

The zref-clever package

Code documentation

gusbrs

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

Version v0.5.1 – 2024-11-28

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	8
	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
5	Configuration	69
	5.1 \zcsetup	69
	5.2 \zcRefTypeSetup	69
	5.3 \zcLanguageSetup	74
6	User interface	84
	6.1 \zcref	84
	6.2 \zcpageref	86
7	Sorting	86
8	Typesetting	93

9	Compatibility	127
9.1	appendix	128
9.2	appendices	129
9.3	memoir	131
9.4	amsmath	132
9.5	mathtools	133
9.6	breqn	134
9.7	listings	135
9.8	enumitem	136
9.9	subcaption	137
9.10	subfig	137
9.11	beamer	138
10	Language files	138
10.1	Localization guidelines	138
10.2	English	141
10.3	German	145
10.4	French	153
10.5	Portuguese	158
10.6	Spanish	162
10.7	Dutch	166
10.8	Italian	171
10.9	Russian	175
10.10	Swedish	198
	Index	202

1 Initial setup

Start the DocStrip guards.

¹ `<*package>`

Identify the internal prefix (L^AT_EX3 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} {2024-11-28} {0.5.1}
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-variant }
174 {
175   Can't~set~variant~'#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-variants-ref }
180 {
181   Language~'#1'~has~no~declared~variants~\msg_line_context:~
182   Nothing~to~do~with~option~'v=#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-variants-setup }
```

```

190 {
191   Language~'#1'~has~no~declared~variants~\msg_line_context:..
192   Nothing~to~do~with~option~'variant=#2'.
193 }
194 \msg_new:nnn { zref-clever } { unknown-variant }
195 {
196   Variant~'#1'~unknown~for~language~'#2'~\msg_line_context:..
197   Using~default~variant.
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. 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^AT_EX3 programming layer does not have the concept of a variable *existing* only locally, it also considers an “error” if an assignment is made to a variable which was not previously declared, but declaration is always global, which means that “setting a local variable at a local scope”, given these requirements, results in it existing, and being empty, globally. Therefore, we need an independent mechanism from the mere existence of a variable to keep track of whether variables are “set” or “unset”, within the logic of the precedence rules for options in different scopes. __zrefclever_opt_var_set_bool:n expands to the name of the boolean variable used to track this state for <option var>. See discussion with Phelype Oleinik at https://tex.stackexchange.com/questions/633341/#comment1579825_633347

```
    \_zrefclever_opt_var_set_bool:n {<option var>}
```

```
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), 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.

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

```
\l__zrefclever_setup_type_tl
\l_zrefclever_setup_language_tl
\l_zrefclever_lang_variant_tl
\l_zrefclever_lang_variants_seq
\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_variant_tl
584 \seq_new:N \l__zrefclever_lang_variants_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^AT_EX3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```

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

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

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

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

```
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     variants .code:n =
786     {
787         \seq_new:c
788         {
789             \_zrefclever_opt_varname_language:enn
790             { \l__zrefclever_setup_language_tl } { variants } { seq }
791         }
792         \seq_gset_from_clist:cn
793         {
794             \_zrefclever_opt_varname_language:enn
795             { \l__zrefclever_setup_language_tl } { variants } { seq }
796         }
797         {#1}
798     } ,
799     variants .value_required:n = true ,
800     % NOTE Option deprecated in 2024-11-24 for v0.5.0.
801     declension .meta:n = { variants = {#1} } ,
802     gender .code:n =
803     {
804         \seq_new:c
805         {
806             \_zrefclever_opt_varname_language:enn
807             { \l__zrefclever_setup_language_tl } { gender } { seq }
808         }
809         \seq_gset_from_clist:cn
810         {
811             \_zrefclever_opt_varname_language:enn
812             { \l__zrefclever_setup_language_tl } { gender } { seq }
813         }
814         {#1}
815     } ,
816     gender .value_required:n = true ,
817     allcaps .choices:nn =
818     { true , false }
819     {
820         \bool_new:c
821         {
822             \_zrefclever_opt_varname_language:enn
823             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
824         }

```

```

825     \use:c { bool_gset_ \l_keys_choice_tl :c }
826     {
827         \__zrefclever_opt_varname_language:enn
828         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
829     }
830 } ,
831 allcaps .default:n = true ,
832 }

```

`__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 variant, 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_variant_tl` are in place.

```

833 \cs_new_protected:Npn \__zrefclever_process_language_settings:
834 {
835     \__zrefclever_language_if_declared:eTF
836     { \l__zrefclever_ref_language_tl }
837     {

```

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

```

838     \__zrefclever_opt_seq_get:cNF
839     {
840         \__zrefclever_opt_varname_language:enn
841         { \l__zrefclever_ref_language_tl } { variants } { seq }
842     }
843     \l__zrefclever_lang_variants_seq
844     { \seq_clear:N \l__zrefclever_lang_variants_seq }
845     \seq_if_empty:NTF \l__zrefclever_lang_variants_seq
846     {
847         \tl_if_empty:NF \l__zrefclever_ref_variant_tl
848         {
849             \msg_warning:nnee { zref-clever }
850             { language-no-variants-ref }
851             { \l__zrefclever_ref_language_tl }
852             { \l__zrefclever_ref_variant_tl }
853             \tl_clear:N \l__zrefclever_ref_variant_tl
854         }
855     }
856     {
857         \tl_if_empty:NTF \l__zrefclever_ref_variant_tl
858         {
859             \seq_get_left:NN \l__zrefclever_lang_variants_seq
860             \l__zrefclever_ref_variant_tl
861         }

```

```

862     {
863         \seq_if_in:NVF \l__zrefclever_lang_variants_seq
864         \l__zrefclever_ref_variant_tl
865         {
866             \msg_warning:nnee { zref-clever }
867             { unknown-variant }
868             { \l__zrefclever_ref_variant_tl }
869             { \l__zrefclever_ref_language_tl }
870             \seq_get_left:NN \l__zrefclever_lang_variants_seq
871             \l__zrefclever_ref_variant_tl
872         }
873     }
874 }

```

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.

```

875     \__zrefclever_opt_seq_get:cNF
876     {
877         \__zrefclever_opt_varname_language:enn
878         { \l__zrefclever_ref_language_tl } { gender } { seq }
879     }
880     \l__zrefclever_lang_gender_seq
881     { \seq_clear:N \l__zrefclever_lang_gender_seq }
882     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
883     {
884         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
885         {
886             \msg_warning:nneee { zref-clever }
887             { language-no-gender }
888             { \l__zrefclever_ref_language_tl }
889             { g }
890             { \l__zrefclever_ref_gender_tl }
891             \tl_clear:N \l__zrefclever_ref_gender_tl
892         }
893     }
894     {
895         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
896         {
897             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
898             \l__zrefclever_ref_gender_tl
899             {
900                 \msg_warning:nnee { zref-clever }
901                 { gender-not-declared }
902                 { \l__zrefclever_ref_language_tl }
903                 { \l__zrefclever_ref_gender_tl }
904                 \tl_clear:N \l__zrefclever_ref_gender_tl
905             }
906         }
907     }

```

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

```

908     \__zrefclever_opt_bool_if:cT
909     {

```

```

910         \_zrefclever_opt_varname_language:enn
911         { \l__zrefclever_ref_language_tl } { allcaps } { bool }
912     }
913     { \keys_set:nn { zref-clever/reference } { cap = true } }
914 }
915 {

```

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

```

916     \tl_if_empty:NF \l__zrefclever_ref_variant_tl
917     {
918         \msg_warning:nnee { zref-clever } { unknown-language-variant }
919         { \l__zrefclever_ref_variant_tl }
920         { \l__zrefclever_ref_language_tl }
921         \tl_clear:N \l__zrefclever_ref_variant_tl
922     }
923     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
924     {
925         \msg_warning:nnee { zref-clever }
926         { language-no-gender }
927         { \l__zrefclever_ref_language_tl }
928         { g }
929         { \l__zrefclever_ref_gender_tl }
930         \tl_clear:N \l__zrefclever_ref_gender_tl
931     }
932 }
933 }

```

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

this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load language files on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it. Hence, `zref-clever`’s built-in language files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/langfile}` by `__zrefclever__provide_langfile:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The language file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`__zrefclever__provide_langfile:n` is only meant to load the built-in language files. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to 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.

```
934 \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 $\langle\text{language}\rangle$ if it is available and if it has not already been loaded.

```

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

```

```

963         \seq_if_empty:NTF \l__zrefclever_lang_variants_seq
964         { \tl_clear:N \l__zrefclever_lang_variant_tl }
965         {
966             \seq_get_left:NN \l__zrefclever_lang_variants_seq
967             \l__zrefclever_lang_variant_tl
968         }
969     \__zrefclever_opt_seq_get:cNF
970     {
971         \__zrefclever_opt_varname_language:nnn
972         {#1} { gender } { seq }
973     }
974     \l__zrefclever_lang_gender_seq
975     { \seq_clear:N \l__zrefclever_lang_gender_seq }
976     \keys_set:nV { zref-clever/langfile } \l__zrefclever_tmpa_tl
977     \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
978     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
979     \msg_info:nne { zref-clever } { langfile-loaded }
980     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
981 }
982 {

```

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.

```

983         \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
984         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
985     }
986 }
987 }
988 \esphack
989 \group_end:
990 }
991 \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.

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

1330         }
1331     }
1332 } ,
1333 } ,
1334 }
1335 \seq_map_inline:Nn
1336 \g__zrefclever_rf_opts_tl_type_names_seq
1337 {
1338     \keys_define:nn { zref-clever/langfile }
1339     {
1340         #1 .value_required:n = true ,
1341         #1 .code:n =
1342         {
1343             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1344             {
1345                 \msg_info:nnn { zref-clever }
1346                 { option-only-type-specific } {#1}
1347             }
1348             {
1349                 \tl_if_empty:NTF \l__zrefclever_lang_variant_tl
1350                 {
1351                     \__zrefclever_opt_tl_gset_if_new:cn
1352                     {
1353                         \__zrefclever_opt_varname_lang_type:eenn
1354                         { \l__zrefclever_setup_language_tl }
1355                         { \l__zrefclever_setup_type_tl }
1356                         {#1} { t1 }
1357                     }
1358                     {##1}
1359                 }
1360                 {
1361                     \__zrefclever_opt_tl_gset_if_new:cn
1362                     {
1363                         \__zrefclever_opt_varname_lang_type:eeen
1364                         { \l__zrefclever_setup_language_tl }
1365                         { \l__zrefclever_setup_type_tl }
1366                         { \l__zrefclever_lang_variant_tl - #1 } { t1 }
1367                     }
1368                     {##1}
1369                 }
1370             }
1371         } ,
1372     }
1373 }
1374 \seq_map_inline:Nn
1375 \g__zrefclever_rf_opts_seq_refbounds_seq
1376 {
1377     \keys_define:nn { zref-clever/langfile }
1378     {
1379         #1 .value_required:n = true ,
1380         #1 .code:n =
1381         {
1382             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1383             {

```

```

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

```



```

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

```

```

1492     }
1493   }
1494 } ,
1495 #1 / false .code:n =
1496 {
1497   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1498   {
1499     \__zrefclever_opt_bool_if_set:cF
1500     {
1501       \__zrefclever_opt_varname_lang_default:enn
1502       { \l__zrefclever_setup_language_tl }
1503       {#1} { bool }
1504     }
1505     {
1506       \__zrefclever_opt_bool_gset_false:c
1507       {
1508         \__zrefclever_opt_varname_lang_default:enn
1509         { \l__zrefclever_setup_language_tl }
1510         {#1} { bool }
1511       }
1512     }
1513   }
1514 }
1515 {
1516   \__zrefclever_opt_bool_if_set:cF
1517   {
1518     \__zrefclever_opt_varname_lang_type:eenn
1519     { \l__zrefclever_setup_language_tl }
1520     { \l__zrefclever_setup_type_tl }
1521     {#1} { bool }
1522   }
1523   {
1524     \__zrefclever_opt_bool_gset_false:c
1525     {
1526       \__zrefclever_opt_varname_lang_type:eenn
1527       { \l__zrefclever_setup_language_tl }
1528       { \l__zrefclever_setup_type_tl }
1529       {#1} { bool }
1530     }
1531   }
1532 }
1533 } ,
1534 #1 .default:n = true ,
1535 no #1 .meta:n = { #1 = false } ,
1536 no #1 .value_forbidden:n = true ,
1537 }
1538 }

```

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.

```

1539 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1540 {
1541   \tl_const:cn
1542     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1543 }
1544 \keyval_parse:nnn
1545 { }
1546 { \__zrefclever_opt_tl_cset_fallback:nn }
1547 {
1548   tpairsep = {,~} ,
1549   tlistsep = {,~} ,
1550   tlastsep = {,~} ,
1551   notesep  = {~} ,
1552   namesep  = {\nobreakspace} ,
1553   pairsep  = {,~} ,
1554   listsep  = {,~} ,
1555   lastsep  = {,~} ,
1556   rangeseq = {\textendash} ,
1557 }

```

4.8 Options

Auxiliary

`__zrefclever_prop_put_non_empty:Nnn` If $\langle value \rangle$ is empty, remove $\langle key \rangle$ from $\langle property list \rangle$. Otherwise, add $\langle key \rangle = \langle value \rangle$ to $\langle property list \rangle$.

```

\__zrefclever_prop_put_non_empty:Nnn \langle property list \rangle { \langle key \rangle } { \langle value \rangle }

1558 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1559 {
1560   \tl_if_empty:nTF {#3}
1561     { \prop_remove:Nn #1 {#2} }
1562     { \prop_put:Nnn #1 {#2} {#3} }
1563 }

```

(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`.

```

1564 \tl_new:N \l__zrefclever_ref_property_tl
1565 \keys_define:nn { zref-clever/reference }
1566 {
1567   ref .code:n =
1568   {

```

```

1569     \tl_if_empty:nTF {#1}
1570     {
1571         \msg_warning:nnn { zref-clever }
1572         { zref-property-undefined } {#1}
1573         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1574     }
1575     {
1576         \zref@ifpropundefined {#1}
1577         {
1578             \msg_warning:nnn { zref-clever }
1579             { zref-property-undefined } {#1}
1580             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1581         }
1582         { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1583     }
1584 },
1585 ref .initial:n = default ,
1586 ref .value_required:n = true ,
1587 page .meta:n = { ref = page },
1588 page .value_forbidden:n = true ,
1589 }

```

typeset option

```

1590 \bool_new:N \l__zrefclever_typeset_ref_bool
1591 \bool_new:N \l__zrefclever_typeset_name_bool
1592 \keys_define:nn { zref-clever/reference }
1593 {
1594     typeset .choice: ,
1595     typeset / both .code:n =
1596     {
1597         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1598         \bool_set_true:N \l__zrefclever_typeset_name_bool
1599     } ,
1600     typeset / ref .code:n =
1601     {
1602         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1603         \bool_set_false:N \l__zrefclever_typeset_name_bool
1604     } ,
1605     typeset / name .code:n =
1606     {
1607         \bool_set_false:N \l__zrefclever_typeset_ref_bool
1608         \bool_set_true:N \l__zrefclever_typeset_name_bool
1609     } ,
1610     typeset .initial:n = both ,
1611     typeset .value_required:n = true ,
1612     noname .meta:n = { typeset = ref } ,
1613     noname .value_forbidden:n = true ,
1614     noref .meta:n = { typeset = name } ,
1615     noref .value_forbidden:n = true ,
1616 }

```

sort option

```

1617 \bool_new:N \l__zrefclever_typeset_sort_bool

```

```

1618 \keys_define:nn { zref-clever/reference }
1619 {
1620   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1621   sort .initial:n = true ,
1622   sort .default:n = true ,
1623   nosort .meta:n = { sort = false },
1624   nosort .value_forbidden:n = true ,
1625 }

```

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`.

```

1626 \seq_new:N \l__zrefclever_typesort_seq
1627 \keys_define:nn { zref-clever/reference }
1628 {
1629   typesort .code:n =
1630   {
1631     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1632     \seq_reverse:N \l__zrefclever_typesort_seq
1633   } ,
1634   typesort .initial:n =
1635   { part , chapter , section , paragraph },
1636   typesort .value_required:n = true ,
1637   notypesort .code:n =
1638   { \seq_clear:N \l__zrefclever_typesort_seq } ,
1639   notypesort .value_forbidden:n = true ,
1640 }

```

comp option

```

1641 \bool_new:N \l__zrefclever_typeset_compress_bool
1642 \keys_define:nn { zref-clever/reference }
1643 {
1644   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1645   comp .initial:n = true ,
1646   comp .default:n = true ,
1647   nocomp .meta:n = { comp = false },
1648   nocomp .value_forbidden:n = true ,
1649 }

```

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 `<tl var to set>`. Of course, `<tl`

`var to set`) must be set to a proper value, and that’s the main task of the function. `endrangefunc` must also handle the case where `\zref@ifrefcontainsprop` is false, since `__zrefclever_get_ref_endrange:nnN` cannot take care of that. For this purpose, it may set `<t1 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_t1: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`.

```

1650 \NewHook { zref-clever/endrange-setup }
1651 \keys_define:nn { zref-clever/reference }
1652 {
1653   endrange .code:n =
1654   {
1655     \str_case:nnF {#1}
1656     {
1657       { ref }
1658       {
1659         \__zrefclever_opt_t1_clear:c
1660         {
1661           \__zrefclever_opt_varname_general:nn
1662           { endrangefunc } { t1 }
1663         }
1664         \__zrefclever_opt_t1_clear:c
1665         {
1666           \__zrefclever_opt_varname_general:nn
1667           { endrangeprop } { t1 }
1668         }
1669       }
1670     } { stripprefix }
1671     {
1672       \__zrefclever_opt_t1_set:cn
1673       {
1674         \__zrefclever_opt_varname_general:nn
1675         { endrangefunc } { t1 }
1676       }
1677       { \__zrefclever_get_endrange_stripprefix }
1678     } \__zrefclever_opt_t1_clear:c
1679     {

```

```

1680         \_zrefclever_opt_varname_general:nn
1681         { endrangeprop } { t1 }
1682     }
1683 }
1684 { pagecomp }
1685 {
1686     \_zrefclever_opt_t1_set:cn
1687     {
1688         \_zrefclever_opt_varname_general:nn
1689         { endrangefunc } { t1 }
1690     }
1691     { \_zrefclever_get_endrange_pagecomp }
1692     \_zrefclever_opt_t1_clear:c
1693     {
1694         \_zrefclever_opt_varname_general:nn
1695         { endrangeprop } { t1 }
1696     }
1697 }
1698 { pagecomp2 }
1699 {
1700     \_zrefclever_opt_t1_set:cn
1701     {
1702         \_zrefclever_opt_varname_general:nn
1703         { endrangefunc } { t1 }
1704     }
1705     { \_zrefclever_get_endrange_pagecomptwo }
1706     \_zrefclever_opt_t1_clear:c
1707     {
1708         \_zrefclever_opt_varname_general:nn
1709         { endrangeprop } { t1 }
1710     }
1711 }
1712 { unset }
1713 {
1714     \_zrefclever_opt_t1_unset:c
1715     {
1716         \_zrefclever_opt_varname_general:nn
1717         { endrangefunc } { t1 }
1718     }
1719     \_zrefclever_opt_t1_unset:c
1720     {
1721         \_zrefclever_opt_varname_general:nn
1722         { endrangeprop } { t1 }
1723     }
1724 }
1725 }
1726 {
1727     \t1_if_empty:nTF {#1}
1728     {
1729         \msg_warning:nnn { zref-clever }
1730         { endrange-property-undefined } {#1}
1731     }
1732     {
1733         \zref@ifpropundefined {#1}

```

```

1734         {
1735             \msg_warning:nnn { zref-clever }
1736             { endrange-property-undefined } {#1}
1737         }
1738         {
1739             \__zrefclever_opt_tl_set:cn
1740             {
1741                 \__zrefclever_opt_varname_general:nn
1742                 { endrangefunc } { t1 }
1743             }
1744             { \__zrefclever_get_endrange_property }
1745             \__zrefclever_opt_tl_set:cn
1746             {
1747                 \__zrefclever_opt_varname_general:nn
1748                 { endrangeprop } { t1 }
1749             }
1750             {#1}
1751         }
1752     }
1753 } ,
1754 endrange .value_required:n = true ,
1755 }
1756
1757 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1758 {
1759     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1760     {
1761         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1762         {
1763             \__zrefclever_extract_default:Nnvn #3
1764             {#2} { \l__zrefclever_ref_property_tl } { }
1765         }
1766         { \tl_set:Nn #3 { zc@missingproperty } }
1767     }
1768     {
1769         \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1770         {

```

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.

```

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

```



```

1782         {#2} { externaldocument } { }
1783     }
1784     {
1785     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1786     {
1787         \exp_args:Nee \tl_if_eq:nnT
1788         {
1789             \__zrefclever_extract_unexp:nnn
1790             {#1} { zc@pgfmt } { }
1791         }
1792         {
1793             \__zrefclever_extract_unexp:nnn
1794             {#2} { zc@pgfmt } { }
1795         }
1796         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1797     }
1798     {
1799         \exp_args:Nee \tl_if_eq:nnT
1800         {
1801             \__zrefclever_extract_unexp:nnn
1802             {#1} { zc@counter } { }
1803         }
1804         {
1805             \__zrefclever_extract_unexp:nnn
1806             {#2} { zc@counter } { }
1807         }
1808         {
1809             \exp_args:Nee \tl_if_eq:nnT
1810             {
1811                 \__zrefclever_extract_unexp:nnn
1812                 {#1} { zc@enclval } { }
1813             }
1814             {
1815                 \__zrefclever_extract_unexp:nnn
1816                 {#2} { zc@enclval } { }
1817             }
1818             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1819         }
1820     }
1821 }
1822 \bool_if:NTF \l__zrefclever_tmpa_bool
1823 {
1824     \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1825     {#2} { l__zrefclever_endrangeprop_tl } { }
1826 }
1827 {
1828     \zref@ifrefcontainsprop
1829     {#2} { \l__zrefclever_ref_property_tl }
1830     {
1831         \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1832         {#2} { l__zrefclever_ref_property_tl } { }
1833     }
1834     { \tl_set:Nn \l__zrefclever_tmpb_tl { zc@missingproperty } }
1835 }

```

```

1836         \exp_args:NNNV
1837         \group_end:
1838         \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1839     }
1840     {
1841         \__zrefclever_extract_default:Nvn #3
1842         {#2} { \l__zrefclever_endrangeprop_tl } { }
1843     }
1844 }
1845 {
1846     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1847     {
1848         \__zrefclever_extract_default:Nvn #3
1849         {#2} { \l__zrefclever_ref_property_tl } { }
1850     }
1851     { \tl_set:Nn #3 { zc@missingproperty } }
1852 }
1853 }
1854 }
1855 \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>.

```

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

```

```

1887     }
1888     \exp_args:NNNV
1889     \group_end:
1890     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1891   }
1892   { \tl_set:Nn #3 { zc@missingproperty } }
1893 }
1894 \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>).

