 507, 508,
    516, 518, 533, 534, 535, 543, 544, 545
\__zrefclever_opt_varname_-
  fallback:nn .....
    .. 14, 348, 348, 1540, 5250, 5283, 5316
\__zrefclever_opt_varname_-
  general:nn ..... 13,
    309, 309, 1659, 1664, 1672, 1678,
    1686, 1692, 1700, 1706, 1714, 1719,
    1739, 1745, 2495, 2499, 2543, 2557,
    2578, 2583, 2588, 5230, 5263, 5296
\__zrefclever_opt_varname_lang_-
  default:nnn ..... 13, 325, 325,
    335, 1086, 1111, 1143, 1149, 1177,
    1184, 1213, 1220, 1249, 1256, 1297,
    1304, 1384, 1400, 1462, 1469, 1500,
    1507, 2984, 3008, 3039, 3045, 3073,
    3080, 3109, 3116, 3145, 3152, 3193,
    3200, 3318, 3370, 3391, 5245, 5278, 5311
\__zrefclever_opt_varname_lang_-
  type:nnnn ..... 14,
    336, 336, 347, 1045, 1054, 1062,
    1120, 1157, 1164, 1192, 1200, 1228,
    1236, 1264, 1272, 1313, 1321, 1351,
    1361, 1416, 1433, 1478, 1486, 1516,
    1524, 2961, 3016, 3053, 3060, 3088,
    3096, 3124, 3132, 3160, 3168, 3209,
    3217, 3274, 3284, 3343, 3378, 3399,

```

5027, 5076, 5086, 5240, 5273, 5306

\\_zrefclever\_opt\_varname\_-  
  language:nnn ..... 13, 314,  
  314, 324, 789, 794, 804, 809, 820,  
  825, 838, 875, 908, 956, 969, 2882, 2895

\\_zrefclever\_opt\_varname\_-  
  type:nnn ..... 13, 311, 311, 313,  
  2642, 2649, 2667, 2672, 2680, 2686,  
  2694, 2700, 2708, 2714, 2722, 2727,  
  2747, 2754, 2804, 2821, 2845, 2854,  
  2863, 5012, 5046, 5055, 5235, 5268, 5301

\g\_zrefclever\_page\_format\_int .  
  ..... 118, 124, 128

\l\_zrefclever\_pairsep\_tl .....  
  ..... 3755, 3900,  
  4021, 4044, 4070, 4122, 4153, 4192, 4274

\g\_zrefclever\_prev\_page\_format\_-  
  tl ..... 8, 119, 122, 125

\\_zrefclever\_process\_language\_-  
  settings: ... 57, 59, 831, 831, 3420

\\_zrefclever\_prop\_put\_non\_-  
  empty:Nnn 43, 1556, 1556, 2358, 2404

\\_zrefclever\_provide\_langfile:n  
  21, 31, 32, 85, 933, 933, 989, 2141, 3419

\l\_zrefclever\_range\_beg\_is\_-  
  first\_bool .....  
  .... 3748, 3798, 4109, 4145, 4179,  
  4444, 4479, 4535, 4580, 4602, 4631, 4684

\l\_zrefclever\_range\_beg\_label\_-  
  tl 94, 3748, 3791, 4072, 4085, 4124,  
  4133, 4163, 4194, 4208, 4218, 4437,  
  4480, 4497, 4548, 4594, 4616, 4645, 4659

\l\_zrefclever\_range\_count\_int .  
  . 94, 3748, 3796, 4034, 4098, 4442,  
  4483, 4494, 4500, 4506, 4514, 4573, 4686

\l\_zrefclever\_range\_end\_ref\_tl  
  ..... 3748, 3792, 4220, 4226,  
  4294, 4300, 4438, 4482, 4499, 4661, 4667

\l\_zrefclever\_range\_same\_count\_-  
  int ..... 94,  
  3748, 3797, 4012, 4063, 4099, 4443,  
  4485, 4502, 4508, 4553, 4574, 4687

\l\_zrefclever\_rangeseq\_tl 3755,  
  3912, 4223, 4233, 4297, 4307, 4664, 4674

\l\_zrefclever\_rangetopair\_bool  
  ..... 3755, 3940, 4258

\l\_zrefclever\_ref\_count\_int 3733,  
  3795, 4040, 4118, 4188, 4440, 4472,  
  4520, 4543, 4555, 4589, 4611, 4620, 4640

\l\_zrefclever\_ref\_decl\_case\_tl  
  ..... 28, 845, 850, 851, 855,  
  858, 862, 866, 869, 914, 917, 919,  
  2181, 2191, 5019, 5023, 5062, 5067, 5072

\\_zrefclever\_ref\_default: .....  