```

1895 \prg_new_protected_conditional:Npnn
1896   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1897   {
1898     \regex_match:nnTF { \A\d+\Z } {#1}
1899     { \prg_return_true: }
1900     { \prg_return_false: }
1901   }
1902 \prg_generate_conditional_variant:Nnn
1903   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End of definition for __zrefclever_is_integer_rgx:n.)

```

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

```

```

1936         { \tl_tail:V \l__zrefclever_tmpb_tl }
1937         \tl_if_empty:NT \l__zrefclever_tmpb_tl
1938         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1939     }
1940     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1941 }
1942 \exp_args:NNNV
1943 \group_end:
1944 \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1945 }
1946 { \tl_set:Nn #3 { zc@missingproperty } }
1947 }
1948 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1949 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1950 {
1951     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1952     {
1953         \group_begin:
1954         \UseHook { zref-clever/endrange-setup }
1955         \protected@edef \l__zrefclever_tmpa_tl
1956         {
1957             \__zrefclever_extract:nnn
1958             {#1} { \l__zrefclever_ref_property_tl } { }
1959         }
1960         \protected@edef \l__zrefclever_tmpb_tl
1961         {
1962             \__zrefclever_extract:nnn
1963             {#2} { \l__zrefclever_ref_property_tl } { }
1964         }
1965         \bool_set_false:N \l__zrefclever_tmpa_bool
1966         \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1967         {
1968             \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1969             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1970         }
1971         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1972         \bool_until_do:Nn \l__zrefclever_tmpa_bool
1973         {
1974             \exp_args:Nee \tl_if_eq:nnTF
1975             { \tl_head:V \l__zrefclever_tmpa_tl }
1976             { \tl_head:V \l__zrefclever_tmpb_tl }
1977             {
1978                 \bool_lazy_or:nnTF
1979                 { \int_compare_p:nNn { \l__zrefclever_tmpb_tl } > { 99 } }
1980                 {
1981                     \int_compare_p:nNn
1982                     { \tl_head:V \l__zrefclever_tmpb_tl } = { 0 }
1983                 }
1984             }
1985             \tl_set:Ne \l__zrefclever_tmpa_tl
1986             { \tl_tail:V \l__zrefclever_tmpa_tl }
1987             \tl_set:Ne \l__zrefclever_tmpb_tl
1988             { \tl_tail:V \l__zrefclever_tmpb_tl }
1989         }

```

```

1990             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1991         }
1992         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1993     }
1994     \exp_args:NNNV
1995     \group_end:
1996     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1997 }
1998 { \tl_set:Nn #3 { zc@missingproperty } }
1999 }
2000 \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`.

```

2001 \bool_new:N \l__zrefclever_typeset_range_bool
2002 \keys_define:mn { zref-clever/reference }
2003 {
2004     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
2005     range .initial:n = false ,
2006     range .default:n = true ,
2007 }

```

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`.

```

2008 \bool_new:N \l__zrefclever_capfirst_bool
2009 \keys_define:mn { zref-clever/reference }
2010 {
2011     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
2012     capfirst .initial:n = false ,
2013     capfirst .default:n = true ,
2014 }

```

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`.

```

2015 \bool_new:N \l__zrefclever_noabbrev_first_bool
2016 \keys_define:mn { zref-clever/reference }
2017 {
2018     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
2019     noabbrevfirst .initial:n = false ,
2020     noabbrevfirst .default:n = true ,
2021 }

```

S option

```
2022 \keys_define:nn { zref-clever/reference }
2023 {
2024   S .meta:n =
2025     { capfirst = {#1} , noabbrevfirst = {#1} },
2026   S .default:n = true ,
2027 }
```

hyperref option

```
2028 \bool_new:N \l__zrefclever_hyperlink_bool
2029 \bool_new:N \l__zrefclever_hyperref_warn_bool
2030 \keys_define:nn { zref-clever/reference }
2031 {
2032   hyperref .choice: ,
2033   hyperref / auto .code:n =
2034     {
2035       \bool_set_true:N \l__zrefclever_hyperlink_bool
2036       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2037     } ,
2038   hyperref / true .code:n =
2039     {
2040       \bool_set_true:N \l__zrefclever_hyperlink_bool
2041       \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2042     } ,
2043   hyperref / false .code:n =
2044     {
2045       \bool_set_false:N \l__zrefclever_hyperlink_bool
2046       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2047     } ,
2048   hyperref .initial:n = auto ,
2049   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.

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

```

2066     {
2067         hyperref .code:n =
2068         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2069         nohyperref .code:n =
2070         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2071     }
2072 }

```

nameinlink option

```

2073 \str_new:N \l__zrefclever_nameinlink_str
2074 \keys_define:nn { zref-clever/reference }
2075 {
2076     nameinlink .choice: ,
2077     nameinlink / true .code:n =
2078     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2079     nameinlink / false .code:n =
2080     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2081     nameinlink / single .code:n =
2082     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2083     nameinlink / tsingle .code:n =
2084     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2085     nameinlink .initial:n = tsingle ,
2086     nameinlink .default:n = true ,
2087 }

```

preposinlink option (deprecated)

```

2088 \keys_define:nn { zref-clever/reference }
2089 {
2090     preposinlink .code:n =
2091     {
2092         % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2093         \msg_warning:nnnn { zref-clever } { option-deprecated }
2094         { preposinlink } { refbounds }
2095     } ,
2096 }

```

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`.

```

2097 \AddToHook { begindocument }
2098 {
2099   \__zrefclever_if_package_loaded:nTF { babel }
2100   {
2101     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
2102     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2103   }
2104   {
2105     \__zrefclever_if_package_loaded:nTF { polyglossia }
2106     {
2107       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2108       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2109     }
2110     {
2111       \tl_set:Nn \l__zrefclever_current_language_tl { english }
2112       \tl_set:Nn \l__zrefclever_main_language_tl { english }
2113     }
2114   }
2115 }
2116 \keys_define:nn { zref-clever/reference }
2117 {
2118   lang .code:n =
2119   {
2120     \AddToHook { begindocument }
2121     {
2122       \str_case:nnF {#1}
2123       {
2124         { current }
2125         {
2126           \tl_set:Nn \l__zrefclever_ref_language_tl
2127             { \l__zrefclever_current_language_tl }
2128         }
2129         { main }
2130         {
2131           \tl_set:Nn \l__zrefclever_ref_language_tl
2132             { \l__zrefclever_main_language_tl }
2133         }
2134       }
2135       {
2136         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2137         \__zrefclever_language_if_declared:nF {#1}
2138         {
2139           \msg_warning:nnn { zref-clever }
2140             { unknown-language-opt } {#1}
2141         }
2142       }
2143     }
2144     \__zrefclever_provide_langfile:e
2145     { \l__zrefclever_ref_language_tl }
2146   } ,

```



```

2147     lang .initial:n = current ,
2148     lang .value_required:n = true ,
2149   }
2150 \AddToHook { begindocument / before }
2151 {
2152   \AddToHook { begindocument }
2153     {

```

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.

```

2154     \keys_define:nn { zref-clever/reference }
2155     {
2156       lang .code:n =
2157         {
2158           \str_case:nnF {#1}
2159             {
2160               { current }
2161               {
2162                 \tl_set:Nn \l__zrefclever_ref_language_tl
2163                   { \l__zrefclever_current_language_tl }
2164               }
2165               { main }
2166               {
2167                 \tl_set:Nn \l__zrefclever_ref_language_tl
2168                   { \l__zrefclever_main_language_tl }
2169               }
2170             }
2171           {
2172             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2173             \__zrefclever_language_if_declared:nF {#1}
2174             {
2175               \msg_warning:nnn { zref-clever }
2176                 { unknown-language-opt } {#1}
2177             }
2178           }
2179         } ,
2180     }
2181   }
2182 }

```

v option

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

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

```

2183 \tl_new:N \l__zrefclever_ref_variant_tl
2184 \keys_define:nn { zref-clever/reference }
2185 {
2186   v .code:n =

```

```

2187     { \msg_warning:nnn { zref-clever } { option-document-only } { v } } ,
2188     % NOTE Option deprecated in 2024-11-24 for v0.5.0.
2189     d .meta:n = { v = {#1} } ,
2190   }
2191 \AddToHook { begindocument }
2192 {
2193   \keys_define:nn { zref-clever/reference }
2194   {

```

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

```

2195     v .tl_set:N = \l__zrefclever_ref_variant_tl ,
2196     v .value_required:n = true ,
2197     % NOTE Option deprecated in 2024-11-24 for v0.5.0.
2198     d .meta:n = { v = {#1} } ,
2199   }
2200 }

```

nudge & co. options

```

2201 \bool_new:N \l__zrefclever_nudge_enabled_bool
2202 \bool_new:N \l__zrefclever_nudge_multitype_bool
2203 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2204 \bool_new:N \l__zrefclever_nudge_singular_bool
2205 \bool_new:N \l__zrefclever_nudge_gender_bool
2206 \tl_new:N \l__zrefclever_ref_gender_tl
2207 \keys_define:nn { zref-clever/reference }
2208 {
2209   nudge .choice: ,
2210   nudge / true .code:n =
2211     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2212   nudge / false .code:n =
2213     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2214   nudge / ifdraft .code:n =
2215     {
2216       \ifdraft
2217         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2218         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2219       } ,
2220   nudge / iffinaal .code:n =
2221     {
2222       \ifoptionfinal
2223         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2224         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2225       } ,
2226   nudge .initial:n = false ,
2227   nudge .default:n = true ,
2228   nonudge .meta:n = { nudge = false } ,
2229   nonudge .value_forbidden:n = true ,
2230   nudgeif .code:n =
2231     {
2232       \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2233       \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2234       \bool_set_false:N \l__zrefclever_nudge_gender_bool

```

```

2235 \clist_map_inline:nn {#1}
2236 {
2237   \str_case:nnF {##1}
2238   {
2239     { multitype }
2240     { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2241     { comptosing }
2242     { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2243     { gender }
2244     { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2245     { all }
2246     {
2247       \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2248       \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2249       \bool_set_true:N \l__zrefclever_nudge_gender_bool
2250     }
2251   }
2252   {
2253     \msg_warning:nnn { zref-clever }
2254     { nudgeif-unknown-value } {##1}
2255   }
2256 }
2257 } ,
2258 nudgeif .value_required:n = true ,
2259 nudgeif .initial:n = all ,
2260 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2261 sg .initial:n = false ,
2262 sg .default:n = true ,
2263 g .code:n =
2264 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2265 }
2266 \AddToHook { begindocument }
2267 {
2268   \keys_define:nn { zref-clever/reference }
2269   {

```

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

```

2270   g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2271   g .value_required:n = true ,
2272 }
2273 }

```

font option

```

2274 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2275 \keys_define:nn { zref-clever/reference }
2276 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2277 \keys_define:nn { zref-clever/reference }
2278 {
2279   titleref .code:n =
2280   {
2281     % NOTE Option deprecated in 2022-04-22 for 0.3.0.

```

```

2282     \msg_warning:nnee { zref-clever }{ option-deprecated } { titleref }
2283     { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2284   } ,
2285 }

```

vario option

```

2286 \keys_define:nn { zref-clever/reference }
2287 {
2288   vario .code:n =
2289   {
2290     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2291     \msg_warning:nnee { zref-clever }{ option-deprecated } { vario }
2292     { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2293   } ,
2294 }

```

note option

```

2295 \tl_new:N \l__zrefclever_zceref_note_tl
2296 \keys_define:nn { zref-clever/reference }
2297 {
2298   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
2299   note .value_required:n = true ,
2300 }

```

check option

Integration with zref-check.

```

2301 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2302 \bool_new:N \l__zrefclever_zceref_with_check_bool
2303 \keys_define:nn { zref-clever/reference }
2304 {
2305   check .code:n =
2306   { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2307 }
2308 \AddToHook { begindocument }
2309 {
2310   \__zrefclever_if_package_loaded:nTF { zref-check }
2311   {
2312     \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2313     {
2314       \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2315       \keys_define:nn { zref-clever/reference }
2316       {
2317         check .code:n =
2318         {
2319           \bool_set_true:N \l__zrefclever_zceref_with_check_bool
2320           \keys_set:nn { zref-check/zcheck } {#1}
2321         } ,
2322         check .value_required:n = true ,
2323       }
2324     }
2325   }
2326   \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2327   \keys_define:nn { zref-clever/reference }

```

```

2328         {
2329             check .code:n =
2330                 {
2331                     \msg_warning:nnn { zref-clever }
2332                     { zref-check-too-old } { 2021-09-16~v0.2.1 }
2333                 } ,
2334             }
2335         }
2336     }
2337     {
2338         \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2339         \keys_define:nn { zref-clever/reference }
2340         {
2341             check .code:n =
2342             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2343         }
2344     }
2345 }

```

reftype option

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

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

```

2346 \tl_new:N \l__zrefclever_reftype_override_tl
2347 \keys_define:nn { zref-clever/label }
2348 {
2349     reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2350     reftype .default:n = {} ,
2351     reftype .initial:n = {} ,
2352 }

```

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`.

```

2353 \prop_new:N \l__zrefclever_counter_type_prop
2354 \keys_define:nn { zref-clever/label }
2355 {
2356     countertype .code:n =
2357     {
2358         \keyval_parse:nnn
2359         {
2360             \msg_warning:nmmm { zref-clever }
2361             { key-requires-value } { countertype }
2362         }
2363         {
2364             \__zrefclever_prop_put_non_empty:Nnn
2365             \l__zrefclever_counter_type_prop
2366         }

```

```

2367         {#1}
2368     } ,
2369     countertype .value_required:n = true ,
2370     countertype .initial:n =
2371     {
2372         subsection    = section ,
2373         subsubsection = section ,
2374         subparagraph  = paragraph ,
2375         enumi         = item ,
2376         enumii        = item ,
2377         enumiii       = item ,
2378         enumiv        = item ,
2379         mpfootnote    = footnote ,
2380     } ,
2381 }

```

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

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclval` property, and stores the list of counters which are potential “enclosing counters” for other counters.

Note that, as far as L^AT_EX 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_resettters_seq`, `chapter` will be returned, and that's not what we want. That's the reason `counterresettters` 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_resettters_seq` with the `counterresetby` option.

For the above reasons, since order matters, the `counterresettters` 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.

```

2382 \seq_new:N \l__zrefclever_counter_resettters_seq
2383 \keys_define:nn { zref-clever/label }
2384 {
2385   counterresettters .code:n =
2386     { \seq_set_from_clist:Nn \l__zrefclever_counter_resettters_seq {#1} } ,
2387   counterresettters .initial:n =
2388     {
2389       subparagraph ,
2390       paragraph ,
2391       subsubsection ,
2392       subsection ,
2393       section ,
2394       chapter ,
2395       part ,
2396     },
2397   counterresettters .value_required:n = true ,
2398 }

```

counterresetby option

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

```

2399 \prop_new:N \l__zrefclever_counter_resetby_prop
2400 \keys_define:nn { zref-clever/label }
2401 {
2402   counterresetby .code:n =
2403     {
2404       \keyval_parse:nnn
2405         {
2406           \msg_warning:nnn { zref-clever }
2407             { key-requires-value } { counterresetby }
2408         }
2409     }

```

```

2409     {
2410         \__zrefclever_prop_put_non_empty:Nnn
2411         \l__zrefclever_counter_resetby_prop
2412     }
2413     {#1}
2414 } ,
2415 counterresetby .value_required:n = true ,
2416 counterresetby .initial:n =
2417 {

```

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.

```

2418     enumii = enumi ,
2419     enumiii = enumii ,
2420     enumiv = enumiii ,
2421 } ,
2422 }

```

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.

```

2423 \tl_new:N \l__zrefclever_current_counter_tl
2424 \keys_define:nn { zref-clever/label }
2425 {
2426     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2427     currentcounter .default:n = \@currentcounter ,
2428     currentcounter .initial:n = \@currentcounter ,
2429 }

```

labelhook option

```

2430 \bool_new:N \l__zrefclever_labelhook_bool
2431 \keys_define:nn { zref-clever/label }
2432 {
2433     labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2434     labelhook .initial:n = true ,
2435     labelhook .default:n = true ,
2436 }

```

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.

```

2437 \AddToHookWithArguments { label }
2438 {
2439     \bool_if:NT \l__zrefclever_labelhook_bool
2440     { \zref@wrapper@babel \zref@label {#1} }

```



```

2441 }
nocompat option
2442 \bool_new:N \g__zrefclever_nocompat_bool
2443 \seq_new:N \g__zrefclever_nocompat_modules_seq
2444 \keys_define:nn { zref-clever/reference }
2445 {
2446   nocompat .code:n =
2447   {
2448     \tl_if_empty:nTF {#1}
2449     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2450     {
2451       \clist_map_inline:nn {#1}
2452       {
2453         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2454         {
2455           \seq_gput_right:Nn
2456             \g__zrefclever_nocompat_modules_seq {##1}
2457         }
2458       }
2459     }
2460   } ,
2461 }
2462 \AddToHook { begindocument }
2463 {
2464   \keys_define:nn { zref-clever/reference }
2465   {
2466     nocompat .code:n =
2467     {
2468       \msg_warning:nnn { zref-clever }
2469       { option-preamble-only } { nocompat }
2470     }
2471   }
2472 }
2473 \AtEndOfPackage
2474 {
2475   \AddToHook { begindocument }
2476   {
2477     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2478     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2479   }
2480 }

```

`_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>}
2481 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2482 {
2483   \AddToHook { begindocument }
2484   {
2485     \bool_if:NF \g__zrefclever_nocompat_bool
2486     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2487     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2488   }
2489 }

```

(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.

```

2490 \seq_map_inline:Nn
2491   \g__zrefclever_rf_opts_tl_reference_seq
2492   {
2493     \keys_define:nn { zref-clever/reference }
2494     {
2495       #1 .default:o = \c_novalue_tl ,
2496       #1 .code:n =
2497         {
2498           \tl_if_novalue:nTF {##1}
2499             {
2500               \__zrefclever_opt_tl_unset:c
2501               { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2502             }
2503             {
2504               \__zrefclever_opt_tl_set:cn
2505               { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2506               {##1}
2507             }
2508           } ,
2509     }
2510 }
2511 \keys_define:nn { zref-clever/reference }
2512 {
2513   refpre .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     { refpre } { refbounds }
2518   } ,
2519   refpos .code:n =
2520   {
2521     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2522     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2523     { refpos } { refbounds }
2524   } ,

```

```

2525 preref .code:n =
2526 {
2527   % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2528   \msg_warning:nnnn { zref-clever }{ option-deprecated }
2529   { preref } { refbounds }
2530 } ,
2531 postref .code:n =
2532 {
2533   % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2534   \msg_warning:nnnn { zref-clever }{ option-deprecated }
2535   { postref } { refbounds }
2536 } ,
2537 }
2538 \seq_map_inline:Nn
2539 \g__zrefclever_rf_opts_seq_refbounds_seq
2540 {
2541   \keys_define:nn { zref-clever/reference }
2542   {
2543     #1 .default:o = \c_novalue_tl ,
2544     #1 .code:n =
2545     {
2546       \tl_if_novalue:nTF {##1}
2547       {
2548         \__zrefclever_opt_seq_unset:c
2549         { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2550       }
2551       {
2552         \seq_clear:N \l__zrefclever_tmpa_seq
2553         \__zrefclever_opt_seq_set_clist_split:Nn
2554         \l__zrefclever_tmpa_seq {##1}
2555         \bool_lazy_or:nnTF
2556         { \tl_if_empty_p:n {##1} }
2557         {
2558           \int_compare_p:nNn
2559           { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2560         }
2561         {
2562           \__zrefclever_opt_seq_set_eq:cN
2563           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2564           \l__zrefclever_tmpa_seq
2565         }
2566         {
2567           \msg_warning:nnee { zref-clever }
2568           { refbounds-must-be-four }
2569           {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2570         }
2571       }
2572     } ,
2573   }
2574 }
2575 \seq_map_inline:Nn
2576 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2577 {
2578   \keys_define:nn { zref-clever/reference }

```

```

2579 {
2580   #1 .choice: ,
2581   #1 / true .code:n =
2582   {
2583     \__zrefclever_opt_bool_set_true:c
2584     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2585   } ,
2586   #1 / false .code:n =
2587   {
2588     \__zrefclever_opt_bool_set_false:c
2589     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2590   } ,
2591   #1 / unset .code:n =
2592   {
2593     \__zrefclever_opt_bool_unset:c
2594     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2595   } ,
2596   #1 .default:n = true ,
2597   no #1 .meta:n = { #1 = false } ,
2598   no #1 .value_forbidden:n = true ,
2599 }
2600 }

```

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>.

```

2601 \keys_define:nn { zref-clever }
2602 {
2603   zcsetup .inherit:n =
2604   {
2605     zref-clever/label ,
2606     zref-clever/reference ,
2607   }
2608 }

```

`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.

```

2609 \bool_lazy_and:nnT
2610 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2611 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2612 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```
\zcsetup{options}  
  
2613 \NewDocumentCommand \zcsetup { m }  
2614 { \__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}  
  
2615 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1  
2616 { \keys_set:nn { zref-clever/zcsetup } {#1} }  
2617 \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}  
  
2618 \NewDocumentCommand \zcRefTypeSetup { m m }  
2619 {  
2620 \tl_set:Nn \l__zrefclever_setup_type_tl {#1}  
2621 \keys_set:nn { zref-clever/typesetup } {#2}  
2622 \tl_clear:N \l__zrefclever_setup_type_tl  
2623 }  
  
(End of definition for \zcRefTypeSetup.)  
  
2624 \seq_map_inline:Nn  
2625 \g__zrefclever_rf_opts_tl_not_type_specific_seq  
2626 {  
2627 \keys_define:nn { zref-clever/typesetup }  
2628 {  
2629 #1 .code:n =  
2630 {  
2631 \msg_warning:nnn { zref-clever }  
2632 { option-not-type-specific } {#1}  
2633 } ,  
2634 }  
2635 }  
2636 \seq_map_inline:Nn
```

```

2637 \g__zrefclever_rf_opts_tl_typesetup_seq
2638 {
2639   \keys_define:nn { zref-clever/typesetup }
2640   {
2641     #1 .default:o = \c_novalue_tl ,
2642     #1 .code:n =
2643     {
2644       \tl_if_novalue:nTF {##1}
2645       {
2646         \__zrefclever_opt_tl_unset:c
2647         {
2648           \__zrefclever_opt_varname_type:enn
2649           { \l__zrefclever_setup_type_tl } {#1} { tl }
2650         }
2651       }
2652       {
2653         \__zrefclever_opt_tl_set:cn
2654         {
2655           \__zrefclever_opt_varname_type:enn
2656           { \l__zrefclever_setup_type_tl } {#1} { tl }
2657         }
2658         {##1}
2659       }
2660     } ,
2661   }
2662 }
2663 \keys_define:nn { zref-clever/typesetup }
2664 {
2665   endrange .code:n =
2666   {
2667     \str_case:nnF {#1}
2668     {
2669       { ref }
2670       {
2671         \__zrefclever_opt_tl_clear:c
2672         {
2673           \__zrefclever_opt_varname_type:enn
2674           { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2675         }
2676         \__zrefclever_opt_tl_clear:c
2677         {
2678           \__zrefclever_opt_varname_type:enn
2679           { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2680         }
2681       }
2682     } { stripprefix }
2683     {
2684       \__zrefclever_opt_tl_set:cn
2685       {
2686         \__zrefclever_opt_varname_type:enn
2687         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2688       }
2689       { __zrefclever_get_endrange_stripprefix }
2690     } \__zrefclever_opt_tl_clear:c

```

```

2691         {
2692             \__zrefclever_opt_varname_type:enn
2693             { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2694         }
2695     }
2696 { pagecomp }
2697 {
2698     \__zrefclever_opt_t1_set:cn
2699     {
2700         \__zrefclever_opt_varname_type:enn
2701         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2702     }
2703     { __zrefclever_get_endrange_pagecomp }
2704     \__zrefclever_opt_t1_clear:c
2705     {
2706         \__zrefclever_opt_varname_type:enn
2707         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2708     }
2709 }
2710 { pagecomp2 }
2711 {
2712     \__zrefclever_opt_t1_set:cn
2713     {
2714         \__zrefclever_opt_varname_type:enn
2715         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2716     }
2717     { __zrefclever_get_endrange_pagecomptwo }
2718     \__zrefclever_opt_t1_clear:c
2719     {
2720         \__zrefclever_opt_varname_type:enn
2721         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2722     }
2723 }
2724 { unset }
2725 {
2726     \__zrefclever_opt_t1_unset:c
2727     {
2728         \__zrefclever_opt_varname_type:enn
2729         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2730     }
2731     \__zrefclever_opt_t1_unset:c
2732     {
2733         \__zrefclever_opt_varname_type:enn
2734         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2735     }
2736 }
2737 }
2738 {
2739     \tl_if_empty:nTF {#1}
2740     {
2741         \msg_warning:nnn { zref-clever }
2742         { endrange-property-undefined } {#1}
2743     }
2744     {

```

```

2745         \zref@ifpropundefined {#1}
2746         {
2747             \msg_warning:nnn { zref-clever }
2748             { endrange-property-undefined } {#1}
2749         }
2750         {
2751             \__zrefclever_opt_tl_set:cn
2752             {
2753                 \__zrefclever_opt_varname_type:enn
2754                 { \l__zrefclever_setup_type_tl }
2755                 { endrangefunc } { tl }
2756             }
2757             { __zrefclever_get_endrange_property }
2758             \__zrefclever_opt_tl_set:cn
2759             {
2760                 \__zrefclever_opt_varname_type:enn
2761                 { \l__zrefclever_setup_type_tl }
2762                 { endrangeprop } { tl }
2763             }
2764             {#1}
2765         }
2766     }
2767 } ,
2768 endrange .value_required:n = true ,
2769 }
2770 \keys_define:nn { zref-clever/typesetup }
2771 {
2772     refpre .code:n =
2773     {
2774         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2775         \msg_warning:nxxx { zref-clever }{ option-deprecated }
2776         { refpre } { refbounds }
2777     } ,
2778     refpos .code:n =
2779     {
2780         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2781         \msg_warning:nxxx { zref-clever }{ option-deprecated }
2782         { refpos } { refbounds }
2783     } ,
2784     preref .code:n =
2785     {
2786         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2787         \msg_warning:nxxx { zref-clever }{ option-deprecated }
2788         { preref } { refbounds }
2789     } ,
2790     postref .code:n =
2791     {
2792         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2793         \msg_warning:nxxx { zref-clever }{ option-deprecated }
2794         { postref } { refbounds }
2795     } ,
2796 }
2797 }
2798 \seq_map_inline:Nn

```



```

2799 \g__zrefclever_rf_opts_seq_refbounds_seq
2800 {
2801   \keys_define:nn { zref-clever/typesetup }
2802   {
2803     #1 .default:o = \c_novalue_tl ,
2804     #1 .code:n =
2805     {
2806       \tl_if_novalue:nTF {##1}
2807       {
2808         \__zrefclever_opt_seq_unset:c
2809         {
2810           \__zrefclever_opt_varname_type:enn
2811           { \l__zrefclever_setup_type_tl } {#1} { seq }
2812         }
2813       }
2814       {
2815         \seq_clear:N \l__zrefclever_tmpa_seq
2816         \__zrefclever_opt_seq_set_clist_split:Nn
2817         \l__zrefclever_tmpa_seq {##1}
2818         \bool_lazy_or:nnTF
2819         { \tl_if_empty_p:n {##1} }
2820         {
2821           \int_compare_p:nNn
2822           { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2823         }
2824         {
2825           \__zrefclever_opt_seq_set_eq:cN
2826           {
2827             \__zrefclever_opt_varname_type:enn
2828             { \l__zrefclever_setup_type_tl } {#1} { seq }
2829           }
2830           \l__zrefclever_tmpa_seq
2831         }
2832         {
2833           \msg_warning:nnee { zref-clever }
2834           { refbounds-must-be-four }
2835           {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2836         }
2837       }
2838     } ,
2839   }
2840 }
2841 \seq_map_inline:Nn
2842 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2843 {
2844   \keys_define:nn { zref-clever/typesetup }
2845   {
2846     #1 .choice: ,
2847     #1 / true .code:n =
2848     {
2849       \__zrefclever_opt_bool_set_true:c
2850       {
2851         \__zrefclever_opt_varname_type:enn
2852         { \l__zrefclever_setup_type_tl }

```

```

2853         {#1} { bool }
2854     }
2855 },
2856 #1 / false .code:n =
2857 {
2858     \__zrefclever_opt_bool_set_false:c
2859     {
2860         \__zrefclever_opt_varname_type:enn
2861         { \l__zrefclever_setup_type_tl }
2862         {#1} { bool }
2863     }
2864 },
2865 #1 / unset .code:n =
2866 {
2867     \__zrefclever_opt_bool_unset:c
2868     {
2869         \__zrefclever_opt_varname_type:enn
2870         { \l__zrefclever_setup_type_tl }
2871         {#1} { bool }
2872     }
2873 },
2874 #1 .default:n = true ,
2875 no #1 .meta:n = { #1 = false } ,
2876 no #1 .value_forbidden:n = true ,
2877 }
2878 }

```

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>}
2879 \NewDocumentCommand \zcLanguageSetup { m m }
2880 {
2881   \group_begin:
2882   \__zrefclever_language_if_declared:nTF {#1}
2883   {
2884     \tl_clear:N \l__zrefclever_setup_type_tl
2885     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2886     \__zrefclever_opt_seq_get:cNF
2887     {
2888       \__zrefclever_opt_varname_language:nnn
2889       {#1} { variants } { seq }
2890     }
2891     \l__zrefclever_lang_variants_seq
2892     { \seq_clear:N \l__zrefclever_lang_variants_seq }
2893     \seq_if_empty:NTF \l__zrefclever_lang_variants_seq

```

```

2894         { \tl_clear:N \l__zrefclever_lang_variant_tl }
2895         {
2896           \seq_get_left:NN \l__zrefclever_lang_variants_seq
2897             \l__zrefclever_lang_variant_tl
2898         }
2899     \__zrefclever_opt_seq_get:cNF
2900     {
2901       \__zrefclever_opt_varname_language:nnn
2902         {#1} { gender } { seq }
2903     }
2904     \l__zrefclever_lang_gender_seq
2905     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2906     \keys_set:nn { zref-clever/langsetup } {#2}
2907 }
2908 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2909 \group_end:
2910 }
2911 \@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.

```

2912 \keys_define:nn { zref-clever/langsetup }
2913 {
2914   type .code:n =
2915   {
2916     \tl_if_empty:nTF {#1}
2917       { \tl_clear:N \l__zrefclever_setup_type_tl }
2918       { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2919   } ,
2920   variant .code:n =
2921   {
2922     \seq_if_empty:NTF \l__zrefclever_lang_variants_seq
2923     {
2924       \msg_warning:nnee { zref-clever } { language-no-variants-setup }
2925       { \l__zrefclever_setup_language_tl } {#1}
2926     }
2927     {
2928       \seq_if_in:NnTF \l__zrefclever_lang_variants_seq {#1}
2929       { \tl_set:Nn \l__zrefclever_lang_variant_tl {#1} }
2930       {
2931         \msg_warning:nnee { zref-clever } { unknown-variant }
2932         {#1} { \l__zrefclever_setup_language_tl }
2933         \seq_get_left:NN \l__zrefclever_lang_variants_seq
2934           \l__zrefclever_lang_variant_tl
2935       }
2936     }
2937   } ,
2938   variant .value_required:n = true ,
2939   % NOTE Option deprecated in 2024-11-24 for v0.5.0.
2940   case .meta:n = { variant = {#1} } ,
2941   gender .value_required:n = true ,
2942   gender .code:n =
2943   {

```

```

2944 \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2945 {
2946   \msg_warning:nneee { zref-clever } { language-no-gender }
2947   { \l__zrefclever_setup_language_tl } { gender } {#1}
2948 }
2949 {
2950   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2951   {
2952     \msg_warning:nnn { zref-clever }
2953     { option-only-type-specific } { gender }
2954   }
2955   {
2956     \seq_clear:N \l__zrefclever_tmpa_seq
2957     \clist_map_inline:nn {#1}
2958     {
2959       \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2960       { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
2961       {
2962         \msg_warning:nnee { zref-clever }
2963         { gender-not-declared }
2964         { \l__zrefclever_setup_language_tl } {##1}
2965       }
2966     }
2967     \__zrefclever_opt_seq_gset_eq:cN
2968     {
2969       \__zrefclever_opt_varname_lang_type:enn
2970       { \l__zrefclever_setup_language_tl }
2971       { \l__zrefclever_setup_type_tl }
2972       { gender }
2973       { seq }
2974     }
2975     \l__zrefclever_tmpa_seq
2976   }
2977 }
2978 } ,
2979 }
2980 \seq_map_inline:Nn
2981 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2982 {
2983   \keys_define:nn { zref-clever/langsetup }
2984   {
2985     #1 .value_required:n = true ,
2986     #1 .code:n =
2987     {
2988       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2989       {
2990         \__zrefclever_opt_tl_gset:cn
2991         {
2992           \__zrefclever_opt_varname_lang_default:enn
2993           { \l__zrefclever_setup_language_tl } {#1} { tl }
2994         }
2995         {##1}
2996       }
2997     }

```

```

2998         \msg_warning:nnn { zref-clever }
2999         { option-not-type-specific } {#1}
3000     } ,
3001 } ,
3002 }
3003 }
3004 \seq_map_inline:Nn
3005 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
3006 {
3007     \keys_define:nn { zref-clever/langsetup }
3008     {
3009         #1 .value_required:n = true ,
3010         #1 .code:n =
3011         {
3012             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3013             {
3014                 \__zrefclever_opt_tl_gset:cn
3015                 {
3016                     \__zrefclever_opt_varname_lang_default:enn
3017                     { \l__zrefclever_setup_language_tl } {#1} { tl }
3018                 }
3019                 {##1}
3020             }
3021             {
3022                 \__zrefclever_opt_tl_gset:cn
3023                 {
3024                     \__zrefclever_opt_varname_lang_type:eenn
3025                     { \l__zrefclever_setup_language_tl }
3026                     { \l__zrefclever_setup_type_tl }
3027                     {#1} { tl }
3028                 }
3029                 {##1}
3030             }
3031         } ,
3032     }
3033 }
3034 \keys_define:nn { zref-clever/langsetup }
3035 {
3036     endrange .value_required:n = true ,
3037     endrange .code:n =
3038     {
3039         \str_case:nnF {#1}
3040         {
3041             { ref }
3042             {
3043                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3044                 {
3045                     \__zrefclever_opt_tl_gclear:c
3046                     {
3047                         \__zrefclever_opt_varname_lang_default:enn
3048                         { \l__zrefclever_setup_language_tl }
3049                         { endrangefunc } { tl }
3050                     }
3051                     \__zrefclever_opt_tl_gclear:c

```

```

3052         {
3053             \_zrefclever_opt_varname_lang_default:enn
3054             { \l_zrefclever_setup_language_tl }
3055             { endrangeprop } { tl }
3056         }
3057     }
3058     {
3059         \_zrefclever_opt_tl_gclear:c
3060         {
3061             \_zrefclever_opt_varname_lang_type:eenn
3062             { \l_zrefclever_setup_language_tl }
3063             { \l_zrefclever_setup_type_tl }
3064             { endrangefunc } { tl }
3065         }
3066         \_zrefclever_opt_tl_gclear:c
3067         {
3068             \_zrefclever_opt_varname_lang_type:eenn
3069             { \l_zrefclever_setup_language_tl }
3070             { \l_zrefclever_setup_type_tl }
3071             { endrangeprop } { tl }
3072         }
3073     }
3074 }
3075 { stripprefix }
3076 {
3077     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3078     {
3079         \_zrefclever_opt_tl_gset:cn
3080         {
3081             \_zrefclever_opt_varname_lang_default:enn
3082             { \l_zrefclever_setup_language_tl }
3083             { endrangefunc } { tl }
3084         }
3085         { __zrefclever_get_endrange_stripprefix }
3086         \_zrefclever_opt_tl_gclear:c
3087         {
3088             \_zrefclever_opt_varname_lang_default:enn
3089             { \l_zrefclever_setup_language_tl }
3090             { endrangeprop } { tl }
3091         }
3092     }
3093     {
3094         \_zrefclever_opt_tl_gset:cn
3095         {
3096             \_zrefclever_opt_varname_lang_type:eenn
3097             { \l_zrefclever_setup_language_tl }
3098             { \l_zrefclever_setup_type_tl }
3099             { endrangefunc } { tl }
3100         }
3101         { __zrefclever_get_endrange_stripprefix }
3102         \_zrefclever_opt_tl_gclear:c
3103         {
3104             \_zrefclever_opt_varname_lang_type:eenn
3105             { \l_zrefclever_setup_language_tl }

```

```

3106         { \l__zrefclever_setup_type_tl }
3107         { endrangeprop } { tl }
3108     }
3109 }
3110 }
3111 { pagecomp }
3112 {
3113   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3114   {
3115     \__zrefclever_opt_tl_gset:cn
3116     {
3117       \__zrefclever_opt_varname_lang_default:enn
3118       { \l__zrefclever_setup_language_tl }
3119       { endrangefunc } { tl }
3120     }
3121     { __zrefclever_get_endrange_pagecomp }
3122     \__zrefclever_opt_tl_gclear:c
3123     {
3124       \__zrefclever_opt_varname_lang_default:enn
3125       { \l__zrefclever_setup_language_tl }
3126       { endrangeprop } { tl }
3127     }
3128   }
3129   {
3130     \__zrefclever_opt_tl_gset:cn
3131     {
3132       \__zrefclever_opt_varname_lang_type:eenn
3133       { \l__zrefclever_setup_language_tl }
3134       { \l__zrefclever_setup_type_tl }
3135       { endrangefunc } { tl }
3136     }
3137     { __zrefclever_get_endrange_pagecomp }
3138     \__zrefclever_opt_tl_gclear:c
3139     {
3140       \__zrefclever_opt_varname_lang_type:eenn
3141       { \l__zrefclever_setup_language_tl }
3142       { \l__zrefclever_setup_type_tl }
3143       { endrangeprop } { tl }
3144     }
3145   }
3146 }
3147 { pagecomp2 }
3148 {
3149   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3150   {
3151     \__zrefclever_opt_tl_gset:cn
3152     {
3153       \__zrefclever_opt_varname_lang_default:enn
3154       { \l__zrefclever_setup_language_tl }
3155       { endrangefunc } { tl }
3156     }
3157     { __zrefclever_get_endrange_pagecomptwo }
3158     \__zrefclever_opt_tl_gclear:c
3159     {

```

```

3160         \_zrefclever_opt_varname_lang_default:enn
3161         { \l__zrefclever_setup_language_tl }
3162         { endrangeprop } { tl }
3163     }
3164 }
3165 {
3166     \_zrefclever_opt_tl_gset:cn
3167     {
3168         \_zrefclever_opt_varname_lang_type:eenn
3169         { \l__zrefclever_setup_language_tl }
3170         { \l__zrefclever_setup_type_tl }
3171         { endrangefunc } { tl }
3172     }
3173     { __zrefclever_get_endrange_pagecomptwo }
3174     \_zrefclever_opt_tl_gclear:c
3175     {
3176         \_zrefclever_opt_varname_lang_type:eenn
3177         { \l__zrefclever_setup_language_tl }
3178         { \l__zrefclever_setup_type_tl }
3179         { endrangeprop } { tl }
3180     }
3181 }
3182 }
3183 }
3184 {
3185     \tl_if_empty:nTF {#1}
3186     {
3187         \msg_warning:nnn { zref-clever }
3188         { endrange-property-undefined } {#1}
3189     }
3190     {
3191         \zref@ifpropundefined {#1}
3192         {
3193             \msg_warning:nnn { zref-clever }
3194             { endrange-property-undefined } {#1}
3195         }
3196         {
3197             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3198             {
3199                 \_zrefclever_opt_tl_gset:cn
3200                 {
3201                     \_zrefclever_opt_varname_lang_default:enn
3202                     { \l__zrefclever_setup_language_tl }
3203                     { endrangefunc } { tl }
3204                 }
3205                 { __zrefclever_get_endrange_property }
3206                 \_zrefclever_opt_tl_gset:cn
3207                 {
3208                     \_zrefclever_opt_varname_lang_default:enn
3209                     { \l__zrefclever_setup_language_tl }
3210                     { endrangeprop } { tl }
3211                 }
3212                 {#1}
3213             }

```



```

3214         {
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                 { endrangefunc } { tl }
3221             }
3222             { __zrefclever_get_endrange_property }
3223         \__zrefclever_opt_tl_gset:cn
3224         {
3225             \__zrefclever_opt_varname_lang_type:eenn
3226             { \l__zrefclever_setup_language_tl }
3227             { \l__zrefclever_setup_type_tl }
3228             { endrangeprop } { tl }
3229         }
3230         {#1}
3231     }
3232 }
3233 }
3234 }
3235 } ,
3236 }
3237 \keys_define:nn { zref-clever/langsetup }
3238 {
3239     refpre .code:n =
3240     {
3241         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3242         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3243         { refpre } { refbounds }
3244     } ,
3245     refpos .code:n =
3246     {
3247         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3248         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3249         { refpos } { refbounds }
3250     } ,
3251     preref .code:n =
3252     {
3253         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3254         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3255         { preref } { refbounds }
3256     } ,
3257     postref .code:n =
3258     {
3259         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3260         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3261         { postref } { refbounds }
3262     } ,
3263 }
3264 \seq_map_inline:Nn
3265 \g__zrefclever_rf_opts_tl_type_names_seq
3266 {
3267     \keys_define:nn { zref-clever/langsetup }

```

```

3268 {
3269   #1 .value_required:n = true ,
3270   #1 .code:n =
3271   {
3272     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3273     {
3274       \msg_warning:nnn { zref-clever }
3275       { option-only-type-specific } {#1}
3276     }
3277     {
3278       \tl_if_empty:NTF \l__zrefclever_lang_variant_tl
3279       {
3280         \__zrefclever_opt_tl_gset:cn
3281         {
3282           \__zrefclever_opt_varname_lang_type:een
3283           { \l__zrefclever_setup_language_tl }
3284           { \l__zrefclever_setup_type_tl }
3285           {#1} { tl }
3286         }
3287         {##1}
3288       }
3289     }
3290     \__zrefclever_opt_tl_gset:cn
3291     {
3292       \__zrefclever_opt_varname_lang_type:een
3293       { \l__zrefclever_setup_language_tl }
3294       { \l__zrefclever_setup_type_tl }
3295       { \l__zrefclever_lang_variant_tl - #1 }
3296       { tl }
3297     }
3298     {##1}
3299   }
3300 } ,
3301 }
3302 }
3303 }
3304 \seq_map_inline:Nn
3305 \g__zrefclever_rf_opts_seq_refbounds_seq
3306 {
3307   \keys_define:nn { zref-clever/langsetup }
3308   {
3309     #1 .value_required:n = true ,
3310     #1 .code:n =
3311     {
3312       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3313       {
3314         \seq_gclear:N \g__zrefclever_tmpa_seq
3315         \__zrefclever_opt_seq_gset_clist_split:Nn
3316         \g__zrefclever_tmpa_seq {##1}
3317         \bool_lazy_or:nnTF
3318         { \tl_if_empty_p:n {##1} }
3319         {
3320           \int_compare_p:nNn
3321           { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }

```

```

3322     }
3323     {
3324     \__zrefclever_opt_seq_gset_eq:cN
3325     {
3326     \__zrefclever_opt_varname_lang_default:enn
3327     { \l__zrefclever_setup_language_tl }
3328     {#1} { seq }
3329     }
3330     \g__zrefclever_tmpa_seq
3331     }
3332     {
3333     \msg_warning:nnee { zref-clever }
3334     { refbounds-must-be-four }
3335     {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
3336     }
3337     }
3338     {
3339     \seq_gclear:N \g__zrefclever_tmpa_seq
3340     \__zrefclever_opt_seq_gset_clist_split:Nn
3341     \g__zrefclever_tmpa_seq {##1}
3342     \bool_lazy_or:nnTF
3343     { \tl_if_empty_p:n {##1} }
3344     {
3345     \int_compare_p:nNn
3346     { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3347     }
3348     {
3349     \__zrefclever_opt_seq_gset_eq:cN
3350     {
3351     \__zrefclever_opt_varname_lang_type:eenn
3352     { \l__zrefclever_setup_language_tl }
3353     { \l__zrefclever_setup_type_tl } {#1} { seq }
3354     }
3355     \g__zrefclever_tmpa_seq
3356     }
3357     {
3358     \msg_warning:nnee { zref-clever }
3359     { refbounds-must-be-four }
3360     {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
3361     }
3362     }
3363     } ,
3364     }
3365     }
3366     \seq_map_inline:Nn
3367     \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3368     {
3369     \keys_define:nn { zref-clever/langsetup }
3370     {
3371     #1 .choice: ,
3372     #1 / true .code:n =
3373     {
3374     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3375     {

```

```

3376         \_zrefclever_opt_bool_gset_true:c
3377         {
3378             \_zrefclever_opt_varname_lang_default:enn
3379             { \l_zrefclever_setup_language_tl }
3380             {#1} { bool }
3381         }
3382     }
3383     {
3384         \_zrefclever_opt_bool_gset_true:c
3385         {
3386             \_zrefclever_opt_varname_lang_type:eenn
3387             { \l_zrefclever_setup_language_tl }
3388             { \l_zrefclever_setup_type_tl }
3389             {#1} { bool }
3390         }
3391     }
3392 },
3393 #1 / false .code:n =
3394 {
3395     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3396     {
3397         \_zrefclever_opt_bool_gset_false:c
3398         {
3399             \_zrefclever_opt_varname_lang_default:enn
3400             { \l_zrefclever_setup_language_tl }
3401             {#1} { bool }
3402         }
3403     }
3404     {
3405         \_zrefclever_opt_bool_gset_false:c
3406         {
3407             \_zrefclever_opt_varname_lang_type:eenn
3408             { \l_zrefclever_setup_language_tl }
3409             { \l_zrefclever_setup_type_tl }
3410             {#1} { bool }
3411         }
3412     }
3413 },
3414 #1 .default:n = true ,
3415 no #1 .meta:n = { #1 = false } ,
3416 no #1 .value_forbidden:n = true ,
3417 }
3418 }

```

6 User interface

6.1 \zcref

\zcref The main user command of the package.

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

```

3419 \NewDocumentCommand \zcref { s O { } m }
3420 { \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}
```

```
3421 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
```

```
3422 {
```

```
3423   \group_begin:
```

Set options.

```
3424     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```
3425     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
```

```
3426     \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.

```
3427     \__zrefclever_provide_langfile:e { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3428     \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3429     \bool_lazy_and:nnT
```

```
      { \l__zrefclever_zrefcheck_available_bool }
```

```
3431     { \l__zrefclever_zcref_with_check_bool }
```

```
3432     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3433     \bool_lazy_or:nnT
```

```
3434     { \l__zrefclever_typeset_sort_bool }
```

```
3435     { \l__zrefclever_typeset_range_bool }
```

```
3436     { \__zrefclever_sort_labels: }
```

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

```
3437     \group_begin:
```

```
3438     \l__zrefclever_ref_typeset_font_tl
```

```
3439     \__zrefclever_typeset_refs:
```

```
3440     \group_end:
```

Typeset note.

```
3441     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
```

```
3442     {
```

```
3443         \__zrefclever_get_rf_opt_tl:neeN { notesep }
```

```
3444         { \l__zrefclever_label_type_a_tl }
```

```
3445         { \l__zrefclever_ref_language_tl }
```

```
3446         \l__zrefclever_tmpa_tl
```

```
3447         \l__zrefclever_tmpa_tl
```

```
3448         \l__zrefclever_zcref_note_tl
```

```
3449     }
```

Integration with zref-check.

```
3450     \bool_lazy_and:nnT
3451     { \l__zrefclever_zrefcheck_available_bool }
3452     { \l__zrefclever_zcref_with_check_bool }
3453     {
3454       \zrefcheck_zcref_end_label_maybe:
3455       \zrefcheck_zcref_run_checks_on_labels:n
3456       { \l__zrefclever_zcref_labels_seq }
3457     }
```

Integration with mathtools.

```
3458     \bool_if:NT \l__zrefclever_mathtools_loaded_bool
3459     {
3460       \__zrefclever_mathtools_showonlyrefs:n
3461       { \l__zrefclever_zcref_labels_seq }
3462     }
3463   \group_end:
3464 }
```

(End of definition for __zrefclever_zcref:nnnn.)

```
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
```

```
3465 \seq_new:N \l__zrefclever_zcref_labels_seq
3466 \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}
```

```
3467 \NewDocumentCommand \zcpageref { s O { } m }
3468 {
3469   \group_begin:
3470   \IfBooleanT {#1}
3471     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3472   \zcref [#2, ref = page] {#3}
3473   \group_end:
3474 }
```

(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.