  4693, 4693, 4734, 4740, 4778, 4819, 4885

\l\_zrefclever\_ref\_gender\_tl ...  
  ..... 29, 882, 888,  
  889, 893, 896, 901, 902, 921, 927,  
  928, 2200, 2264, 4976, 4984, 4990, 4998

\l\_zrefclever\_ref\_language\_tl .  
  ..... 25, 28, 55, 733,  
  737, 834, 839, 849, 867, 876, 886,  
  900, 909, 918, 925, 2124, 2129, 2134,  
  2142, 2160, 2165, 2170, 3419, 3437,  
  3803, 3807, 3811, 3895, 3899, 3903,  
  3907, 3911, 3915, 3919, 3923, 3927,  
  3931, 3935, 3939, 3943, 3947, 3951,  
  3955, 3959, 3963, 3967, 3971, 3975,  
  3979, 4980, 4992, 5003, 5028, 5077, 5087

\l\_zrefclever\_ref\_property\_tl .  
  ..... 43, 48, 1562,  
  1571, 1578, 1580, 1759, 1783, 1827,  
  1844, 1856, 1863, 1868, 1904, 1911,  
  1916, 1949, 1956, 1961, 3480, 3512,  
  3828, 3881, 3885, 4699, 4784, 4838, 5161

\l\_zrefclever\_ref\_typeset\_font\_-  
  tl ..... 2268, 2270, 3430

\l\_zrefclever\_refbounds\_first\_-  
  pb\_seq ..... 3770,  
  3952, 4026, 4076, 4148, 4199, 4279

\l\_zrefclever\_refbounds\_first\_-  
  rb\_seq . 3770, 3956, 4182, 4313, 4635

\l\_zrefclever\_refbounds\_first\_-  
  seq 3770, 3944, 4323, 4538, 4584, 4606

\l\_zrefclever\_refbounds\_first\_-  
  sg\_seq . 3770, 3948, 4005, 4015, 4112

\l\_zrefclever\_refbounds\_last\_-  
  pe\_seq ..... 3770,  
  3976, 4023, 4046, 4073, 4125, 4155, 4276

\l\_zrefclever\_refbounds\_last\_-  
  re\_seq .....  
  ... 3770, 3980, 4227, 4235, 4301, 4309

\l\_zrefclever\_refbounds\_last\_-  
  seq 3770, 3972, 4054, 4089, 4134, 4170

\l\_zrefclever\_refbounds\_mid\_rb\_-  
  seq ... 3770, 3964, 4195, 4209, 4646

\l\_zrefclever\_refbounds\_mid\_re\_-  
  seq ..... 3770, 3968, 4668, 4676

\l\_zrefclever\_refbounds\_mid\_seq  
  ..... 3770, 3960, 4086,  
  4164, 4525, 4549, 4561, 4595, 4617, 4625

\l\_zrefclever\_reffont\_tl .....  
  ..... 3755, 3920,  
  4714, 4726, 4755, 4766, 4804, 4861, 4876

\l\_zrefclever\_reftype\_override\_-  
  tl ..... 34, 44, 2340, 2343

\g_zrefclever_rf_opts_bool_- maybe_type_specific_seq . . . . .	\__zrefclever_sort_default_- different_types:nn . . . . .
. . . . . 53, 586, 1451, 2570, 2836, 3359	. . . . . 45, 87, 91, 3540, 3669, 3669
\g_zrefclever_rf_opts_seq_- refbounds_seq . . . . .	\__zrefclever_sort_default_same_- type:nn . . . . .
. . . . . 586, 1373, 2533, 2793, 3297	. . . . . 86, 89, 3539, 3542, 3542
\g_zrefclever_rf_opts_tl_maybe_- type_specific_seq . 586, 1100, 2997	\__zrefclever_sort_labels: . . . . .
\g_zrefclever_rf_opts_tl_not_- type_specific_seq . . . . .	. . . . . 87, 88, 92, 3428, 3477, 3477
. . . . . 586, 1075, 2619, 2973	\__zrefclever_sort_page:nn . . . . .
\g_zrefclever_rf_opts_tl_- reference_seq . . . . .	. . . . . 92, 3513, 3721, 3721
. . . . . 586, 2485	\l__zrefclever_sort_prior_a_int
\g_zrefclever_rf_opts_tl_type_- names_seq . . . . .	3474, 3671, 3677, 3678, 3684, 3694, 3702
. . . . . 586, 1334, 3257	\l__zrefclever_sort_prior_b_int
\g_zrefclever_rf_opts_tl_- typesetup_seq . . . . .	3474, 3672, 3679, 3680, 3687, 3695, 3703
. . . . . 586, 2631	\l__zrefclever_tlastsep_tl . . . . .