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

`\l_zrefclever_label_type_b_tl`

`\l_zrefclever_label_enclval_a_tl` 3475 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 3476 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 3477 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 3478 `\tl_new:N \l__zrefclever_label_enclval_b_tl`

3479 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`

3480 `\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.

3481 `\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 sort priority of the “current” and “next” labels.

`\l_zrefclever_sort_prior_b_int`

3482 `\int_new:N \l__zrefclever_sort_prior_a_int`

3483 `\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`.

3484 `\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.

3485 `\cs_new_protected:Npn __zrefclever_sort_labels:`

3486 `{`

Store label types sequence.

3487 `\seq_clear:N \l__zrefclever_label_types_seq`

3488 `\tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }`

3489 `{`

3490 `\seq_map_function:NN \l__zrefclever_zcref_labels_seq`

3491 `__zrefclever_label_type_put_new_right:n`

3492 `}`

Sort.

```

3493 \seq_sort:Nn \l__zrefclever_zceref_labels_seq
3494 {
3495   \zref@ifrefundefined {##1}
3496   {
3497     \zref@ifrefundefined {##2}
3498     {
3499       % Neither label is defined.
3500       \sort_return_same:
3501     }
3502     {
3503       % The second label is defined, but the first isn't, leave the
3504       % undefined first (to be more visible).
3505       \sort_return_same:
3506     }
3507   }
3508   {
3509     \zref@ifrefundefined {##2}
3510     {
3511       % The first label is defined, but the second isn't, bring the
3512       % second forward.
3513       \sort_return_swapped:
3514     }
3515     {
3516       % The interesting case: both labels are defined. References
3517       % to the "default" property or to the "page" are quite
3518       % different with regard to sorting, so we branch them here to
3519       % specialized functions.
3520       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3521       { \__zrefclever_sort_page:n {##1} {##2} }
3522       { \__zrefclever_sort_default:n {##1} {##2} }
3523     }
3524   }
3525 }
3526 }

```

(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_zceref_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)}

3527 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3528 {
3529   \__zrefclever_extract_default:Nnnn
3530   \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3531   \seq_if_in:NVF \l__zrefclever_label_types_seq

```



```

3532 \l__zrefclever_label_type_a_tl
3533 {
3534 \seq_put_right:NV \l__zrefclever_label_types_seq
3535 \l__zrefclever_label_type_a_tl
3536 }
3537 }

```

(End of definition for __zrefclever_label_type_put_new_right:n.)

__zrefclever_sort_default:nn The heavy-lifting function for sorting of defined labels for “default” references (that is, a standard reference, not to “page”). This function is expected to be called within the sorting loop of __zrefclever_sort_labels: and receives the pair of labels being considered for a change of order or not. It should *always* “return” either \sort_return_same: or \sort_return_swapped:.

```

\__zrefclever_sort_default:nn {<label a>} {<label b>}
3538 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3539 {
3540 \__zrefclever_extract_default:Nnnn
3541 \l__zrefclever_label_type_a_tl {#1} { zc@type } { zc@missingtype }
3542 \__zrefclever_extract_default:Nnnn
3543 \l__zrefclever_label_type_b_tl {#2} { zc@type } { zc@missingtype }
3544 \tl_if_eq:NNTF
3545 \l__zrefclever_label_type_a_tl
3546 \l__zrefclever_label_type_b_tl
3547 { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3548 { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3549 }

```

(End of definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
3550 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3551 {
3552 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3553 {#1} { zc@enclval } { }
3554 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3555 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3556 {#2} { zc@enclval } { }
3557 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3558 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3559 {#1} { externaldocument } { }
3560 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3561 {#2} { externaldocument } { }
3562 \bool_set_false:N \l__zrefclever_sort_decided_bool
3563 % First we check if there's any "external document" difference (coming
3564 % from `zref-xr') and, if so, sort based on that.
3565 \tl_if_eq:NNTF
3566 \l__zrefclever_label_extdoc_a_tl
3567 \l__zrefclever_label_extdoc_b_tl
3568 {
3569 \bool_if:nTF
3570 {
3571 \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&

```

```

3572         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3573     }
3574     {
3575         \bool_set_true:N \l__zrefclever_sort_decided_bool
3576         \sort_return_same:
3577     }
3578     {
3579         \bool_if:nTF
3580         {
3581             ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3582             \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3583         }
3584         {
3585             \bool_set_true:N \l__zrefclever_sort_decided_bool
3586             \sort_return_swapped:
3587         }
3588         {
3589             \bool_set_true:N \l__zrefclever_sort_decided_bool
3590             % Two different "external documents": last resort, sort by the
3591             % document name itself.
3592             \str_compare:eNeTF
3593             { \l__zrefclever_label_extdoc_b_tl } <
3594             { \l__zrefclever_label_extdoc_a_tl }
3595             { \sort_return_swapped: }
3596             { \sort_return_same: }
3597         }
3598     }
3599 }
3600 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3601 {
3602     \bool_if:nTF
3603     {
3604         % Both are empty: neither label has any (further) "enclosing
3605         % counters" (left).
3606         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3607         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3608     }
3609     {
3610         \bool_set_true:N \l__zrefclever_sort_decided_bool
3611         \int_compare:nNnTF
3612         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3613         >
3614         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3615         { \sort_return_swapped: }
3616         { \sort_return_same: }
3617     }
3618     {
3619         \bool_if:nTF
3620         {
3621             % `a' is empty (and `b' is not): `b' may be nested in `a'.
3622             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3623         }
3624         {
3625             \bool_set_true:N \l__zrefclever_sort_decided_bool

```

```

3626 \int_compare:nNnTF
3627 { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3628 >
3629 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3630 { \sort_return_swapped: }
3631 { \sort_return_same: }
3632 }
3633 {
3634 \bool_if:nTF
3635 {
3636 % `b' is empty (and `a' is not): `a' may be nested in `b'.
3637 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3638 }
3639 {
3640 \bool_set_true:N \l__zrefclever_sort_decided_bool
3641 \int_compare:nNnTF
3642 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3643 <
3644 { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3645 { \sort_return_same: }
3646 { \sort_return_swapped: }
3647 }
3648 {
3649 % Neither is empty: we can compare the values of the
3650 % current enclosing counter in the loop, if they are
3651 % equal, we are still in the loop, if they are not, a
3652 % sorting decision can be made directly.
3653 \int_compare:nNnTF
3654 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3655 =
3656 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3657 {
3658 \tl_set:Ne \l__zrefclever_label_enclval_a_tl
3659 { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3660 \tl_set:Ne \l__zrefclever_label_enclval_b_tl
3661 { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3662 }
3663 {
3664 \bool_set_true:N \l__zrefclever_sort_decided_bool
3665 \int_compare:nNnTF
3666 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3667 >
3668 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3669 { \sort_return_swapped: }
3670 { \sort_return_same: }
3671 }
3672 }
3673 }
3674 }
3675 }
3676 }

```

(End of definition for `__zrefclever_sort_default_same_type:mn`.)

_zrefclever_sort_default_different_types:nn

```
\_zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
```

```
3677 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
3678 {
```

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”.

```
3679 \int_zero:N \l__zrefclever_sort_prior_a_int
3680 \int_zero:N \l__zrefclever_sort_prior_b_int
3681 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3682 {
3683   \tl_if_eq:nnTF {##2} {{othertypes}}
3684   {
3685     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3686     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3687     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3688     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3689   }
3690   {
3691     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3692     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3693     {
3694       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3695       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3696     }
3697   }
3698 }
```

Then do the actual sorting.

```
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_same: }
3706 {
3707   \bool_if:nTF
3708   {
3709     \int_compare_p:nNn
3710     { \l__zrefclever_sort_prior_a_int } >
3711     { \l__zrefclever_sort_prior_b_int }
3712   }
3713   { \sort_return_swapped: }
3714   {
3715     % Sort priorities are equal: the type that occurs first in
3716     % `labels', as given by the user, is kept (or brought) forward.
3717     \seq_map_inline:Nn \l__zrefclever_label_types_seq
3718     {
3719       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3720       { \seq_map_break:n { \sort_return_same: } }
3721       {
3722         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3723         { \seq_map_break:n { \sort_return_swapped: } }

```

```

3724         }
3725     }
3726 }
3727 }
3728 }

```

(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)}

```

```

3729 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3730 {
3731   \int_compare:nNnTF
3732     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3733     >
3734     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3735     { \sort_return_swapped: }
3736     { \sort_return_same:   }
3737 }

```

(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` 3738 `\seq_new:N \l__zrefclever_typeset_labels_seq`
`\l__zrefclever_last_of_type_bool`

3739 \bool_new:N \l__zrefclever_typeset_last_bool
 3740 \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.)

Auxiliary variables for __zrefclever_typeset_refs: main counters.

\l_zrefclever_type_count_int
 \l_zrefclever_label_count_int
 \l__zrefclever_ref_count_int

3741 \int_new:N \l__zrefclever_type_count_int
 3742 \int_new:N \l__zrefclever_label_count_int
 3743 \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.)

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

\l__zrefclever_label_a_tl
 \l__zrefclever_label_b_tl
 \l_zrefclever_typeset_queue_prev_tl
 \l_zrefclever_typeset_queue_curr_tl
 \l_zrefclever_type_first_label_tl
 \l_zrefclever_type_first_label_type_tl

3744 \tl_new:N \l__zrefclever_label_a_tl
 3745 \tl_new:N \l__zrefclever_label_b_tl
 3746 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
 3747 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
 3748 \tl_new:N \l_zrefclever_type_first_label_tl
 3749 \tl_new:N \l_zrefclever_type_first_label_type_tl

(End of definition for \l__zrefclever_label_a_tl and others.)

Auxiliary variables for __zrefclever_typeset_refs: type name handling.

\l__zrefclever_type_name_tl
 \l_zrefclever_name_in_link_bool
 \l_zrefclever_type_name_missing_bool
 \l_zrefclever_name_format_tl
 \l_zrefclever_name_format_fallback_tl
 \l_zrefclever_type_name_gender_seq

3750 \tl_new:N \l__zrefclever_type_name_tl
 3751 \bool_new:N \l_zrefclever_name_in_link_bool
 3752 \bool_new:N \l_zrefclever_type_name_missing_bool
 3753 \tl_new:N \l_zrefclever_name_format_tl
 3754 \tl_new:N \l_zrefclever_name_format_fallback_tl
 3755 \seq_new:N \l_zrefclever_type_name_gender_seq

(End of definition for \l__zrefclever_type_name_tl and others.)

Auxiliary variables for __zrefclever_typeset_refs: range handling.

\l_zrefclever_range_count_int
 \l_zrefclever_range_same_count_int
 \l_zrefclever_range_beg_label_tl
 \l_zrefclever_range_beg_is_first_bool
 \l_zrefclever_range_end_ref_tl
 \l_zrefclever_next_maybe_range_bool
 \l_zrefclever_next_is_same_bool

3756 \int_new:N \l__zrefclever_range_count_int
 3757 \int_new:N \l__zrefclever_range_same_count_int
 3758 \tl_new:N \l__zrefclever_range_beg_label_tl
 3759 \bool_new:N \l__zrefclever_range_beg_is_first_bool
 3760 \tl_new:N \l_zrefclever_range_end_ref_tl
 3761 \bool_new:N \l_zrefclever_next_maybe_range_bool
 3762 \bool_new:N \l_zrefclever_next_is_same_bool

(End of definition for \l__zrefclever_range_count_int and others.)

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

\l__zrefclever_tpairsep_tl
 \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_rangeseptl
 \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

3763 \tl_new:N \l__zrefclever_tpairsep_tl
 3764 \tl_new:N \l__zrefclever_tlistsep_tl
 3765 \tl_new:N \l__zrefclever_tlastsep_tl
 3766 \tl_new:N \l__zrefclever_namesep_tl
 3767 \tl_new:N \l_zrefclever_pairsep_tl
 3768 \tl_new:N \l__zrefclever_listsep_tl
 3769 \tl_new:N \l__zrefclever_lastsep_tl
 3770 \tl_new:N \l__zrefclever_rangeseptl

```

3771 \tl_new:N \l__zrefclever_namefont_tl
3772 \tl_new:N \l__zrefclever_reffont_tl
3773 \tl_new:N \l__zrefclever_endrangefunc_tl
3774 \tl_new:N \l__zrefclever_endrangeprop_tl
3775 \bool_new:N \l__zrefclever_cap_bool
3776 \bool_new:N \l__zrefclever_abbrev_bool
3777 \bool_new:N \l__zrefclever_rangetopair_bool

```

(End of definition for \l__zrefclever_tpairsep_tl and others.)

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_rb_seq
  \l__zrefclever_refbounds_mid_re_seq
  \l__zrefclever_refbounds_last_seq
  \l__zrefclever_refbounds_last_pe_seq
  \l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_seq
\l__zrefclever_type_first_refbounds_set_bool
3778 \seq_new:N \l__zrefclever_refbounds_first_seq
3779 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3780 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3781 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3782 \seq_new:N \l__zrefclever_refbounds_mid_seq
3783 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3784 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3785 \seq_new:N \l__zrefclever_refbounds_last_seq
3786 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3787 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3788 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3789 \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.

```

3790 \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.

```

3791 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3792   {
3793     \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3794     \l__zrefclever_zcref_labels_seq
3795     \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3796     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3797     \tl_clear:N \l__zrefclever_type_first_label_tl
3798     \tl_clear:N \l__zrefclever_type_first_label_type_tl
3799     \tl_clear:N \l__zrefclever_range_beg_label_tl
3800     \tl_clear:N \l__zrefclever_range_end_ref_tl
3801     \int_zero:N \l__zrefclever_label_count_int
3802     \int_zero:N \l__zrefclever_type_count_int
3803     \int_zero:N \l__zrefclever_ref_count_int
3804     \int_zero:N \l__zrefclever_range_count_int
3805     \int_zero:N \l__zrefclever_range_same_count_int
3806     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3807     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool

```



```

3808 % Get type block options (not type-specific).
3809 \__zrefclever_get_rf_opt_tl:neeN { tpairsep }
3810   { \l__zrefclever_label_type_a_tl }
3811   { \l__zrefclever_ref_language_tl }
3812   \l__zrefclever_tpairsep_tl
3813 \__zrefclever_get_rf_opt_tl:neeN { tlistsep }
3814   { \l__zrefclever_label_type_a_tl }
3815   { \l__zrefclever_ref_language_tl }
3816   \l__zrefclever_tlistsep_tl
3817 \__zrefclever_get_rf_opt_tl:neeN { tlastsep }
3818   { \l__zrefclever_label_type_a_tl }
3819   { \l__zrefclever_ref_language_tl }
3820   \l__zrefclever_tlastsep_tl
3821 % Process label stack.
3822 \bool_set_false:N \l__zrefclever_typeset_last_bool
3823 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3824   {
3825     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3826     \l__zrefclever_label_a_tl
3827     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3828       {
3829         \tl_clear:N \l__zrefclever_label_b_tl
3830         \bool_set_true:N \l__zrefclever_typeset_last_bool
3831       }
3832       {
3833         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3834         \l__zrefclever_label_b_tl
3835       }
3836     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3837       {
3838         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3839         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3840       }
3841       {
3842         \__zrefclever_extract_default:NVnn
3843         \l__zrefclever_label_type_a_tl
3844         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3845         \__zrefclever_extract_default:NVnn
3846         \l__zrefclever_label_type_b_tl
3847         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3848       }
3849     % First, we establish whether the "current label" (i.e. `a') is the
3850     % last one of its type. This can happen because the "next label"
3851     % (i.e. `b') is of a different type (or different definition status),
3852     % or because we are at the end of the list.
3853     \bool_if:NTF \l__zrefclever_typeset_last_bool
3854       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3855       {
3856         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3857         {
3858           \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3859             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3860             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3861         }
3862       }

```

```

3862     {
3863         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3864         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3865         {
3866             % Neither is undefined, we must check the types.
3867             \tl_if_eq:NNTF
3868             \l__zrefclever_label_type_a_tl
3869             \l__zrefclever_label_type_b_tl
3870             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3871             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3872         }
3873     }
3874 }
3875 % Handle warnings in case of reference or type undefined.
3876 % Test: `zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3877 \zref@refused { \l__zrefclever_label_a_tl }
3878 % Test: `zc-typeset01.lvt': "Typeset refs: warn missing type"
3879 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3880 {}
3881 {
3882     \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3883     {
3884         \msg_warning:nne { zref-clever } { missing-type }
3885         { \l__zrefclever_label_a_tl }
3886     }
3887     \zref@ifrefcontainsprop
3888     { \l__zrefclever_label_a_tl }
3889     { \l__zrefclever_ref_property_tl }
3890     { }
3891     {
3892         \msg_warning:nnee { zref-clever } { missing-property }
3893         { \l__zrefclever_ref_property_tl }
3894         { \l__zrefclever_label_a_tl }
3895     }
3896 }
3897 % Get possibly type-specific separators, refbounds, font and other
3898 % options, once per type.
3899 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3900 {
3901     \__zrefclever_get_rf_opt_tl:neeN { namesep }
3902     { \l__zrefclever_label_type_a_tl }
3903     { \l__zrefclever_ref_language_tl }
3904     \l__zrefclever_namesep_tl
3905     \__zrefclever_get_rf_opt_tl:neeN { pairsep }
3906     { \l__zrefclever_label_type_a_tl }
3907     { \l__zrefclever_ref_language_tl }
3908     \l__zrefclever_pairsep_tl
3909     \__zrefclever_get_rf_opt_tl:neeN { listsep }
3910     { \l__zrefclever_label_type_a_tl }
3911     { \l__zrefclever_ref_language_tl }
3912     \l__zrefclever_listsep_tl
3913     \__zrefclever_get_rf_opt_tl:neeN { lastsep }
3914     { \l__zrefclever_label_type_a_tl }
3915     { \l__zrefclever_ref_language_tl }

```

```

3916 \l__zrefclever_lastsep_tl
3917 \__zrefclever_get_rf_opt_tl:neeN { rangesep }
3918 { \l__zrefclever_label_type_a_tl }
3919 { \l__zrefclever_ref_language_tl }
3920 \l__zrefclever_rangesep_tl
3921 \__zrefclever_get_rf_opt_tl:neeN { namefont }
3922 { \l__zrefclever_label_type_a_tl }
3923 { \l__zrefclever_ref_language_tl }
3924 \l__zrefclever_namefont_tl
3925 \__zrefclever_get_rf_opt_tl:neeN { reffont }
3926 { \l__zrefclever_label_type_a_tl }
3927 { \l__zrefclever_ref_language_tl }
3928 \l__zrefclever_reffont_tl
3929 \__zrefclever_get_rf_opt_tl:neeN { endrangefunc }
3930 { \l__zrefclever_label_type_a_tl }
3931 { \l__zrefclever_ref_language_tl }
3932 \l__zrefclever_endrangefunc_tl
3933 \__zrefclever_get_rf_opt_tl:neeN { endrangeprop }
3934 { \l__zrefclever_label_type_a_tl }
3935 { \l__zrefclever_ref_language_tl }
3936 \l__zrefclever_endrangeprop_tl
3937 \__zrefclever_get_rf_opt_bool:nneeN { cap } { false }
3938 { \l__zrefclever_label_type_a_tl }
3939 { \l__zrefclever_ref_language_tl }
3940 \l__zrefclever_cap_bool
3941 \__zrefclever_get_rf_opt_bool:nneeN { abbrev } { false }
3942 { \l__zrefclever_label_type_a_tl }
3943 { \l__zrefclever_ref_language_tl }
3944 \l__zrefclever_abbrev_bool
3945 \__zrefclever_get_rf_opt_bool:nneeN { rangetopair } { true }
3946 { \l__zrefclever_label_type_a_tl }
3947 { \l__zrefclever_ref_language_tl }
3948 \l__zrefclever_rangetopair_bool
3949 \__zrefclever_get_rf_opt_seq:neeN { rebounds-first }
3950 { \l__zrefclever_label_type_a_tl }
3951 { \l__zrefclever_ref_language_tl }
3952 \l__zrefclever_rebounds_first_seq
3953 \__zrefclever_get_rf_opt_seq:neeN { rebounds-first-sg }
3954 { \l__zrefclever_label_type_a_tl }
3955 { \l__zrefclever_ref_language_tl }
3956 \l__zrefclever_rebounds_first_sg_seq
3957 \__zrefclever_get_rf_opt_seq:neeN { rebounds-first-pb }
3958 { \l__zrefclever_label_type_a_tl }
3959 { \l__zrefclever_ref_language_tl }
3960 \l__zrefclever_rebounds_first_pb_seq
3961 \__zrefclever_get_rf_opt_seq:neeN { rebounds-first-rb }
3962 { \l__zrefclever_label_type_a_tl }
3963 { \l__zrefclever_ref_language_tl }
3964 \l__zrefclever_rebounds_first_rb_seq
3965 \__zrefclever_get_rf_opt_seq:neeN { rebounds-mid }
3966 { \l__zrefclever_label_type_a_tl }
3967 { \l__zrefclever_ref_language_tl }
3968 \l__zrefclever_rebounds_mid_seq
3969 \__zrefclever_get_rf_opt_seq:neeN { rebounds-mid-rb }

```

```

3970         { \l__zrefclever_label_type_a_tl }
3971         { \l__zrefclever_ref_language_tl }
3972         \l__zrefclever_refbounds_mid_rb_seq
3973     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-re }
3974         { \l__zrefclever_label_type_a_tl }
3975         { \l__zrefclever_ref_language_tl }
3976         \l__zrefclever_refbounds_mid_re_seq
3977     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last }
3978         { \l__zrefclever_label_type_a_tl }
3979         { \l__zrefclever_ref_language_tl }
3980         \l__zrefclever_refbounds_last_seq
3981     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-pe }
3982         { \l__zrefclever_label_type_a_tl }
3983         { \l__zrefclever_ref_language_tl }
3984         \l__zrefclever_refbounds_last_pe_seq
3985     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-re }
3986         { \l__zrefclever_label_type_a_tl }
3987         { \l__zrefclever_ref_language_tl }
3988         \l__zrefclever_refbounds_last_re_seq
3989     }
3990     % Here we send this to a couple of auxiliary functions.
3991     \bool_if:NTF \l__zrefclever_last_of_type_bool
3992     % There exists no next label of the same type as the current.
3993     { \__zrefclever_typeset_refs_last_of_type: }
3994     % There exists a next label of the same type as the current.
3995     { \__zrefclever_typeset_refs_not_last_of_type: }
3996 }
3997 }

```

(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.

```

3998 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3999 {
4000     % Process the current label to the current queue.
4001     \int_case:nnF { \l__zrefclever_label_count_int }
4002     {
4003         % It is the last label of its type, but also the first one, and that's
4004         % what matters here: just store it.
4005         % Test: `zc-typeset01.lvt': "Last of type: single"
4006         { 0 }
4007         {
4008             \tl_set:NV \l__zrefclever_type_first_label_tl
4009             \l__zrefclever_label_a_tl
4010             \tl_set:NV \l__zrefclever_type_first_label_type_tl

```

```

4011         \l__zrefclever_label_type_a_tl
4012     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4013         \l__zrefclever_refbounds_first_sg_seq
4014     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4015 }
4016 % The last is the second: we have a pair (if not repeated).
4017 % Test: `zc-typeset01.lvt': "Last of type: pair"
4018 { 1 }
4019 {
4020     \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4021     {
4022         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4023             \l__zrefclever_refbounds_first_sg_seq
4024         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4025     }
4026     {
4027         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4028             {
4029                 \exp_not:V \l__zrefclever_pairsep_tl
4030                 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4031                 \l__zrefclever_refbounds_last_pe_seq
4032             }
4033         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4034             \l__zrefclever_refbounds_first_pb_seq
4035         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4036     }
4037 }
4038 }
4039 % Last is third or more of its type: without repetition, we'd have the
4040 % last element on a list, but control for possible repetition.
4041 {
4042     \int_case:nnF { \l__zrefclever_range_count_int }
4043     {
4044         % There was no range going on.
4045         % Test: `zc-typeset01.lvt': "Last of type: not range"
4046         { 0 }
4047         {
4048             \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4049             {
4050                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4051                     {
4052                         \exp_not:V \l__zrefclever_pairsep_tl
4053                         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4054                         \l__zrefclever_refbounds_last_pe_seq
4055                     }
4056             }
4057             {
4058                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4059                     {
4060                         \exp_not:V \l__zrefclever_lastsep_tl
4061                         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4062                         \l__zrefclever_refbounds_last_seq
4063                     }
4064             }
4065         }
4066     }

```

```

4065     }
4066     % Last in the range is also the second in it.
4067     % Test: `zc-typeset01.lvt': "Last of type: pair in sequence"
4068     { 1 }
4069     {
4070     \int_compare:nNnTF
4071     { \l__zrefclever_range_same_count_int } = { 1 }
4072     {
4073     % We know `range_beg_is_first_bool' is false, since this is
4074     % the second element in the range, but the third or more in
4075     % the type list.
4076     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4077     {
4078     \exp_not:V \l__zrefclever_pairsep_tl
4079     \__zrefclever_get_ref:VN
4080     \l__zrefclever_range_beg_label_tl
4081     \l__zrefclever_refbounds_last_pe_seq
4082     }
4083     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4084     \l__zrefclever_refbounds_first_pb_seq
4085     \bool_set_true:N
4086     \l__zrefclever_type_first_refbounds_set_bool
4087     }
4088     {
4089     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4090     {
4091     \exp_not:V \l__zrefclever_listsep_tl
4092     \__zrefclever_get_ref:VN
4093     \l__zrefclever_range_beg_label_tl
4094     \l__zrefclever_refbounds_mid_seq
4095     \exp_not:V \l__zrefclever_lastsep_tl
4096     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4097     \l__zrefclever_refbounds_last_seq
4098     }
4099     }
4100     }
4101     }
4102     % Last in the range is third or more in it.
4103     {
4104     \int_case:nnF
4105     {
4106     \l__zrefclever_range_count_int -
4107     \l__zrefclever_range_same_count_int
4108     }
4109     {
4110     % Repetition, not a range.
4111     % Test: `zc-typeset01.lvt': "Last of type: range to one"
4112     { 0 }
4113     {
4114     % If `range_beg_is_first_bool' is true, it means it was also
4115     % the first of the type, and hence its typesetting was
4116     % already handled, and we just have to set refbounds.
4117     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4118     {

```

```

4119         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4120         \l__zrefclever_refbounds_first_sg_seq
4121     \bool_set_true:N
4122         \l__zrefclever_type_first_refbounds_set_bool
4123     }
4124     {
4125         \int_compare:nNnTF
4126         { \l__zrefclever_ref_count_int } < { 2 }
4127         {
4128             \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4129             {
4130                 \exp_not:V \l__zrefclever_pairsep_tl
4131                 \__zrefclever_get_ref:VN
4132                 \l__zrefclever_range_beg_label_tl
4133                 \l__zrefclever_refbounds_last_pe_seq
4134             }
4135         }
4136         {
4137             \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4138             {
4139                 \exp_not:V \l__zrefclever_lastsep_tl
4140                 \__zrefclever_get_ref:VN
4141                 \l__zrefclever_range_beg_label_tl
4142                 \l__zrefclever_refbounds_last_seq
4143             }
4144         }
4145     }
4146 }
4147 % A `range', but with no skipped value, treat as pair if range
4148 % started with first of type, otherwise as list.
4149 % Test: `zc-typeset01.lvt': "Last of type: range to pair"
4150 { 1 }
4151 {
4152     % Ditto.
4153     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4154     {
4155         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4156         \l__zrefclever_refbounds_first_pb_seq
4157         \bool_set_true:N
4158             \l__zrefclever_type_first_refbounds_set_bool
4159         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4160         {
4161             \exp_not:V \l__zrefclever_pairsep_tl
4162             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4163             \l__zrefclever_refbounds_last_pe_seq
4164         }
4165     }
4166     {
4167         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4168         {
4169             \exp_not:V \l__zrefclever_listsep_tl
4170             \__zrefclever_get_ref:VN
4171             \l__zrefclever_range_beg_label_tl
4172             \l__zrefclever_refbounds_mid_seq

```

```

4173     }
4174     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4175     {
4176         \exp_not:V \l__zrefclever_lastsep_tl
4177         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4178         \l__zrefclever_refbounds_last_seq
4179     }
4180 }
4181 }
4182 }
4183 {
4184     % An actual range.
4185     % Test: `zc-typeset01.lvt': "Last of type: range"
4186     % Ditto.
4187     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4188     {
4189         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4190         \l__zrefclever_refbounds_first_rb_seq
4191         \bool_set_true:N
4192         \l__zrefclever_type_first_refbounds_set_bool
4193     }
4194     {
4195         \int_compare:nNnTF
4196         { \l__zrefclever_ref_count_int } < { 2 }
4197         {
4198             \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4199             {
4200                 \exp_not:V \l__zrefclever_pairsep_tl
4201                 \__zrefclever_get_ref:VN
4202                 \l__zrefclever_range_beg_label_tl
4203                 \l__zrefclever_refbounds_mid_rb_seq
4204             }
4205             \seq_set_eq:NN
4206             \l__zrefclever_type_first_refbounds_seq
4207             \l__zrefclever_refbounds_first_pb_seq
4208             \bool_set_true:N
4209             \l__zrefclever_type_first_refbounds_set_bool
4210         }
4211         {
4212             \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4213             {
4214                 \exp_not:V \l__zrefclever_lastsep_tl
4215                 \__zrefclever_get_ref:VN
4216                 \l__zrefclever_range_beg_label_tl
4217                 \l__zrefclever_refbounds_mid_rb_seq
4218             }
4219         }
4220     }
4221     \bool_lazy_and:nnTF
4222     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4223     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4224     {
4225         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4226         \l__zrefclever_range_beg_label_tl

```



```

4227         \l__zrefclever_label_a_tl
4228         \l__zrefclever_range_end_ref_tl
4229     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4230     {
4231         \exp_not:V \l__zrefclever_rangesep_tl
4232         \__zrefclever_get_ref_endrange:VVN
4233         \l__zrefclever_label_a_tl
4234         \l__zrefclever_range_end_ref_tl
4235         \l__zrefclever_refbounds_last_re_seq
4236     }
4237 }
4238 {
4239     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4240     {
4241         \exp_not:V \l__zrefclever_rangesep_tl
4242         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4243         \l__zrefclever_refbounds_last_re_seq
4244     }
4245 }
4246 }
4247 }
4248 }
4249 % Handle "range" option. The idea is simple: if the queue is not empty,
4250 % we replace it with the end of the range (or pair). We can still
4251 % retrieve the end of the range from `label_a' since we know to be
4252 % processing the last label of its type at this point.
4253 \bool_if:NT \l__zrefclever_typeset_range_bool
4254 {
4255     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4256     {
4257         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4258         { }
4259         {
4260             \msg_warning:nne { zref-clever } { single-element-range }
4261             { \l__zrefclever_type_first_label_type_tl }
4262         }
4263     }
4264     {
4265         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4266         \bool_if:NT \l__zrefclever_rangetopair_bool
4267         {
4268             \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4269             { }
4270             {
4271                 \__zrefclever_labels_in_sequence:nn
4272                 { \l__zrefclever_type_first_label_tl }
4273                 { \l__zrefclever_label_a_tl }
4274             }
4275         }
4276         % Test: `zc-typeset01.lvt': "Last of type: option range"
4277         % Test: `zc-typeset01.lvt': "Last of type: option range to pair"
4278         \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4279         {
4280             \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl

```

```

4281         {
4282             \exp_not:V \l__zrefclever_pairsep_tl
4283             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4284             \l__zrefclever_refbounds_last_pe_seq
4285         }
4286     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4287     \l__zrefclever_refbounds_first_pb_seq
4288     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4289 }
4290 {
4291     \bool_lazy_and:nnTF
4292     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4293     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VWN } }
4294     {
4295         % We must get `type_first_label_tl' instead of
4296         % `range_beg_label_tl' here, since it is not necessary
4297         % that the first of type was actually starting a range for
4298         % the `range' option to be used.
4299         \use:c { \l__zrefclever_endrangefunc_tl :VWN }
4300         \l__zrefclever_type_first_label_tl
4301         \l__zrefclever_label_a_tl
4302         \l__zrefclever_range_end_ref_tl
4303         \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4304         {
4305             \exp_not:V \l__zrefclever_rangesep_tl
4306             \__zrefclever_get_ref_endrange:VWN
4307             \l__zrefclever_label_a_tl
4308             \l__zrefclever_range_end_ref_tl
4309             \l__zrefclever_refbounds_last_re_seq
4310         }
4311     }
4312     {
4313         \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4314         {
4315             \exp_not:V \l__zrefclever_rangesep_tl
4316             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4317             \l__zrefclever_refbounds_last_re_seq
4318         }
4319     }
4320     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4321     \l__zrefclever_refbounds_first_rb_seq
4322     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4323 }
4324 }
4325 }
4326 % If none of the special cases for the first of type refbounds have been
4327 % set, do it.
4328 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4329 {
4330     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4331     \l__zrefclever_refbounds_first_seq
4332 }
4333 % Now that the type block is finished, we can add the name and the first
4334 % ref to the queue. Also, if "typeset" option is not "both", handle it

```

```

4335 % here as well.
4336 \__zrefclever_type_name_setup:
4337 \bool_if:nTF
4338 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4339 {
4340   \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4341     { \__zrefclever_get_ref_first: }
4342 }
4343 {
4344   \bool_if:NTF \l__zrefclever_typeset_ref_bool
4345   {
4346     % Test: `zc-typeset01.lvt': "Last of type: option typeset ref"
4347     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4348       {
4349         \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4350         \l__zrefclever_type_first_refbounds_seq
4351       }
4352   }
4353   {
4354     \bool_if:NTF \l__zrefclever_typeset_name_bool
4355     {
4356       % Test: `zc-typeset01.lvt': "Last of type: option typeset name"
4357       \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4358         {
4359           \bool_if:NTF \l__zrefclever_name_in_link_bool
4360             {
4361               \exp_not:N \group_begin:
4362               \exp_not:V \l__zrefclever_namefont_tl
4363               \__zrefclever_hyperlink:nnn
4364               {
4365                 \__zrefclever_extract_url_unexp:V
4366                 \l__zrefclever_type_first_label_tl
4367               }
4368               {
4369                 \__zrefclever_extract_unexp:Vnn
4370                 \l__zrefclever_type_first_label_tl
4371                 { anchor } { }
4372               }
4373               { \exp_not:V \l__zrefclever_type_name_tl }
4374               \exp_not:N \group_end:
4375             }
4376             {
4377               \exp_not:N \group_begin:
4378               \exp_not:V \l__zrefclever_namefont_tl
4379               \exp_not:V \l__zrefclever_type_name_tl
4380               \exp_not:N \group_end:
4381             }
4382           }
4383         }
4384       {
4385         % Logically, this case would correspond to "typeset=none", but
4386         % it should not occur, given that the options are set up to
4387         % typeset either "ref" or "name". Still, leave here a
4388         % sensible fallback, equal to the behavior of "both".

```

```

4389             % Test: `zc-typeset01.lvt': "Last of type: option typeset none"
4390             \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4391             { \__zrefclever_get_ref_first: }
4392         }
4393     }
4394 }
4395 % Typeset the previous type block, if there is one.
4396 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4397 {
4398     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4399     { \l__zrefclever_tlistsep_tl }
4400     \l__zrefclever_typeset_queue_prev_tl
4401 }
4402 % Extra log for testing.
4403 \bool_if:NT \l__zrefclever_verbose_testing_bool
4404 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4405 % Wrap up loop, or prepare for next iteration.
4406 \bool_if:NTF \l__zrefclever_typeset_last_bool
4407 {
4408     % We are finishing, typeset the current queue.
4409     \int_case:nnF { \l__zrefclever_type_count_int }
4410     {
4411         % Single type.
4412         % Test: `zc-typeset01.lvt': "Last of type: single type"
4413         { 0 }
4414         { \l__zrefclever_typeset_queue_curr_tl }
4415         % Pair of types.
4416         % Test: `zc-typeset01.lvt': "Last of type: pair of types"
4417         { 1 }
4418         {
4419             \l__zrefclever_tpairsep_tl
4420             \l__zrefclever_typeset_queue_curr_tl
4421         }
4422     }
4423     {
4424         % Last in list of types.
4425         % Test: `zc-typeset01.lvt': "Last of type: list of types"
4426         \l__zrefclever_tlastsep_tl
4427         \l__zrefclever_typeset_queue_curr_tl
4428     }
4429     % And nudge in case of multitype reference.
4430     \bool_lazy_all:nT
4431     {
4432         { \l__zrefclever_nudge_enabled_bool }
4433         { \l__zrefclever_nudge_multitype_bool }
4434         { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4435     }
4436     { \msg_warning:nn { zref-clever } { nudge-multitype } }
4437 }
4438 {
4439     % There are further labels, set variables for next iteration.
4440     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4441     \l__zrefclever_typeset_queue_curr_tl
4442     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl

```

```

4443 \tl_clear:N \l__zrefclever_type_first_label_tl
4444 \tl_clear:N \l__zrefclever_type_first_label_type_tl
4445 \tl_clear:N \l__zrefclever_range_beg_label_tl
4446 \tl_clear:N \l__zrefclever_range_end_ref_tl
4447 \int_zero:N \l__zrefclever_label_count_int
4448 \int_zero:N \l__zrefclever_ref_count_int
4449 \int_incr:N \l__zrefclever_type_count_int
4450 \int_zero:N \l__zrefclever_range_count_int
4451 \int_zero:N \l__zrefclever_range_same_count_int
4452 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4453 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4454 }
4455 }

```

(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.

```

4456 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4457 {
4458   % Signal if next label may form a range with the current one (only
4459   % considered if compression is enabled in the first place).
4460   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4461   \bool_set_false:N \l__zrefclever_next_is_same_bool
4462   \bool_if:NT \l__zrefclever_typeset_compress_bool
4463   {
4464     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4465     { }
4466     {
4467       \__zrefclever_labels_in_sequence:nn
4468       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
4469     }
4470   }
4471   % Process the current label to the current queue.
4472   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4473   {
4474     % Current label is the first of its type (also not the last, but it
4475     % doesn't matter here): just store the label.
4476     \tl_set:NV \l__zrefclever_type_first_label_tl
4477     \l__zrefclever_label_a_tl
4478     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4479     \l__zrefclever_label_type_a_tl
4480     \int_incr:N \l__zrefclever_ref_count_int
4481     % If the next label may be part of a range, signal it (we deal with it
4482     % as the "first", and must do it there, to handle hyperlinking), but
4483     % also step the range counters.
4484     % Test: `zc-typeset01.lvt': "Not last of type: first is range"
4485     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4486     {
4487       \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4488       \tl_set:NV \l__zrefclever_range_beg_label_tl
4489       \l__zrefclever_label_a_tl
4490       \tl_clear:N \l__zrefclever_range_end_ref_tl
4491       \int_incr:N \l__zrefclever_range_count_int
4492       \bool_if:NT \l__zrefclever_next_is_same_bool

```

```

4493         { \int_incr:N \l__zrefclever_range_same_count_int }
4494     }
4495 }
4496 {
4497 % Current label is neither the first (nor the last) of its type.
4498 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4499 {
4500 % Starting, or continuing a range.
4501 \int_compare:nNnTF
4502 { \l__zrefclever_range_count_int } = { 0 }
4503 {
4504 % There was no range going, we are starting one.
4505 \tl_set:NV \l__zrefclever_range_beg_label_tl
4506 \l__zrefclever_label_a_tl
4507 \tl_clear:N \l__zrefclever_range_end_ref_tl
4508 \int_incr:N \l__zrefclever_range_count_int
4509 \bool_if:NT \l__zrefclever_next_is_same_bool
4510 { \int_incr:N \l__zrefclever_range_same_count_int }
4511 }
4512 {
4513 % Second or more in the range, but not the last.
4514 \int_incr:N \l__zrefclever_range_count_int
4515 \bool_if:NT \l__zrefclever_next_is_same_bool
4516 { \int_incr:N \l__zrefclever_range_same_count_int }
4517 }
4518 }
4519 {
4520 % Next element is not in sequence: there was no range, or we are
4521 % closing one.
4522 \int_case:nnF { \l__zrefclever_range_count_int }
4523 {
4524 % There was no range going on.
4525 % Test: `zc-typeset01.lvt': "Not last of type: no range"
4526 { 0 }
4527 {
4528 \int_incr:N \l__zrefclever_ref_count_int
4529 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4530 {
4531 \exp_not:V \l__zrefclever_listsep_tl
4532 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4533 \l__zrefclever_refbounds_mid_seq
4534 }
4535 }
4536 % Last is second in the range: if `range_same_count' is also
4537 % `1', it's a repetition (drop it), otherwise, it's a "pair
4538 % within a list", treat as list.
4539 % Test: `zc-typeset01.lvt': "Not last of type: range pair to one"
4540 % Test: `zc-typeset01.lvt': "Not last of type: range pair"
4541 { 1 }
4542 {
4543 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4544 {
4545 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4546 \l__zrefclever_refbounds_first_seq

```

```

4547         \bool_set_true:N
4548         \l__zrefclever_type_first_refbounds_set_bool
4549     }
4550     {
4551         \int_incr:N \l__zrefclever_ref_count_int
4552         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4553         {
4554             \exp_not:V \l__zrefclever_listsep_tl
4555             \__zrefclever_get_ref:VN
4556             \l__zrefclever_range_beg_label_tl
4557             \l__zrefclever_refbounds_mid_seq
4558         }
4559     }
4560 \int_compare:nNnF
4561 { \l__zrefclever_range_same_count_int } = { 1 }
4562 {
4563     \int_incr:N \l__zrefclever_ref_count_int
4564     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4565     {
4566         \exp_not:V \l__zrefclever_listsep_tl
4567         \__zrefclever_get_ref:VN
4568         \l__zrefclever_label_a_tl
4569         \l__zrefclever_refbounds_mid_seq
4570     }
4571 }
4572 }
4573 }
4574 {
4575     % Last is third or more in the range: if `range_count' and
4576     % `range_same_count' are the same, its a repetition (drop it),
4577     % if they differ by `1', its a list, if they differ by more,
4578     % it is a real range.
4579     \int_case:nnF
4580     {
4581         \l__zrefclever_range_count_int -
4582         \l__zrefclever_range_same_count_int
4583     }
4584     {
4585         % Test: `zc-typeset01.lvt': "Not last of type: range to one"
4586         { 0 }
4587         {
4588             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4589             {
4590                 \seq_set_eq:NN
4591                 \l__zrefclever_type_first_refbounds_seq
4592                 \l__zrefclever_refbounds_first_seq
4593                 \bool_set_true:N
4594                 \l__zrefclever_type_first_refbounds_set_bool
4595             }
4596             {
4597                 \int_incr:N \l__zrefclever_ref_count_int
4598                 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4599                 {
4600                     \exp_not:V \l__zrefclever_listsep_tl

```

```

4601         \__zrefclever_get_ref:VN
4602         \l__zrefclever_range_beg_label_tl
4603         \l__zrefclever_refbounds_mid_seq
4604     }
4605 }
4606 }
4607 % Test: `zc-typeset01.lvt': "Not last of type: range to pair"
4608 { 1 }
4609 {
4610     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4611     {
4612         \seq_set_eq:NN
4613         \l__zrefclever_type_first_refbounds_seq
4614         \l__zrefclever_refbounds_first_seq
4615         \bool_set_true:N
4616         \l__zrefclever_type_first_refbounds_set_bool
4617     }
4618     {
4619         \int_incr:N \l__zrefclever_ref_count_int
4620         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4621         {
4622             \exp_not:V \l__zrefclever_listsep_tl
4623             \__zrefclever_get_ref:VN
4624             \l__zrefclever_range_beg_label_tl
4625             \l__zrefclever_refbounds_mid_seq
4626         }
4627     }
4628     \int_incr:N \l__zrefclever_ref_count_int
4629     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4630     {
4631         \exp_not:V \l__zrefclever_listsep_tl
4632         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4633         \l__zrefclever_refbounds_mid_seq
4634     }
4635 }
4636 }
4637 {
4638 % Test: `zc-typeset01.lvt': "Not last of type: range"
4639 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4640 {
4641     \seq_set_eq:NN
4642     \l__zrefclever_type_first_refbounds_seq
4643     \l__zrefclever_refbounds_first_rb_seq
4644     \bool_set_true:N
4645     \l__zrefclever_type_first_refbounds_set_bool
4646 }
4647 {
4648     \int_incr:N \l__zrefclever_ref_count_int
4649     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4650     {
4651         \exp_not:V \l__zrefclever_listsep_tl
4652         \__zrefclever_get_ref:VN
4653         \l__zrefclever_range_beg_label_tl
4654         \l__zrefclever_refbounds_mid_rb_seq

```



```

4655     }
4656   }
4657   % For the purposes of the serial comma, and thus for the
4658   % distinction of `lastsep' and `pairsep', a "range" counts
4659   % as one. Since `range_beg' has already been counted
4660   % (here or with the first of type), we refrain from
4661   % incrementing `ref_count_int'.
4662   \bool_lazy_and:nnTF
4663   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4664   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4665   {
4666     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4667     \l__zrefclever_range_beg_label_tl
4668     \l__zrefclever_label_a_tl
4669     \l__zrefclever_range_end_ref_tl
4670     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4671     {
4672       \exp_not:V \l__zrefclever_rangesep_tl
4673       \__zrefclever_get_ref_endrange:VVN
4674       \l__zrefclever_label_a_tl
4675       \l__zrefclever_range_end_ref_tl
4676       \l__zrefclever_refbounds_mid_re_seq
4677     }
4678   }
4679   {
4680     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4681     {
4682       \exp_not:V \l__zrefclever_rangesep_tl
4683       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4684       \l__zrefclever_refbounds_mid_re_seq
4685     }
4686   }
4687 }
4688 }
4689 % We just closed a range, reset `range_beg_is_first' in case a
4690 % second range for the same type occurs, in which case its
4691 % `range_beg' will no longer be `first'.
4692 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4693 % Reset counters.
4694 \int_zero:N \l__zrefclever_range_count_int
4695 \int_zero:N \l__zrefclever_range_same_count_int
4696 }
4697 }
4698 % Step label counter for next iteration.
4699 \int_incr:N \l__zrefclever_label_count_int
4700 }

```

(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 T_EXnical 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.

```
4701 \cs_new_protected:Npn \__zrefclever_ref_default:
4702   { \zref@default }
4703 \cs_new_protected:Npn \__zrefclever_name_default:
4704   { \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 re-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>}

4705 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4706   {
4707     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4708     {
4709       \bool_if:nTF
4710         {
4711           \l__zrefclever_hyperlink_bool &&
4712           ! \l__zrefclever_link_star_bool
4713         }
4714         {
4715           \seq_item:Nn #2 { 1 }
4716           \__zrefclever_hyperlink:nnn
4717             { \__zrefclever_extract_url_unexp:n {#1} }
4718             { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4719             {
4720               \seq_item:Nn #2 { 2 }
4721               \exp_not:N \group_begin:
```

```

4722         \exp_not:V \l__zrefclever_reffont_tl
4723         \__zrefclever_extract_unexp:nvn {#1}
4724         { l__zrefclever_ref_property_tl } { }
4725         \exp_not:N \group_end:
4726         \seq_item:Nn #2 { 3 }
4727     }
4728     \seq_item:Nn #2 { 4 }
4729 }
4730 {
4731     \seq_item:Nn #2 { 1 }
4732     \seq_item:Nn #2 { 2 }
4733     \exp_not:N \group_begin:
4734     \exp_not:V \l__zrefclever_reffont_tl
4735     \__zrefclever_extract_unexp:nvn {#1}
4736     { l__zrefclever_ref_property_tl } { }
4737     \exp_not:N \group_end:
4738     \seq_item:Nn #2 { 3 }
4739     \seq_item:Nn #2 { 4 }
4740 }
4741 }
4742 { \__zrefclever_ref_default: }
4743 }
4744 \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>}
4745 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3
4746 {
4747     \str_if_eq:nnTF {#2} { zc@missingproperty }
4748     { \__zrefclever_ref_default: }
4749     {
4750         \bool_if:nTF
4751         {
4752             \l__zrefclever_hyperlink_bool &&
4753             ! \l__zrefclever_link_star_bool
4754         }
4755         {
4756             \seq_item:Nn #3 { 1 }
4757             \__zrefclever_hyperlink:nnn
4758             { \__zrefclever_extract_url_unexp:n {#1} }
4759             { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4760             {
4761                 \seq_item:Nn #3 { 2 }
4762                 \exp_not:N \group_begin:
4763                 \exp_not:V \l__zrefclever_reffont_tl
4764                 \exp_not:n {#2}
4765                 \exp_not:N \group_end:
4766                 \seq_item:Nn #3 { 3 }
4767             }
4768             \seq_item:Nn #3 { 4 }
4769         }
4770         {
4771             \seq_item:Nn #3 { 1 }

```

```

4772         \seq_item:Nn #3 { 2 }
4773         \exp_not:N \group_begin:
4774             \exp_not:V \l__zrefclever_reffont_tl
4775             \exp_not:n {#2}
4776         \exp_not:N \group_end:
4777         \seq_item:Nn #3 { 3 }
4778         \seq_item:Nn #3 { 4 }
4779     }
4780 }
4781 }
4782 \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.