\l__zrefclever_setup_language_tl	. . . . . 3755, 3812, 4418
. . . . . 581, 762, 790, 795,	\l__zrefclever_tlistsep_tl . . . . .
805, 810, 821, 826, 952, 1003, 1010,	. . . . . 3755, 3808, 4391
1023, 1040, 1046, 1055, 1063, 1087,	\l__zrefclever_tmpa_bool 134, 1772,
1112, 1121, 1144, 1150, 1158, 1165,	1794, 1816, 1820, 1870, 1871, 1882,
1178, 1185, 1193, 1201, 1214, 1221,	1884, 1918, 1922, 1924, 1925, 1936,
1229, 1237, 1250, 1257, 1265, 1273,	1938, 1963, 1967, 1969, 1970, 1988, 1990
1298, 1305, 1314, 1322, 1352, 1362,	\l__zrefclever_tmpa_int . . . . .
1385, 1401, 1417, 1434, 1463, 1470,	134,
1479, 1487, 1501, 1508, 1517, 1525,	5598, 5602, 5609, 5610, 5613, 5615, 5617
2879, 2919, 2926, 2939, 2956, 2962,	\g_zrefclever_tmpa_seq 134, 1388,
2985, 3009, 3017, 3040, 3046, 3054,	1390, 1395, 1404, 1409, 1421, 1423,
3061, 3074, 3081, 3089, 3097, 3110,	1428, 1438, 1443, 3306, 3308, 3313,
3117, 3125, 3133, 3146, 3153, 3161,	3322, 3327, 3331, 3333, 3338, 3347, 3352
3169, 3194, 3201, 3210, 3218, 3275,	\l__zrefclever_tmpa_seq . . . . .
3285, 3319, 3344, 3371, 3379, 3392, 3400	. . . . . 134, 1032, 1036, 1068,
\l__zrefclever_setup_type_tl . . . . .	2546, 2548, 2553, 2558, 2563, 2809,
. . . . . 581, 953, 995, 996, 1026, 1047,	2811, 2816, 2824, 2829, 2948, 2952, 2967
1056, 1064, 1082, 1107, 1122, 1139,	\l__zrefclever_tmpa_tl . . . . .
1159, 1166, 1173, 1194, 1202, 1209,	. . . . . 134, 950, 974,
1230, 1238, 1245, 1266, 1274, 1293,	1860, 1874, 1877, 1878, 1908, 1919,
1315, 1323, 1341, 1353, 1363, 1380,	1928, 1931, 1932, 1953, 1964, 1973,
1418, 1435, 1458, 1480, 1488, 1496,	1983, 1984, 3438, 3439, 5525, 5528, 5529
1518, 1526, 2614, 2616, 2643, 2650,	\l__zrefclever_tmpb_tl . . . . .
2668, 2673, 2681, 2687, 2695, 2701,	134,
2709, 2715, 2723, 2728, 2748, 2755,	1822, 1829, 1832, 1836, 1865, 1875,
2805, 2822, 2846, 2855, 2864, 2878,	1879, 1880, 1881, 1888, 1913, 1921,
2911, 2912, 2942, 2963, 2980, 3004,	1929, 1933, 1934, 1935, 1942, 1958,
3018, 3035, 3055, 3062, 3069, 3090,	1966, 1974, 1977, 1980, 1985, 1986, 1994
3098, 3105, 3126, 3134, 3141, 3162,	\l__zrefclever_tpairsep_tl . . . . .
3170, 3189, 3211, 3219, 3264, 3276,	. . . . . 3755, 3804, 4411
3286, 3304, 3345, 3366, 3380, 3387, 3401	\l__zrefclever_type_count_int . . . . .
\l__zrefclever_sort_decided_bool	. . . . . 94, 118, 3733, 3794, 4388,
. . . . . 3473, 3554, 3567,	4390, 4401, 4426, 4441, 4921, 4934, 5130
3577, 3581, 3592, 3602, 3617, 3632, 3656	\l__zrefclever_type_first_label_- tl 93, 116, 3736, 3789, 4000, 4249,
\__zrefclever_sort_default:nn . . . . .	4260, 4264, 4292, 4341, 4358, 4362,
. . . . . 88, 3514, 3530, 3530	4435, 4468, 4777, 4783, 4789, 4793,
	4806, 4837, 4851, 4855, 4863, 4878, 4902
	\l__zrefclever_type_first_label_- type_tl . . . . . 93, 118, 3736,
	3790, 4002, 4253, 4436, 4470, 4909,



4955, 4971, 4979, 4991, 4997, 5013,  
 5029, 5039, 5047, 5056, 5078, 5088, 5100  
 \l\_zrefclever\_type\_first\_  
   refbounds\_seq .....  