```

4783 \cs_new:Npn \__zrefclever_get_ref_first:
4784 {
4785     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4786     { \__zrefclever_ref_default: }
4787     {
4788         \bool_if:NTF \l__zrefclever_name_in_link_bool
4789         {
4790             \zref@ifrefcontainsprop
4791             { \l__zrefclever_type_first_label_tl }
4792             { \l__zrefclever_ref_property_tl }
4793             {
4794                 \__zrefclever_hyperlink:nnn
4795                 {
4796                     \__zrefclever_extract_url_unexp:V
4797                     \l__zrefclever_type_first_label_tl
4798                 }
4799                 {
4800                     \__zrefclever_extract_unexp:Vnn
4801                     \l__zrefclever_type_first_label_tl { anchor } { }
4802                 }
4803             }
4804             \exp_not:N \group_begin:
4805                 \exp_not:V \l__zrefclever_namefont_tl
4806                 \exp_not:V \l__zrefclever_type_name_tl
4807             \exp_not:N \group_end:
4808             \exp_not:V \l__zrefclever_namesep_tl
4809             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4810             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4811             \exp_not:N \group_begin:
4812                 \exp_not:V \l__zrefclever_reffont_tl
4813                 \__zrefclever_extract_unexp:Vnn
4814                 \l__zrefclever_type_first_label_tl

```

```

4815         { l__zrefclever_ref_property_tl } { }
4816         \exp_not:N \group_end:
4817         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4818     }
4819     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4820 }
4821 {
4822     \exp_not:N \group_begin:
4823     \exp_not:V \l__zrefclever_namefont_tl
4824     \exp_not:V \l__zrefclever_type_name_tl
4825     \exp_not:N \group_end:
4826     \exp_not:V \l__zrefclever_namesep_tl
4827     \__zrefclever_ref_default:
4828 }
4829 }
4830 {
4831     \bool_if:nTF \l__zrefclever_type_name_missing_bool
4832     {
4833         \__zrefclever_name_default:
4834         \exp_not:V \l__zrefclever_namesep_tl
4835     }
4836     {
4837         \exp_not:N \group_begin:
4838         \exp_not:V \l__zrefclever_namefont_tl
4839         \exp_not:V \l__zrefclever_type_name_tl
4840         \exp_not:N \group_end:
4841         \tl_if_empty:NF \l__zrefclever_type_name_tl
4842         { \exp_not:V \l__zrefclever_namesep_tl }
4843     }
4844     \zref@ifrefcontainsprop
4845     { \l__zrefclever_type_first_label_tl }
4846     { \l__zrefclever_ref_property_tl }
4847     {
4848         \bool_if:nTF
4849         {
4850             \l__zrefclever_hyperlink_bool &&
4851             ! \l__zrefclever_link_star_bool
4852         }
4853         {
4854             \seq_item:Nn
4855             \l__zrefclever_type_first_refbounds_seq { 1 }
4856             \__zrefclever_hyperlink:nnn
4857             {
4858                 \__zrefclever_extract_url_unexp:V
4859                 \l__zrefclever_type_first_label_tl
4860             }
4861             {
4862                 \__zrefclever_extract_unexp:Vnn
4863                 \l__zrefclever_type_first_label_tl { anchor } { }
4864             }
4865             {
4866                 \seq_item:Nn
4867                 \l__zrefclever_type_first_refbounds_seq { 2 }
4868                 \exp_not:N \group_begin:

```

```

4869         \exp_not:V \l__zrefclever_reffont_tl
4870         \__zrefclever_extract_unexp:Vvn
4871         \l__zrefclever_type_first_label_tl
4872         { \l__zrefclever_ref_property_tl } { }
4873     \exp_not:N \group_end:
4874     \seq_item:Nn
4875         \l__zrefclever_type_first_refbounds_seq { 3 }
4876     }
4877     \seq_item:Nn
4878         \l__zrefclever_type_first_refbounds_seq { 4 }
4879     }
4880     {
4881     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4882     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4883     \exp_not:N \group_begin:
4884     \exp_not:V \l__zrefclever_reffont_tl
4885     \__zrefclever_extract_unexp:Vvn
4886     \l__zrefclever_type_first_label_tl
4887     { \l__zrefclever_ref_property_tl } { }
4888     \exp_not:N \group_end:
4889     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4890     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4891     }
4892     }
4893     { \__zrefclever_ref_default: }
4894     }
4895 }
4896 }

```

(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`.

```

4897 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4898     {
4899     \bool_if:nTF
4900     { \l__zrefclever_typeset_ref_bool && ! \l__zrefclever_typeset_name_bool }
4901     {
4902     % `typeset=ref' / `noname' option
4903     % Probably redundant, since in this case the type name is not being
4904     % typeset. But, for completeness sake:
4905     \tl_clear:N \l__zrefclever_type_name_tl
4906     \bool_set_false:N \l__zrefclever_name_in_link_bool

```

```

4907     \bool_set_true:N \l__zrefclever_type_name_missing_bool
4908   }
4909   {
4910     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4911     {
4912       \tl_clear:N \l__zrefclever_type_name_tl
4913       \bool_set_true:N \l__zrefclever_type_name_missing_bool
4914     }
4915     {
4916       \tl_if_eq:NnTF
4917         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4918         {
4919           \tl_clear:N \l__zrefclever_type_name_tl
4920           \bool_set_true:N \l__zrefclever_type_name_missing_bool
4921         }
4922         {
4923           % Determine whether we should use capitalization,
4924           % abbreviation, and plural.
4925           \bool_lazy_or:nnTF
4926             { \l__zrefclever_cap_bool }
4927             {
4928               \l__zrefclever_capfirst_bool &&
4929               \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4930             }
4931             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4932             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4933           % If the queue is empty, we have a singular, otherwise,
4934           % plural.
4935           \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4936             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4937             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4938           \bool_lazy_and:nnTF
4939             { \l__zrefclever_abbrev_bool }
4940             {
4941               ! \int_compare_p:nNn
4942                 { \l__zrefclever_type_count_int } = { 0 } ||
4943               ! \l__zrefclever_noabbrev_first_bool
4944             }
4945             {
4946               \tl_set:NV \l__zrefclever_name_format_fallback_tl
4947                 \l__zrefclever_name_format_tl
4948               \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4949             }
4950             { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4951           % Handle number and gender nudges.
4952           % Note that these nudges get disabled for `typeset=ref' /
4953           % `noname' option, but in this case they are not really
4954           % meaningful anyway.
4955           \bool_if:NT \l__zrefclever_nudge_enabled_bool
4956           {
4957             \bool_if:NTF \l__zrefclever_nudge_singular_bool
4958             {
4959               \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4960               {

```

```

4961         \msg_warning:nne { zref-clever }
4962         { nudge-plural-when-sg }
4963         { \l__zrefclever_type_first_label_type_tl }
4964     }
4965 }
4966 {
4967     \bool_lazy_all:nT
4968     {
4969         { \l__zrefclever_nudge_comptosing_bool }
4970         { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4971         {
4972             \int_compare_p:nNn
4973             { \l__zrefclever_label_count_int } > { 0 }
4974         }
4975     }
4976     {
4977         \msg_warning:nne { zref-clever }
4978         { nudge-comptosing }
4979         { \l__zrefclever_type_first_label_type_tl }
4980     }
4981 }
4982 \bool_lazy_and:nnT
4983 { \l__zrefclever_nudge_gender_bool }
4984 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4985 {
4986     \__zrefclever_get_rf_opt_seq:neeN { gender }
4987     { \l__zrefclever_type_first_label_type_tl }
4988     { \l__zrefclever_ref_language_tl }
4989     \l__zrefclever_type_name_gender_seq
4990     \seq_if_in:NVF
4991     \l__zrefclever_type_name_gender_seq
4992     \l__zrefclever_ref_gender_tl
4993     {
4994         \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4995         {
4996             \msg_warning:nneee { zref-clever }
4997             { nudge-gender-not-declared-for-type }
4998             { \l__zrefclever_ref_gender_tl }
4999             { \l__zrefclever_type_first_label_type_tl }
5000             { \l__zrefclever_ref_language_tl }
5001         }
5002     }
5003     {
5004         \msg_warning:nneeee { zref-clever }
5005         { nudge-gender-mismatch }
5006         { \l__zrefclever_type_first_label_type_tl }
5007         { \l__zrefclever_ref_gender_tl }
5008         {
5009             \seq_use:Nn
5010             \l__zrefclever_type_name_gender_seq { ,~ }
5011         }
5012         { \l__zrefclever_ref_language_tl }
5013     }
5014 }

```



```

5015     }
5016 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5017 {
5018     \__zrefclever_opt_tl_get:cNF
5019     {
5020         \__zrefclever_opt_varname_type:een
5021         { \l__zrefclever_type_first_label_type_tl }
5022         { \l__zrefclever_name_format_tl }
5023         { tl }
5024     }
5025     \l__zrefclever_type_name_tl
5026     {
5027         \tl_if_empty:NF \l__zrefclever_ref_variant_tl
5028         {
5029             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5030             \tl_put_left:NV \l__zrefclever_name_format_tl
5031             \l__zrefclever_ref_variant_tl
5032         }
5033         \__zrefclever_opt_tl_get:cNF
5034         {
5035             \__zrefclever_opt_varname_lang_type:eeen
5036             { \l__zrefclever_ref_language_tl }
5037             { \l__zrefclever_type_first_label_type_tl }
5038             { \l__zrefclever_name_format_tl }
5039             { tl }
5040         }
5041         \l__zrefclever_type_name_tl
5042         {
5043             \tl_clear:N \l__zrefclever_type_name_tl
5044             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5045             \msg_warning:nnee { zref-clever } { missing-name }
5046             { \l__zrefclever_name_format_tl }
5047             { \l__zrefclever_type_first_label_type_tl }
5048         }
5049     }
5050 }
5051 {
5052     \__zrefclever_opt_tl_get:cNF
5053     {
5054         \__zrefclever_opt_varname_type:een
5055         { \l__zrefclever_type_first_label_type_tl }
5056         { \l__zrefclever_name_format_tl }
5057         { tl }
5058     }
5059     \l__zrefclever_type_name_tl
5060     {
5061         \__zrefclever_opt_tl_get:cNF
5062         {
5063             \__zrefclever_opt_varname_type:een
5064             { \l__zrefclever_type_first_label_type_tl }
5065             { \l__zrefclever_name_format_fallback_tl }
5066             { tl }
5067         }
5068         \l__zrefclever_type_name_tl

```

```

5069         {
5070         \tl_if_empty:NF \l__zrefclever_ref_variant_tl
5071         {
5072             \tl_put_left:Nn
5073             \l__zrefclever_name_format_tl { - }
5074             \tl_put_left:NV \l__zrefclever_name_format_tl
5075             \l__zrefclever_ref_variant_tl
5076             \tl_put_left:Nn
5077             \l__zrefclever_name_format_fallback_tl { - }
5078             \tl_put_left:NV
5079             \l__zrefclever_name_format_fallback_tl
5080             \l__zrefclever_ref_variant_tl
5081         }
5082         \__zrefclever_opt_tl_get:cNF
5083         {
5084             \__zrefclever_opt_varname_lang_type:eeen
5085             { \l__zrefclever_ref_language_tl }
5086             { \l__zrefclever_type_first_label_type_tl }
5087             { \l__zrefclever_name_format_tl }
5088             { tl }
5089         }
5090         \l__zrefclever_type_name_tl
5091         {
5092             \__zrefclever_opt_tl_get:cNF
5093             {
5094                 \__zrefclever_opt_varname_lang_type:eeen
5095                 { \l__zrefclever_ref_language_tl }
5096                 { \l__zrefclever_type_first_label_type_tl }
5097                 { \l__zrefclever_name_format_fallback_tl }
5098                 { tl }
5099             }
5100             \l__zrefclever_type_name_tl
5101             {
5102                 \tl_clear:N \l__zrefclever_type_name_tl
5103                 \bool_set_true:N
5104                 \l__zrefclever_type_name_missing_bool
5105                 \msg_warning:nnee { zref-clever }
5106                 { missing-name }
5107                 { \l__zrefclever_name_format_tl }
5108                 { \l__zrefclever_type_first_label_type_tl }
5109             }
5110         }
5111     }
5112 }
5113 }
5114 }
5115 }
5116 % Signal whether the type name is to be included in the hyperlink or
5117 % not.
5118 \bool_lazy_any:nTF
5119 {
5120     { ! \l__zrefclever_hyperlink_bool }
5121     { \l__zrefclever_link_star_bool }
5122     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }

```

```

5123     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5124   }
5125   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5126   {
5127     \bool_lazy_any:nTF
5128     {
5129       { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5130       {
5131         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5132         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5133       }
5134       {
5135         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5136         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5137         \l__zrefclever_typeset_last_bool &&
5138         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5139       }
5140     }
5141     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5142     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5143   }
5144 }
5145 }

```

(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>}
5146 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5147 {
5148   \tl_if_empty:nTF {#1}
5149   { \hyperlink {#2} {#3} }
5150   { \hyper@linkfile {#3} {#1} {#2} }
5151 }

```

(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`.

```

5152 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5153 {
5154   \zref@ifpropundefined { urluse }
5155   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5156   {
5157     \zref@ifrefcontainsprop {#1} { urluse }
5158     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5159     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5160   }
5161 }
5162 \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 `\langle label b \rangle` comes in immediate sequence from `\langle label a \rangle`. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__zrefclever_next_is_same_bool` to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside `__zrefclever_typeset_refs_not_last_of_type:`, so this function is expected to be called at its beginning, if compression is enabled.

```

\__zrefclever_labels_in_sequence:nn {\langle label a \rangle} {\langle label b \rangle}
5163 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5164 {
5165   \exp_args:Nee \tl_if_eq:nnT
5166   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5167   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5168   {
5169     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5170     {
5171       \exp_args:Nee \tl_if_eq:nnT
5172       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5173       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5174       {
5175         \int_compare:nNnTF
5176         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5177         =
5178         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5179         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5180         {
5181           \int_compare:nNnTF
5182           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5183           =
5184           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5185           {
5186             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5187             \bool_set_true:N \l__zrefclever_next_is_same_bool
5188           }
5189         }
5190       }
5191     }
5192   }
5193   \exp_args:Nee \tl_if_eq:nnT
5194   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5195   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5196   {
5197     \exp_args:Nee \tl_if_eq:nnT
5198     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5199     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5200     {
5201       \int_compare:nNnTF
5202       { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5203       =
5204       { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }

```

```

5205         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5206         {
5207         \int_compare:nNnT
5208             { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5209             =
5210             { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5211         {

```

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.

```

5212         \exp_args:Nee \tl_if_eq:nnT
5213         {
5214             \__zrefclever_extract_unexp:nvn {#1}
5215             { l__zrefclever_ref_property_tl } { }
5216         }
5217         {
5218             \__zrefclever_extract_unexp:nvn {#2}
5219             { l__zrefclever_ref_property_tl } { }
5220         }
5221         {
5222             \bool_set_true:N
5223             \l__zrefclever_next_maybe_range_bool
5224             \bool_set_true:N
5225             \l__zrefclever_next_is_same_bool
5226         }
5227     }
5228 }
5229 }
5230 }
5231 }
5232 }
5233 }

```

(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>}
5234 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5235 {
5236   % First attempt: general options.
5237   \__zrefclever_opt_tl_get:cNF
5238   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5239   #4
5240   {
5241     % If not found, try type specific options.
5242     \__zrefclever_opt_tl_get:cNF
5243     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }

```

```

5244         #4
5245         {
5246             % If not found, try type- and language-specific.
5247             \_zrefclever_opt_tl_get:cNF
5248             { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { t1 } }
5249             #4
5250             {
5251                 % If not found, try language-specific default.
5252                 \_zrefclever_opt_tl_get:cNF
5253                 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { t1 } }
5254                 #4
5255                 {
5256                     % If not found, try fallback.
5257                     \_zrefclever_opt_tl_get:cNF
5258                     { \_zrefclever_opt_varname_fallback:nn {#1} { t1 } }
5259                     #4
5260                     { \tl_clear:N #4 }
5261                 }
5262             }
5263         }
5264     }
5265 }
5266 \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>}
5267 \cs_new_protected:Npn \_zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5268 {
5269     % First attempt: general options.
5270     \_zrefclever_opt_seq_get:cNF
5271     { \_zrefclever_opt_varname_general:nn {#1} { seq } }
5272     #4
5273     {
5274         % If not found, try type specific options.
5275         \_zrefclever_opt_seq_get:cNF
5276         { \_zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5277         #4
5278         {
5279             % If not found, try type- and language-specific.
5280             \_zrefclever_opt_seq_get:cNF
5281             { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5282             #4
5283             {
5284                 % If not found, try language-specific default.
5285                 \_zrefclever_opt_seq_get:cNF
5286                 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5287                 #4
5288                 {
5289                     % If not found, try fallback.
5290                     \_zrefclever_opt_seq_get:cNF
5291                     { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5292                     #4

```

```

5293         { \seq_clear:N #4 }
5294     }
5295 }
5296 }
5297 }
5298 }
5299 \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>}
5300 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5301 {
5302   % First attempt: general options.
5303   \__zrefclever_opt_bool_get:cNF
5304   { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5305   #5
5306   {
5307     % If not found, try type specific options.
5308     \__zrefclever_opt_bool_get:cNF
5309     { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5310     #5
5311     {
5312       % If not found, try type- and language-specific.
5313       \__zrefclever_opt_bool_get:cNF
5314       { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5315       #5
5316       {
5317         % If not found, try language-specific default.
5318         \__zrefclever_opt_bool_get:cNF
5319         { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5320         #5
5321         {
5322           % If not found, try fallback.
5323           \__zrefclever_opt_bool_get:cNF
5324           { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5325           #5
5326           { \use:c { bool_set_ #2 :N } #5 }
5327         }
5328       }
5329     }
5330   }
5331 }
5332 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { neeN }

```

(End of definition for __zrefclever_get_rf_opt_bool:nnnnN.)

9 Compatibility

This section is meant to aggregate any “special handling” needed for L^AT_EX kernel features, document classes, and packages, needed for zref-clever to work properly with them.

9.1 appendix

One relevant case of different reference types sharing the same counter is the `\appendix` which in some document classes, including the standard ones, change the sectioning commands looks but, of course, keep using the same counter. `book.cls` and `report.cls` reset counters `chapter` and `section` to 0, change `\@chapapp` to use `\appendixname` and use `\@Alph` for `\thechapter`. `article.cls` resets counters `section` and `subsection` to 0, and uses `\@Alph` for `\thesection`. `memoir.cls`, `scrbook.cls` and `scrarticle.cls` do the same as their corresponding standard classes, and sometimes a little more, but what interests us here is pretty much the same. See also the `appendix` package.

The standard `\appendix` command is a one way switch, in other words, it cannot be reverted (see <https://tex.stackexchange.com/a/444057>). So, even if the fact that it is a “switch” rather than an environment complicates things, because we have to make ungrouped settings to correspond to its effects, in practice this is not a big deal, since these settings are never really reverted (by default, at least). Hence, hooking into `\appendix` is a viable and natural alternative. The `memoir` class and the `appendix` package define the `appendices` and `subappendices` environments, which provide for a way for the appendix to “end”, but in this case, of course, we can hook into the environment instead.

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

```
5333 \__zrefclever_compat_module:nn { appendix }
5334   {
5335     \newcounter { zc@appendix }
5336     \cs_if_exist:cTF { chapter }
5337       {
5338         \__zrefclever_zcsetup:e
5339         {
5340           counterresetby =
5341             {
```

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

```
5342           zc@appendix = \__zrefclever_counter_reset_by:n { chapter } ,
5343           chapter = zc@appendix ,
5344         } ,
5345       }
5346     }
5347   {
5348     \cs_if_exist:cT { section }
5349     {
5350       \__zrefclever_zcsetup:e
5351       {
5352         counterresetby =
5353           {
5354             zc@appendix = \__zrefclever_counter_reset_by:n { section } ,
5355             section = zc@appendix ,
5356           } ,
5357       }
5358     }
5359   }
5360 \AddToHook { cmd / appendix / before }
5361   {
5362     \setcounter { zc@appendix } { 1 }
5363     \__zrefclever_zcsetup:n
```



```

5364     {
5365         countertype =
5366         {
5367             chapter      = appendix ,
5368             section      = appendix ,
5369             subsection   = appendix ,
5370             subsubsection = appendix ,
5371             paragraph    = appendix ,
5372             subparagraph = appendix ,
5373         }
5374     }
5375 }
5376 }

```

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.

```

5377 \__zrefclever_compat_module:nn { appendices }
5378 {
5379     \__zrefclever_if_package_loaded:nT { appendix }
5380     {
5381         \AddToHook { env / appendices / begin }
5382         {

```

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.

```

5383         \setcounter { zc@appendix } { 1 }
5384         \__zrefclever_zcsetup:n
5385         {
5386             countertype =
5387             {

```

```

5388         chapter      = appendix ,
5389         section       = appendix ,
5390         subsection    = appendix ,
5391         subsection    = appendix ,
5392         paragraph     = appendix ,
5393         subparagraph  = appendix ,
5394     }
5395 }
5396 }
5397 \AddToHook { env / appendices / end }
5398 { \setcounter { zc@appendix } { 0 } }
5399 \newcounter { zc@subappendix }
5400 \cs_if_exist:cTF { chapter }
5401 {
5402     \__zrefclever_zcsetup:e
5403     {
5404         counterresetby =
5405         {
5406             zc@subappendix = \__zrefclever_counter_reset_by:n { section } ,
5407             section = zc@subappendix ,
5408         } ,
5409     }
5410 }
5411 {
5412     \__zrefclever_zcsetup:e
5413     {
5414         counterresetby =
5415         {
5416             zc@subappendix = \__zrefclever_counter_reset_by:n { subsection } ,
5417             subsection = zc@subappendix ,
5418         } ,
5419     }
5420 }
5421 \AddToHook { env / subappendices / begin }
5422 {

```

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.

```

5423     \setcounter { zc@subappendix } { 1 }
5424     \__zrefclever_zcsetup:n
5425     {
5426         countertype =
5427         {
5428             section      = appendix ,
5429             subsection   = appendix ,
5430             subsection   = appendix ,
5431             paragraph    = appendix ,
5432             subparagraph = appendix ,
5433         } ,
5434     }
5435 }
5436 \AddToHook { env / subappendices / end }
5437 { \setcounter { zc@subappendix } { 0 } }

```

```

5438     \msg_info:nnn { zref-clever } { compat-package } { appendix }
5439   }
5440 }

```

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.

```

5441 \__zrefclever_compat_module:nn { memoir }
5442 {
5443   \__zrefclever_if_class_loaded:nT { memoir }
5444   {

```

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`.

```

5445   \__zrefclever_zcsetup:n
5446   {
5447     countertype =
5448     {
5449       subfigure = figure ,
5450       subtable  = table ,
5451       poemline  = line ,
5452     } ,
5453     counterresetby =
5454     {
5455       subfigure = figure ,
5456       subtable  = table ,
5457     } ,
5458   }

```

Support for subcaption references.

```

5459   \zref@newprop { subcaption }
5460   { \cs_if_exist_use:c { @thesub \@capytype } }
5461   \AddToHook{ memoir/subcaption/aftercounter }
5462   { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5463   \__zrefclever_zcsetup:n
5464   {

```

```

5465         countertype =
5466         {
5467             sidefootnote = footnote ,
5468             pagenote = endnote ,
5469         } ,
5470     }
5471     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5472 }
5473 }

```

9.4 amsmath

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

```

5474 \__zrefclever_compat_module:nn { amsmath }
5475 {
5476     \__zrefclever_if_package_loaded:nT { amsmath }
5477     {

```

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

```

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

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout and 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 accept labels or tags anyway. The `-ed` (`gathered`, `aligned`, and `alignedat`) and `cases` environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like `array`, `matrix` and variations. Finally, `split` too can only be used as part of another environment. We also arrange, at this point, for the provision of the `subeq` property, for the convenience of referring to them directly or to build terse ranges with the `endrange` option.

```

5494     \zref@newprop { subeq } { \alph { equation } }
5495     \clist_map_inline:nn
5496       {
5497         equation ,
5498         equation* ,
5499         align ,
5500         align* ,
5501         alignat ,
5502         alignat* ,
5503         flalign ,
5504         flalign* ,
5505         xalignat ,
5506         xalignat* ,
5507         gather ,
5508         gather* ,
5509         multiline ,
5510         multiline* ,
5511       }
5512     {
5513       \AddToHook { env / #1 / begin }
5514         {
5515           \__zrefclever_zcsetup:n { currentcounter = equation }
5516           \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5517             { \zref@localaddprop \ZREF@mainlist { subeq } }
5518         }
5519     }
5520     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5521 }
5522 }
```

9.5 mathtools

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

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

```

5523 \bool_new:N \l__zrefclever_mathtools_loaded_bool
5524 \__zrefclever_compat_module:nn { mathtools }
5525 {
5526   \__zrefclever_if_package_loaded:nT { mathtools }
5527   {
5528     \bool_set_true:N \l__zrefclever_mathtools_loaded_bool
5529     \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5530     {
5531       \seq_map_inline:Nn #1
5532       {
5533         \tl_set:Nc \l__zrefclever_tmpa_tl
5534         { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5535         \bool_lazy_or:nnT
5536         { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { equation } }
5537         { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { parentequation } }
5538         { \noeqref {##1} }
5539       }
5540     }
5541     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5542   }
5543 }

```

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.

```

5544 \__zrefclever_compat_module:nn { breqn }
5545 {
5546   \__zrefclever_if_package_loaded:nT { breqn }
5547   {

```

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>).

```

5548   \bool_new:N \l__zrefclever_breqn_dgroup_bool
5549   \AddToHook { env / dgroup / begin }
5550   {
5551     \__zrefclever_zcsetup:e
5552     {
5553       counterresetby =
5554       {

```

```

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

```

9.7 listings

```

5584 \_zrefclever_compat_module:nn { listings }
5585 {
5586     \_zrefclever_if_package_loaded:nT { listings }
5587     {
5588         \_zrefclever_zcsetup:n
5589         {
5590             countertype =
5591             {
5592                 lstlisting = listing ,
5593                 lstnumber = line ,
5594             } ,
5595             counterresetby = { lstnumber = lstlisting } ,
5596         }

```

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.

```

5597     \cs_if_exist:NT \lst@AddToHook
5598     {
5599         \lst@AddToHook { Init }
5600         { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5601     }
5602     \msg_info:nnn { zref-clever } { compat-package } { listings }
5603 }
5604 }

```

9.8 enumitem

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

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

```

5605 \__zrefclever_compat_module:nn { enumitem }
5606 {
5607     \__zrefclever_if_package_loaded:nT { enumitem }
5608     {
5609         \int_set:Nn \l__zrefclever_tmpa_int { 5 }
5610         \bool_while_do:nn
5611         {
5612             \cs_if_exist_p:c
5613             { c@ enum \int_to_roman:n { \l__zrefclever_tmpa_int } }
5614         }
5615         {
5616             \__zrefclever_zcsetup:e
5617             {
5618                 counterresetby =
5619                 {
5620                     enum \int_to_roman:n { \l__zrefclever_tmpa_int } =
5621                     enum \int_to_roman:n { \l__zrefclever_tmpa_int - 1 }
5622                 } ,
5623                 countertype =
5624                 { enum \int_to_roman:n { \l__zrefclever_tmpa_int } = item } ,
5625             }
5626             \int_incr:N \l__zrefclever_tmpa_int
5627         }
5628         \int_compare:nNnT { \l__zrefclever_tmpa_int } > { 5 }
5629         { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5630     }
5631 }

```


9.9 subcaption

```
5632 \__zrefclever_compat_module:nm { subcaption }
5633 {
5634   \__zrefclever_if_package_loaded:nT { subcaption }
5635   {
5636     \__zrefclever_zcsetup:n
5637     {
5638       countertype =
5639       {
5640         subfigure = figure ,
5641         subtable = table ,
5642       } ,
5643       counterresetby =
5644       {
5645         subfigure = figure ,
5646         subtable = table ,
5647       } ,
5648     }
5649   }
5650 }
```

Support for subref reference.

```
5649   \zref@newprop { subref }
5650   { \cs_if_exist_use:c { thesub \@capttype } }
5651   \tl_if_exist:NT \caption@subtypehook
5652   {
5653     \tl_put_right:Nn \caption@subtypehook
5654     { \zref@localaddprop \ZREF@mainlist { subref } }
5655   }
5656 }
5657 }
```

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.

```
5658 \__zrefclever_compat_module:nm { subfig }
5659 {
5660   \__zrefclever_if_package_loaded:nT { subfig }
5661   {
5662     \__zrefclever_zcsetup:n
5663     {
5664       countertype =
5665       {
5666         subfigure = figure ,
5667         subtable = table ,
5668       } ,
5669       counterresetby =
5670       {
5671         subfigure = figure ,
5672         subtable = table ,
5673       } ,
5674     }
5675   }
5676 }
```

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>.

```
5677 \__zrefclever_compat_module:nn { beamer }
5678   {
5679     \__zrefclever_if_class_loaded:nT { beamer }
5680     {
5681       \AddToHookWithArguments { label } [ zref-clever/compat/beamer ]
5682       { \xdef\@currentHref{#1} }
5683       \DeclareHookRule { label }
5684       { zref-clever/compat/beamer } { before } { zref-clever }
5685       \AddToHookWithArguments { cmd/refcounter/before }
5686       [ zref-clever/compat/beamer ]
5687       { \edef\@currentcounter{#1} }
5688     }
5689   }
5690 </package>
```

10 Language files

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

10.1 Localization guidelines

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

The first general observation is that, contrary to a common initial reaction of those faced with the task of localizing the reference types, is that the job is not quite one of

“translation”. The reference type names are just the internal names used by the package to refer to them, technically, they could just as well be foobars. Of course, for practical reasons, they were chosen to be semantic. However, what we are searching for is not really the translation to the reference type name itself, but rather for the word / term / expression which is typically used to refer to the document object that the reference type is meant to represent. And terms that should work well in the contexts which cross-references are commonly used.

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

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

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

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

Math & Co.: A good number of reference types provided by the package are meant to cater for document objects commonly used in Mathematics and related areas. They

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

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

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

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

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

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

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

10.2 English

English language file has been initially provided by the author.

```

5691 (*package)
5692 \zcDeclareLanguage { english }
5693 \zcDeclareLanguageAlias { american } { english }
5694 \zcDeclareLanguageAlias { australian } { english }
5695 \zcDeclareLanguageAlias { british } { english }
5696 \zcDeclareLanguageAlias { canadian } { english }
5697 \zcDeclareLanguageAlias { newzealand } { english }
5698 \zcDeclareLanguageAlias { UKenglish } { english }
5699 \zcDeclareLanguageAlias { USenglish } { english }
5700 \end{package}

5701 (*lang-english)

5702 namesep = {\nobreakspace} ,
5703 pairsep = {\and\nobreakspace} ,
5704 listsep = {,~} ,
5705 lastsep = {\and\nobreakspace} ,
5706 tpairsep = {\and\nobreakspace} ,
5707 tlistsep = {,~} ,
5708 tlastsep = {,~\and\nobreakspace} ,
5709 notesep = {~} ,
5710 rangesep = {\to\nobreakspace} ,
5711
5712 type = book ,
5713 Name-sg = Book ,
5714 name-sg = book ,
5715 Name-pl = Books ,
5716 name-pl = books ,
5717
5718 type = part ,

```

```

5719 Name-sg = Part ,
5720 name-sg = part ,
5721 Name-pl = Parts ,
5722 name-pl = parts ,
5723
5724 type = chapter ,
5725 Name-sg = Chapter ,
5726 name-sg = chapter ,
5727 Name-pl = Chapters ,
5728 name-pl = chapters ,
5729
5730 type = section ,
5731 Name-sg = Section ,
5732 name-sg = section ,
5733 Name-pl = Sections ,
5734 name-pl = sections ,
5735
5736 type = paragraph ,
5737 Name-sg = Paragraph ,
5738 name-sg = paragraph ,
5739 Name-pl = Paragraphs ,
5740 name-pl = paragraphs ,
5741 Name-sg-ab = Par. ,
5742 name-sg-ab = par. ,
5743 Name-pl-ab = Par. ,
5744 name-pl-ab = par. ,
5745
5746 type = appendix ,
5747 Name-sg = Appendix ,
5748 name-sg = appendix ,
5749 Name-pl = Appendices ,
5750 name-pl = appendices ,
5751
5752 type = page ,
5753 Name-sg = Page ,
5754 name-sg = page ,
5755 Name-pl = Pages ,
5756 name-pl = pages ,
5757 rangesep = {\textendash} ,
5758 rangetopair = false ,
5759
5760 type = line ,
5761 Name-sg = Line ,
5762 name-sg = line ,
5763 Name-pl = Lines ,
5764 name-pl = lines ,
5765
5766 type = figure ,
5767 Name-sg = Figure ,
5768 name-sg = figure ,
5769 Name-pl = Figures ,
5770 name-pl = figures ,
5771 Name-sg-ab = Fig. ,
5772 name-sg-ab = fig. ,

```

```

5773 Name-pl-ab = Figs. ,
5774 name-pl-ab = figs. ,
5775
5776 type = table ,
5777 Name-sg = Table ,
5778 name-sg = table ,
5779 Name-pl = Tables ,
5780 name-pl = tables ,
5781
5782 type = item ,
5783 Name-sg = Item ,
5784 name-sg = item ,
5785 Name-pl = Items ,
5786 name-pl = items ,
5787
5788 type = footnote ,
5789 Name-sg = Footnote ,
5790 name-sg = footnote ,
5791 Name-pl = Footnotes ,
5792 name-pl = footnotes ,
5793
5794 type = endnote ,
5795 Name-sg = Note ,
5796 name-sg = note ,
5797 Name-pl = Notes ,
5798 name-pl = notes ,
5799
5800 type = note ,
5801 Name-sg = Note ,
5802 name-sg = note ,
5803 Name-pl = Notes ,
5804 name-pl = notes ,
5805
5806 type = equation ,
5807 Name-sg = Equation ,
5808 name-sg = equation ,
5809 Name-pl = Equations ,
5810 name-pl = equations ,
5811 Name-sg-ab = Eq. ,
5812 name-sg-ab = eq. ,
5813 Name-pl-ab = Eqs. ,
5814 name-pl-ab = eqs. ,
5815 refbounds-first-sg = {,(,)}, ,
5816 refbounds = {(,,)} ,
5817
5818 type = theorem ,
5819 Name-sg = Theorem ,
5820 name-sg = theorem ,
5821 Name-pl = Theorems ,
5822 name-pl = theorems ,
5823
5824 type = lemma ,
5825 Name-sg = Lemma ,
5826 name-sg = lemma ,

```

```

5827 Name-pl = Lemmas ,
5828 name-pl = lemmas ,
5829
5830 type = corollary ,
5831 Name-sg = Corollary ,
5832 name-sg = corollary ,
5833 Name-pl = Corollaries ,
5834 name-pl = corollaries ,
5835
5836 type = proposition ,
5837 Name-sg = Proposition ,
5838 name-sg = proposition ,
5839 Name-pl = Propositions ,
5840 name-pl = propositions ,
5841
5842 type = definition ,
5843 Name-sg = Definition ,
5844 name-sg = definition ,
5845 Name-pl = Definitions ,
5846 name-pl = definitions ,
5847
5848 type = proof ,
5849 Name-sg = Proof ,
5850 name-sg = proof ,
5851 Name-pl = Proofs ,
5852 name-pl = proofs ,
5853
5854 type = result ,
5855 Name-sg = Result ,
5856 name-sg = result ,
5857 Name-pl = Results ,
5858 name-pl = results ,
5859
5860 type = remark ,
5861 Name-sg = Remark ,
5862 name-sg = remark ,
5863 Name-pl = Remarks ,
5864 name-pl = remarks ,
5865
5866 type = example ,
5867 Name-sg = Example ,
5868 name-sg = example ,
5869 Name-pl = Examples ,
5870 name-pl = examples ,
5871
5872 type = algorithm ,
5873 Name-sg = Algorithm ,
5874 name-sg = algorithm ,
5875 Name-pl = Algorithms ,
5876 name-pl = algorithms ,
5877
5878 type = listing ,
5879 Name-sg = Listing ,
5880 name-sg = listing ,

```



```

5881 Name-pl = Listings ,
5882 name-pl = listings ,
5883
5884 type = exercise ,
5885 Name-sg = Exercise ,
5886 name-sg = exercise ,
5887 Name-pl = Exercises ,
5888 name-pl = exercises ,
5889
5890 type = solution ,
5891 Name-sg = Solution ,
5892 name-sg = solution ,
5893 Name-pl = Solutions ,
5894 name-pl = solutions ,
5895 </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`.

```

5896 <*package>
5897 \zcDeclareLanguage
5898 [ variants = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5899 { german }
5900 \zcDeclareLanguageAlias { ngerman      } { german }
5901 \zcDeclareLanguageAlias { austrian     } { german }
5902 \zcDeclareLanguageAlias { naustrian    } { german }
5903 \zcDeclareLanguageAlias { swissgerman  } { german }
5904 \zcDeclareLanguageAlias { nswissgerman } { german }
5905 </package>
5906 <*lang-german>
5907 namesep = {\nobreakspace} ,
5908 pairsep = {\und\nobreakspace} ,
5909 listsep = {,~} ,
5910 lastsep = {\und\nobreakspace} ,
5911 tpairsep = {\und\nobreakspace} ,
5912 tlistsep = {,~} ,
5913 tlastsep = {\und\nobreakspace} ,
5914 notesep = {~} ,
5915 rangeseq = {\bis\nobreakspace} ,
5916
5917 type = book ,
5918 gender = n ,
5919 variant = N ,
5920 Name-sg = Buch ,
5921 Name-pl = Bücher ,
5922 variant = A ,
5923 Name-sg = Buch ,
5924 Name-pl = Bücher ,
5925 variant = D ,
5926 Name-sg = Buch ,

```

5927 Name-pl = Büchern ,
5928 variant = G ,
5929 Name-sg = Buches ,
5930 Name-pl = Bücher ,
5931
5932 type = part ,
5933 gender = m ,
5934 variant = N ,
5935 Name-sg = Teil ,
5936 Name-pl = Teile ,
5937 variant = A ,
5938 Name-sg = Teil ,
5939 Name-pl = Teile ,
5940 variant = D ,
5941 Name-sg = Teil ,
5942 Name-pl = Teilen ,
5943 variant = G ,
5944 Name-sg = Teiles ,
5945 Name-pl = Teile ,
5946
5947 type = chapter ,
5948 gender = n ,
5949 variant = N ,
5950 Name-sg = Kapitel ,
5951 Name-pl = Kapitel ,
5952 variant = A ,
5953 Name-sg = Kapitel ,
5954 Name-pl = Kapitel ,
5955 variant = D ,
5956 Name-sg = Kapitel ,
5957 Name-pl = Kapiteln ,
5958 variant = G ,
5959 Name-sg = Kapitels ,
5960 Name-pl = Kapitel ,
5961
5962 type = section ,
5963 gender = m ,
5964 variant = N ,
5965 Name-sg = Abschnitt ,
5966 Name-pl = Abschnitte ,
5967 variant = A ,
5968 Name-sg = Abschnitt ,
5969 Name-pl = Abschnitte ,
5970 variant = D ,
5971 Name-sg = Abschnitt ,
5972 Name-pl = Abschnitten ,
5973 variant = G ,
5974 Name-sg = Abschnitts ,
5975 Name-pl = Abschnitte ,
5976
5977 type = paragraph ,
5978 gender = m ,
5979 variant = N ,
5980 Name-sg = Absatz ,

```

5981     Name-pl = Absätze ,
5982     variant = A ,
5983     Name-sg = Absatz ,
5984     Name-pl = Absätze ,
5985     variant = D ,
5986     Name-sg = Absatz ,
5987     Name-pl = Absätzen ,
5988     variant = G ,
5989     Name-sg = Absatzes ,
5990     Name-pl = Absätze ,
5991
5992 type = appendix ,
5993     gender = m ,
5994     variant = N ,
5995     Name-sg = Anhang ,
5996     Name-pl = Anhänge ,
5997     variant = A ,
5998     Name-sg = Anhang ,
5999     Name-pl = Anhänge ,
6000     variant = D ,
6001     Name-sg = Anhang ,
6002     Name-pl = Anhängen ,
6003     variant = G ,
6004     Name-sg = Anhangs ,
6005     Name-pl = Anhänge ,
6006
6007 type = page ,
6008     gender = f ,
6009     variant = N ,
6010     Name-sg = Seite ,
6011     Name-pl = Seiten ,
6012     variant = A ,
6013     Name-sg = Seite ,
6014     Name-pl = Seiten ,
6015     variant = D ,
6016     Name-sg = Seite ,
6017     Name-pl = Seiten ,
6018     variant = G ,
6019     Name-sg = Seite ,
6020     Name-pl = Seiten ,
6021     rangeseq = {\textendash} ,
6022     rangetopair = false ,
6023
6024 type = line ,
6025     gender = f ,
6026     variant = N ,
6027     Name-sg = Zeile ,
6028     Name-pl = Zeilen ,
6029     variant = A ,
6030     Name-sg = Zeile ,
6031     Name-pl = Zeilen ,
6032     variant = D ,
6033     Name-sg = Zeile ,
6034     Name-pl = Zeilen ,

```

```

6035     variant = G ,
6036         Name-sg = Zeile ,
6037         Name-pl = Zeilen ,
6038
6039 type = figure ,
6040     gender = f ,
6041     variant = N ,
6042         Name-sg = Abbildung ,
6043         Name-pl = Abbildungen ,
6044         Name-sg-ab = Abb. ,
6045         Name-pl-ab = Abb. ,
6046     variant = A ,
6047         Name-sg = Abbildung ,
6048         Name-pl = Abbildungen ,
6049         Name-sg-ab = Abb. ,
6050         Name-pl-ab = Abb. ,
6051     variant = D ,
6052         Name-sg = Abbildung ,
6053         Name-pl = Abbildungen ,
6054         Name-sg-ab = Abb. ,
6055         Name-pl-ab = Abb. ,
6056     variant = G ,
6057         Name-sg = Abbildung ,
6058         Name-pl = Abbildungen ,
6059         Name-sg-ab = Abb. ,
6060         Name-pl-ab = Abb. ,
6061
6062 type = table ,
6063     gender = f ,
6064     variant = N ,
6065         Name-sg = Tabelle ,
6066         Name-pl = Tabellen ,
6067     variant = A ,
6068         Name-sg = Tabelle ,
6069         Name-pl = Tabellen ,
6070     variant = D ,
6071         Name-sg = Tabelle ,
6072         Name-pl = Tabellen ,
6073     variant = G ,
6074         Name-sg = Tabelle ,
6075         Name-pl = Tabellen ,
6076
6077 type = item ,
6078     gender = m ,
6079     variant = N ,
6080         Name-sg = Punkt ,
6081         Name-pl = Punkte ,
6082     variant = A ,
6083         Name-sg = Punkt ,
6084         Name-pl = Punkte ,
6085     variant = D ,
6086         Name-sg = Punkt ,
6087         Name-pl = Punkten ,
6088     variant = G ,

```

```

6089     Name-sg = Punktes ,
6090     Name-pl = Punkte ,
6091
6092 type = footnote ,
6093     gender = f ,
6094     variant = N ,
6095     Name-sg = Fußnote ,
6096     Name-pl = Fußnoten ,
6097     variant = A ,
6098     Name-sg = Fußnote ,
6099     Name-pl = Fußnoten ,
6100     variant = D ,
6101     Name-sg = Fußnote ,
6102     Name-pl = Fußnoten ,
6103     variant = G ,
6104     Name-sg = Fußnote ,
6105     Name-pl = Fußnoten ,
6106
6107 type = endnote ,
6108     gender = f ,
6109     variant = N ,
6110     Name-sg = Endnote ,
6111     Name-pl = Endnoten ,
6112     variant = A ,
6113     Name-sg = Endnote ,
6114     Name-pl = Endnoten ,
6115     variant = D ,
6116     Name-sg = Endnote ,
6117     Name-pl = Endnoten ,
6118     variant = G ,
6119     Name-sg = Endnote ,
6120     Name-pl = Endnoten ,
6121
6122 type = note ,
6123     gender = f ,
6124     variant = N ,
6125     Name-sg = Anmerkung ,
6126     Name-pl = Anmerkungen ,
6127     variant = A ,
6128     Name-sg = Anmerkung ,
6129     Name-pl = Anmerkungen ,
6130     variant = D ,
6131     Name-sg = Anmerkung ,
6132     Name-pl = Anmerkungen ,
6133     variant = G ,
6134     Name-sg = Anmerkung ,
6135     Name-pl = Anmerkungen ,
6136
6137 type = equation ,
6138     gender = f ,
6139     variant = N ,
6140     Name-sg = Gleichung ,
6141     Name-pl = Gleichungen ,
6142     variant = A ,

```

```

6143     Name-sg = Gleichung ,
6144     Name-pl = Gleichungen ,
6145     variant = D ,
6146     Name-sg = Gleichung ,
6147     Name-pl = Gleichungen ,
6148     variant = G ,
6149     Name-sg = Gleichung ,
6150     Name-pl = Gleichungen ,
6151     refbounds-first-sg = {(,(),)} ,
6152     refbounds = {(,,)} ,
6153
6154 type = theorem ,
6155     gender = n ,
6156     variant = N ,
6157     Name-sg = Theorem ,
6158     Name-pl = Theoreme ,
6159     variant = A ,
6160     Name-sg = Theorem ,
6161     Name-pl = Theoreme ,
6162     variant = D ,
6163     Name-sg = Theorem ,
6164     Name-pl = Theoremen ,
6165     variant = G ,
6166     Name-sg = Theorems ,
6167     Name-pl = Theoreme ,
6168
6169 type = lemma ,
6170     gender = n ,
6171     variant = N ,
6172     Name-sg = Lemma ,
6173     Name-pl = Lemmata ,
6174     variant = A ,
6175     Name-sg = Lemma ,
6176     Name-pl = Lemmata ,
6177     variant = D ,
6178     Name-sg = Lemma ,
6179     Name-pl = Lemmata ,
6180     variant = G ,
6181     Name-sg = Lemmas ,
6182     Name-pl = Lemmata ,
6183
6184 type = corollary ,
6185     gender = n ,
6186     variant = N ,
6187     Name-sg = Korollar ,
6188     Name-pl = Korollare ,
6189     variant = A ,
6190     Name-sg = Korollar ,
6191     Name-pl = Korollare ,
6192     variant = D ,
6193     Name-sg = Korollar ,
6194     Name-pl = Korollaren ,
6195     variant = G ,
6196     Name-sg = Korollars ,