     ..... 3770, 4004, 4014, 4025,  
     4075, 4111, 4147, 4181, 4198, 4278,  
     4312, 4322, 4342, 4537, 4583, 4605,  
     4634, 4801, 4802, 4809, 4811, 4847,  
     4859, 4867, 4870, 4873, 4874, 4881, 4882  
 \l\_zrefclever\_type\_first\_  
   refbounds\_set\_bool .....  
     .... 3770, 3799, 4006, 4016, 4027,  
     4078, 4114, 4150, 4184, 4201, 4280,  
     4314, 4320, 4445, 4540, 4586, 4608, 4637  
 \l\_zrefclever\_type\_name\_gender\_  
   seq ... 3742, 4981, 4983, 4986, 5001  
 \l\_zrefclever\_type\_name\_  
   missing\_bool ..... 118,  
     3742, 4823, 4899, 4905, 4912, 5036, 5096  
 \\_zrefclever\_type\_name\_setup: .  
   ..... 21, 23, 116, 4328, 4889, 4889  
 \l\_zrefclever\_type\_name\_tl ....  
   ..... 116, 118,  
     3742, 4365, 4371, 4798, 4816, 4831,  
     4833, 4897, 4904, 4911, 5017, 5033,  
     5035, 5051, 5060, 5082, 5092, 5094, 5114  
 \l\_zrefclever\_typeset\_compress\_  
   bool ..... 1639, 1642, 4454  
 \l\_zrefclever\_typeset\_labels\_  
   seq . 93, 3730, 3785, 3817, 3819, 3825  
 \l\_zrefclever\_typeset\_last\_bool  
   ..... 93,  
     3730, 3814, 3815, 3822, 3845, 4398, 5129  
 \l\_zrefclever\_typeset\_name\_bool  
   1589, 1596, 1601, 1606, 4330, 4346, 4892  
 \l\_zrefclever\_typeset\_queue\_  
   curr\_tl .. 93, 96, 113, 118, 3736,  
     3788, 4019, 4042, 4050, 4068, 4081,  
     4120, 4129, 4151, 4159, 4166, 4190,  
     4204, 4221, 4231, 4247, 4272, 4295,  
     4305, 4332, 4339, 4349, 4382, 4396,  
     4406, 4412, 4419, 4433, 4434, 4521,  
     4544, 4556, 4590, 4612, 4621, 4641,  
     4662, 4672, 4927, 4951, 4962, 5124, 5128  
 \l\_zrefclever\_typeset\_queue\_  
   prev\_tl .. 93, 3736, 3787, 4392, 4432  
 \l\_zrefclever\_typeset\_range\_  
   bool .. 1769, 1999, 2002, 3427, 4245  
 \l\_zrefclever\_typeset\_ref\_bool  
   1588, 1595, 1600, 1605, 4330, 4336, 4892  
 \\_zrefclever\_typeset\_refs: ....  
   ..... 93-95, 3431, 3783, 3783  
 \\_zrefclever\_typeset\_refs\_last\_  
   of\_type: .....  
   .. 100, 113, 115, 118, 3985, 3990, 3990  
 \\_zrefclever\_typeset\_refs\_not\_  
   last\_of\_type: .....  
   ... 94, 100, 113, 123, 3987, 4448, 4448  
 \l\_zrefclever\_typeset\_sort\_bool  
   ..... 1615, 1618, 3426  
 \l\_zrefclever\_typesort\_seq ....  
   .. 45, 91, 1624, 1629, 1630, 1636, 3673  
 \l\_zrefclever\_verbose\_testing\_  
   bool ..... 3782, 4395  
 \\_zrefclever\_zcref:nnn .....  
   ..... 28, 57, 3412, 3413  
 \\_zrefclever\_zcref:nnnn 84, 87, 3413  
 \l\_zrefclever\_zcref\_labels\_seq  
   ..... 87, 88,  
     3417, 3448, 3453, 3457, 3482, 3485, 3786  
 \l\_zrefclever\_zcref\_note\_tl ...  
   ..... 2289, 2292, 3433, 3440  
 \l\_zrefclever\_zcref\_with\_check\_  
   bool ..... 2296, 2313, 3423, 3444  
 \\_zrefclever\_zcsetup:n .....  
   ..... 68, 2608, 2609, 2609,  
     2611, 5330, 5342, 5355, 5376, 5394,  
     5404, 5416, 5437, 5455, 5473, 5507,  
     5543, 5568, 5580, 5590, 5605, 5625, 5648  
 \l\_zrefclever\_zrefcheck\_  
   available\_bool .....  
   ... 2295, 2308, 2320, 2332, 3422, 3443