```

```

6197     Name-pl = Korollare ,
6198
6199 type = proposition ,
6200     gender = m ,
6201     variant = N ,
6202         Name-sg = Satz ,
6203         Name-pl = Sätze ,
6204     variant = A ,
6205         Name-sg = Satz ,
6206         Name-pl = Sätze ,
6207     variant = D ,
6208         Name-sg = Satz ,
6209         Name-pl = Sätzen ,
6210     variant = G ,
6211         Name-sg = Satzes ,
6212         Name-pl = Sätze ,
6213
6214 type = definition ,
6215     gender = f ,
6216     variant = N ,
6217         Name-sg = Definition ,
6218         Name-pl = Definitionen ,
6219     variant = A ,
6220         Name-sg = Definition ,
6221         Name-pl = Definitionen ,
6222     variant = D ,
6223         Name-sg = Definition ,
6224         Name-pl = Definitionen ,
6225     variant = G ,
6226         Name-sg = Definition ,
6227         Name-pl = Definitionen ,
6228
6229 type = proof ,
6230     gender = m ,
6231     variant = N ,
6232         Name-sg = Beweis ,
6233         Name-pl = Beweise ,
6234     variant = A ,
6235         Name-sg = Beweis ,
6236         Name-pl = Beweise ,
6237     variant = D ,
6238         Name-sg = Beweis ,
6239         Name-pl = Beweisen ,
6240     variant = G ,
6241         Name-sg = Beweises ,
6242         Name-pl = Beweise ,
6243
6244 type = result ,
6245     gender = n ,
6246     variant = N ,
6247         Name-sg = Ergebnis ,
6248         Name-pl = Ergebnisse ,
6249     variant = A ,
6250         Name-sg = Ergebnis ,

```

```

6251     Name-pl = Ergebnisse ,
6252     variant = D ,
6253     Name-sg = Ergebnis ,
6254     Name-pl = Ergebnissen ,
6255     variant = G ,
6256     Name-sg = Ergebnisses ,
6257     Name-pl = Ergebnisse ,
6258
6259 type = remark ,
6260     gender = f ,
6261     variant = N ,
6262     Name-sg = Bemerkung ,
6263     Name-pl = Bemerkungen ,
6264     variant = A ,
6265     Name-sg = Bemerkung ,
6266     Name-pl = Bemerkungen ,
6267     variant = D ,
6268     Name-sg = Bemerkung ,
6269     Name-pl = Bemerkungen ,
6270     variant = G ,
6271     Name-sg = Bemerkung ,
6272     Name-pl = Bemerkungen ,
6273
6274 type = example ,
6275     gender = n ,
6276     variant = N ,
6277     Name-sg = Beispiel ,
6278     Name-pl = Beispiele ,
6279     variant = A ,
6280     Name-sg = Beispiel ,
6281     Name-pl = Beispiele ,
6282     variant = D ,
6283     Name-sg = Beispiel ,
6284     Name-pl = Beispielen ,
6285     variant = G ,
6286     Name-sg = Beispiels ,
6287     Name-pl = Beispiele ,
6288
6289 type = algorithm ,
6290     gender = m ,
6291     variant = N ,
6292     Name-sg = Algorithmus ,
6293     Name-pl = Algorithmen ,
6294     variant = A ,
6295     Name-sg = Algorithmus ,
6296     Name-pl = Algorithmen ,
6297     variant = D ,
6298     Name-sg = Algorithmus ,
6299     Name-pl = Algorithmen ,
6300     variant = G ,
6301     Name-sg = Algorithmus ,
6302     Name-pl = Algorithmen ,
6303
6304 type = listing ,

```



```

6305 gender = n ,
6306 variant = N ,
6307     Name-sg = Listing ,
6308     Name-pl = Listings ,
6309 variant = A ,
6310     Name-sg = Listing ,
6311     Name-pl = Listings ,
6312 variant = D ,
6313     Name-sg = Listing ,
6314     Name-pl = Listings ,
6315 variant = G ,
6316     Name-sg = Listings ,
6317     Name-pl = Listings ,
6318
6319 type = exercise ,
6320 gender = f ,
6321 variant = N ,
6322     Name-sg = Übungsaufgabe ,
6323     Name-pl = Übungsaufgaben ,
6324 variant = A ,
6325     Name-sg = Übungsaufgabe ,
6326     Name-pl = Übungsaufgaben ,
6327 variant = D ,
6328     Name-sg = Übungsaufgabe ,
6329     Name-pl = Übungsaufgaben ,
6330 variant = G ,
6331     Name-sg = Übungsaufgabe ,
6332     Name-pl = Übungsaufgaben ,
6333
6334 type = solution ,
6335 gender = f ,
6336 variant = N ,
6337     Name-sg = Lösung ,
6338     Name-pl = Lösungen ,
6339 variant = A ,
6340     Name-sg = Lösung ,
6341     Name-pl = Lösungen ,
6342 variant = D ,
6343     Name-sg = Lösung ,
6344     Name-pl = Lösungen ,
6345 variant = G ,
6346     Name-sg = Lösung ,
6347     Name-pl = Lösungen ,
6348 </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) and the `fr.comp.text.tex` (at <https://groups.google.com/g/fr.comp.text.tex/c/Fa11Tf6MFFs>) mailing lists.

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

```
6349 (*package)
6350 \zcDeclareLanguage [ gender = { f , m } ] { french }
6351 \zcDeclareLanguageAlias { acadian } { french }
6352 \endpackage

6353 (*lang-french)

6354 namesep = {\nobreakspace} ,
6355 pairsep = {\et\nobreakspace} ,
6356 listsep = {,~} ,
6357 lastsep = {\et\nobreakspace} ,
6358 tpairsep = {\et\nobreakspace} ,
6359 tlistsep = {,~} ,
6360 tlastsep = {\et\nobreakspace} ,
6361 notesep = {\} ,
6362 rangesep = {\à\nobreakspace} ,
6363
6364 type = book ,
6365   gender = m ,
6366   Name-sg = Livre ,
6367   name-sg = livre ,
6368   Name-pl = Livres ,
6369   name-pl = livres ,
6370
6371 type = part ,
6372   gender = f ,
6373   Name-sg = Partie ,
6374   name-sg = partie ,
6375   Name-pl = Parties ,
6376   name-pl = parties ,
6377
6378 type = chapter ,
6379   gender = m ,
6380   Name-sg = Chapitre ,
6381   name-sg = chapitre ,
6382   Name-pl = Chapitres ,
6383   name-pl = chapitres ,
6384
6385 type = section ,
6386   gender = f ,
6387   Name-sg = Section ,
6388   name-sg = section ,
6389   Name-pl = Sections ,
6390   name-pl = sections ,
6391
6392 type = paragraph ,
6393   gender = m ,
6394   Name-sg = Paragraphe ,
6395   name-sg = paragraphe ,
6396   Name-pl = Paragraphes ,
6397   name-pl = paragraphes ,
6398
6399 type = appendix ,
```

```

6400     gender = f ,
6401     Name-sg = Annexe ,
6402     name-sg = annexe ,
6403     Name-pl = Annexes ,
6404     name-pl = annexes ,
6405
6406 type = page ,
6407     gender = f ,
6408     Name-sg = Page ,
6409     name-sg = page ,
6410     Name-pl = Pages ,
6411     name-pl = pages ,
6412     rangesep = {-} ,
6413     rangetopair = false ,
6414
6415 type = line ,
6416     gender = f ,
6417     Name-sg = Ligne ,
6418     name-sg = ligne ,
6419     Name-pl = Lignes ,
6420     name-pl = lignes ,
6421
6422 type = figure ,
6423     gender = f ,
6424     Name-sg = Figure ,
6425     name-sg = figure ,
6426     Name-pl = Figures ,
6427     name-pl = figures ,
6428
6429 type = table ,
6430     gender = f ,
6431     Name-sg = Table ,
6432     name-sg = table ,
6433     Name-pl = Tables ,
6434     name-pl = tables ,
6435
6436 type = item ,
6437     gender = m ,
6438     Name-sg = Point ,
6439     name-sg = point ,
6440     Name-pl = Points ,
6441     name-pl = points ,
6442
6443 type = footnote ,
6444     gender = f ,
6445     Name-sg = Note ,
6446     name-sg = note ,
6447     Name-pl = Notes ,
6448     name-pl = notes ,
6449
6450 type = endnote ,
6451     gender = f ,
6452     Name-sg = Note ,
6453     name-sg = note ,

```

```

6454 Name-pl = Notes ,
6455 name-pl = notes ,
6456
6457 type = note ,
6458 gender = f ,
6459 Name-sg = Note ,
6460 name-sg = note ,
6461 Name-pl = Notes ,
6462 name-pl = notes ,
6463
6464 type = equation ,
6465 gender = f ,
6466 Name-sg = Équation ,
6467 name-sg = équation ,
6468 Name-pl = Équations ,
6469 name-pl = équations ,
6470 refbounds-first-sg = {,(,)}, ,
6471 refbounds = {(,,)} ,
6472
6473 type = theorem ,
6474 gender = m ,
6475 Name-sg = Théorème ,
6476 name-sg = théorème ,
6477 Name-pl = Théorèmes ,
6478 name-pl = théorèmes ,
6479
6480 type = lemma ,
6481 gender = m ,
6482 Name-sg = Lemme ,
6483 name-sg = lemme ,
6484 Name-pl = Lemmes ,
6485 name-pl = lemmes ,
6486
6487 type = corollary ,
6488 gender = m ,
6489 Name-sg = Corollaire ,
6490 name-sg = corollaire ,
6491 Name-pl = Corollaires ,
6492 name-pl = corollaires ,
6493
6494 type = proposition ,
6495 gender = f ,
6496 Name-sg = Proposition ,
6497 name-sg = proposition ,
6498 Name-pl = Propositions ,
6499 name-pl = propositions ,
6500
6501 type = definition ,
6502 gender = f ,
6503 Name-sg = Définition ,
6504 name-sg = définition ,
6505 Name-pl = Définitions ,
6506 name-pl = définitions ,
6507

```

```

6508 type = proof ,
6509     gender = f ,
6510     Name-sg = Démonstration ,
6511     name-sg = démonstration ,
6512     Name-pl = Démonstrations ,
6513     name-pl = démonstrations ,
6514
6515 type = result ,
6516     gender = m ,
6517     Name-sg = Résultat ,
6518     name-sg = résultat ,
6519     Name-pl = Résultats ,
6520     name-pl = résultats ,
6521
6522 type = remark ,
6523     gender = f ,
6524     Name-sg = Remarque ,
6525     name-sg = remarque ,
6526     Name-pl = Remarques ,
6527     name-pl = remarques ,
6528
6529 type = example ,
6530     gender = m ,
6531     Name-sg = Exemple ,
6532     name-sg = exemple ,
6533     Name-pl = Exemples ,
6534     name-pl = exemples ,
6535
6536 type = algorithm ,
6537     gender = m ,
6538     Name-sg = Algorithme ,
6539     name-sg = algorithme ,
6540     Name-pl = Algorithmes ,
6541     name-pl = algorithmes ,
6542
6543 type = listing ,
6544     gender = m ,
6545     Name-sg = Listing ,
6546     name-sg = listing ,
6547     Name-pl = Listings ,
6548     name-pl = listings ,
6549
6550 type = exercise ,
6551     gender = m ,
6552     Name-sg = Exercice ,
6553     name-sg = exercice ,
6554     Name-pl = Exercices ,
6555     name-pl = exercices ,
6556
6557 type = solution ,
6558     gender = f ,
6559     Name-sg = Solution ,
6560     name-sg = solution ,
6561     Name-pl = Solutions ,

```

```

6562 name-pl = solutions ,
6563 </lang-french>

```

10.5 Portuguese

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

```

6564 <*package>
6565 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6566 \zcDeclareLanguageAlias { brazilian } { portuguese }
6567 \zcDeclareLanguageAlias { brazil } { portuguese }
6568 \zcDeclareLanguageAlias { portuges } { portuguese }
6569 </package>

6570 <*lang-portuguese>

6571 namesep = {\nobreakspace} ,
6572 pairsep = {\~e\nobreakspace} ,
6573 listsep = { ,\~ } ,
6574 lastsep = {\~e\nobreakspace} ,
6575 tpairsep = {\~e\nobreakspace} ,
6576 tlistsep = { ,\~ } ,
6577 tlastsep = {\~e\nobreakspace} ,
6578 notesep = { \~ } ,
6579 rangesep = {\~a\nobreakspace} ,
6580
6581 type = book ,
6582 gender = m ,
6583 Name-sg = Livro ,
6584 name-sg = livro ,
6585 Name-pl = Livros ,
6586 name-pl = livros ,
6587
6588 type = part ,
6589 gender = f ,
6590 Name-sg = Parte ,
6591 name-sg = parte ,
6592 Name-pl = Partes ,
6593 name-pl = partes ,
6594
6595 type = chapter ,
6596 gender = m ,
6597 Name-sg = Capítulo ,
6598 name-sg = capítulo ,
6599 Name-pl = Capítulos ,
6600 name-pl = capítulos ,
6601
6602 type = section ,
6603 gender = f ,
6604 Name-sg = Seção ,
6605 name-sg = seção ,
6606 Name-pl = Seções ,
6607 name-pl = seções ,

```

```

6608
6609 type = paragraph ,
6610     gender = m ,
6611     Name-sg = Parágrafo ,
6612     name-sg = parágrafo ,
6613     Name-pl = Parágrafos ,
6614     name-pl = parágrafos ,
6615     Name-sg-ab = Par. ,
6616     name-sg-ab = par. ,
6617     Name-pl-ab = Par. ,
6618     name-pl-ab = par. ,
6619
6620 type = appendix ,
6621     gender = m ,
6622     Name-sg = Apêndice ,
6623     name-sg = apêndice ,
6624     Name-pl = Apêndices ,
6625     name-pl = apêndices ,
6626
6627 type = page ,
6628     gender = f ,
6629     Name-sg = Página ,
6630     name-sg = página ,
6631     Name-pl = Páginas ,
6632     name-pl = páginas ,
6633     rangeseq = {\textendash} ,
6634     rangetopair = false ,
6635
6636 type = line ,
6637     gender = f ,
6638     Name-sg = Linha ,
6639     name-sg = linha ,
6640     Name-pl = Linhas ,
6641     name-pl = linhas ,
6642
6643 type = figure ,
6644     gender = f ,
6645     Name-sg = Figura ,
6646     name-sg = figura ,
6647     Name-pl = Figuras ,
6648     name-pl = figuras ,
6649     Name-sg-ab = Fig. ,
6650     name-sg-ab = fig. ,
6651     Name-pl-ab = Figs. ,
6652     name-pl-ab = figs. ,
6653
6654 type = table ,
6655     gender = f ,
6656     Name-sg = Tabela ,
6657     name-sg = tabela ,
6658     Name-pl = Tabelas ,
6659     name-pl = tabelas ,
6660
6661 type = item ,

```

```

6662 gender = m ,
6663 Name-sg = Item ,
6664 name-sg = item ,
6665 Name-pl = Itens ,
6666 name-pl = itens ,
6667
6668 type = footnote ,
6669 gender = f ,
6670 Name-sg = Nota ,
6671 name-sg = nota ,
6672 Name-pl = Notas ,
6673 name-pl = notas ,
6674
6675 type = endnote ,
6676 gender = f ,
6677 Name-sg = Nota ,
6678 name-sg = nota ,
6679 Name-pl = Notas ,
6680 name-pl = notas ,
6681
6682 type = note ,
6683 gender = f ,
6684 Name-sg = Nota ,
6685 name-sg = nota ,
6686 Name-pl = Notas ,
6687 name-pl = notas ,
6688
6689 type = equation ,
6690 gender = f ,
6691 Name-sg = Equação ,
6692 name-sg = equação ,
6693 Name-pl = Equações ,
6694 name-pl = equações ,
6695 Name-sg-ab = Eq. ,
6696 name-sg-ab = eq. ,
6697 Name-pl-ab = Eqs. ,
6698 name-pl-ab = eqs. ,
6699 refbounds-first-sg = {,(,)} ,
6700 refbounds = {(,,)} ,
6701
6702 type = theorem ,
6703 gender = m ,
6704 Name-sg = Teorema ,
6705 name-sg = teorema ,
6706 Name-pl = Teoremas ,
6707 name-pl = teoremas ,
6708
6709 type = lemma ,
6710 gender = m ,
6711 Name-sg = Lema ,
6712 name-sg = lema ,
6713 Name-pl = Lemas ,
6714 name-pl = lemas ,
6715

```



```

6716 type = corollary ,
6717     gender = m ,
6718     Name-sg = Corolário ,
6719     name-sg = corolário ,
6720     Name-pl = Corolários ,
6721     name-pl = corolários ,
6722
6723 type = proposition ,
6724     gender = f ,
6725     Name-sg = Proposição ,
6726     name-sg = proposição ,
6727     Name-pl = Proposições ,
6728     name-pl = proposições ,
6729
6730 type = definition ,
6731     gender = f ,
6732     Name-sg = Definição ,
6733     name-sg = definição ,
6734     Name-pl = Definições ,
6735     name-pl = definições ,
6736
6737 type = proof ,
6738     gender = f ,
6739     Name-sg = Demonstração ,
6740     name-sg = demonstração ,
6741     Name-pl = Demonstrações ,
6742     name-pl = demonstrações ,
6743
6744 type = result ,
6745     gender = m ,
6746     Name-sg = Resultado ,
6747     name-sg = resultado ,
6748     Name-pl = Resultados ,
6749     name-pl = resultados ,
6750
6751 type = remark ,
6752     gender = f ,
6753     Name-sg = Observação ,
6754     name-sg = observação ,
6755     Name-pl = Observações ,
6756     name-pl = observações ,
6757
6758 type = example ,
6759     gender = m ,
6760     Name-sg = Exemplo ,
6761     name-sg = exemplo ,
6762     Name-pl = Exemplos ,
6763     name-pl = exemplos ,
6764
6765 type = algorithm ,
6766     gender = m ,
6767     Name-sg = Algoritmo ,
6768     name-sg = algoritmo ,
6769     Name-pl = Algoritmos ,

```

```

6770 name-pl = algoritmos ,
6771
6772 type = listing ,
6773 gender = f ,
6774 Name-sg = Listagem ,
6775 name-sg = listagem ,
6776 Name-pl = Listagens ,
6777 name-pl = listagens ,
6778
6779 type = exercise ,
6780 gender = m ,
6781 Name-sg = Exercício ,
6782 name-sg = exercício ,
6783 Name-pl = Exercícios ,
6784 name-pl = exercícios ,
6785
6786 type = solution ,
6787 gender = f ,
6788 Name-sg = Solução ,
6789 name-sg = solução ,
6790 Name-pl = Soluções ,
6791 name-pl = soluções ,
6792 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```

6793 <*package>
6794 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6795 </package>
6796 <*lang-spanish>
6797 namesep = {\nobreakspace} ,
6798 pairsep = {\~y\nobreakspace} ,
6799 listsep = {,~} ,
6800 lastsep = {\~y\nobreakspace} ,
6801 tpairsep = {\~y\nobreakspace} ,
6802 tlistsep = {,~} ,
6803 tlastsep = {\~y\nobreakspace} ,
6804 notesep = {\~} ,
6805 rangesep = {\~a\nobreakspace} ,
6806
6807 type = book ,
6808 gender = m ,
6809 Name-sg = Libro ,
6810 name-sg = libro ,
6811 Name-pl = Libros ,
6812 name-pl = libros ,
6813
6814 type = part ,
6815 gender = f ,
6816 Name-sg = Parte ,
6817 name-sg = parte ,

```

```

6818 Name-pl = Partes ,
6819 name-pl = partes ,
6820
6821 type = chapter ,
6822 gender = m ,
6823 Name-sg = Capítulo ,
6824 name-sg = capítulo ,
6825 Name-pl = Capítulos ,
6826 name-pl = capítulos ,
6827
6828 type = section ,
6829 gender = f ,
6830 Name-sg = Sección ,
6831 name-sg = sección ,
6832 Name-pl = Secciones ,
6833 name-pl = secciones ,
6834
6835 type = paragraph ,
6836 gender = m ,
6837 Name-sg = Párrafo ,
6838 name-sg = párrafo ,
6839 Name-pl = Párrafos ,
6840 name-pl = párrafos ,
6841
6842 type = appendix ,
6843 gender = m ,
6844 Name-sg = Apéndice ,
6845 name-sg = apéndice ,
6846 Name-pl = Apéndices ,
6847 name-pl = apéndices ,
6848
6849 type = page ,
6850 gender = f ,
6851 Name-sg = Página ,
6852 name-sg = página ,
6853 Name-pl = Páginas ,
6854 name-pl = páginas ,
6855 rangsep = {\textendash} ,
6856 rangetopair = false ,
6857
6858 type = line ,
6859 gender = f ,
6860 Name-sg = Línea ,
6861 name-sg = línea ,
6862 Name-pl = Líneas ,
6863 name-pl = líneas ,
6864
6865 type = figure ,
6866 gender = f ,
6867 Name-sg = Figura ,
6868 name-sg = figura ,
6869 Name-pl = Figuras ,
6870 name-pl = figuras ,
6871

```

```

6872 type = table ,
6873     gender = m ,
6874     Name-sg = Cuadro ,
6875     name-sg = cuadro ,
6876     Name-pl = Cuadros ,
6877     name-pl = cuadros ,
6878
6879 type = item ,
6880     gender = m ,
6881     Name-sg = Punto ,
6882     name-sg = punto ,
6883     Name-pl = Puntos ,
6884     name-pl = puntos ,
6885
6886 type = footnote ,
6887     gender = f ,
6888     Name-sg = Nota ,
6889     name-sg = nota ,
6890     Name-pl = Notas ,
6891     name-pl = notas ,
6892
6893 type = endnote ,
6894     gender = f ,
6895     Name-sg = Nota ,
6896     name-sg = nota ,
6897     Name-pl = Notas ,
6898     name-pl = notas ,
6899
6900 type = note ,
6901     gender = f ,
6902     Name-sg = Nota ,
6903     name-sg = nota ,
6904     Name-pl = Notas ,
6905     name-pl = notas ,
6906
6907 type = equation ,
6908     gender = f ,
6909     Name-sg = Ecuación ,
6910     name-sg = ecuación ,
6911     Name-pl = Ecuaciones ,
6912     name-pl = ecuaciones ,
6913     refbounds-first-sg = {,(,)}, ,
6914     refbounds = {(,,)} ,
6915
6916 type = theorem ,
6917     gender = m ,
6918     Name-sg = Teorema ,
6919     name-sg = teorema ,
6920     Name-pl = Teoremas ,
6921     name-pl = teoremas ,
6922
6923 type = lemma ,
6924     gender = m ,
6925     Name-sg = Lema ,

```

```

6926 name-sg = lema ,
6927 Name-pl = Lemas ,
6928 name-pl = lemas ,
6929
6930 type = corollary ,
6931 gender = m ,
6932 Name-sg = Corolario ,
6933 name-sg = corolario ,
6934 Name-pl = Corolarios ,
6935 name-pl = corolarios ,
6936
6937 type = proposition ,
6938 gender = f ,
6939 Name-sg = Proposición ,
6940 name-sg = proposición ,
6941 Name-pl = Proposiciones ,
6942 name-pl = proposiciones ,
6943
6944 type = definition ,
6945 gender = f ,
6946 Name-sg = Definición ,
6947 name-sg = definición ,
6948 Name-pl = Definiciones ,
6949 name-pl = definiciones ,
6950
6951 type = proof ,
6952 gender = f ,
6953 Name-sg = Demostración ,
6954 name-sg = demostración ,
6955 Name-pl = Demostraciones ,
6956 name-pl = demostraciones ,
6957
6958 type = result ,
6959 gender = m ,
6960 Name-sg = Resultado ,
6961 name-sg = resultado ,
6962 Name-pl = Resultados ,
6963 name-pl = resultados ,
6964
6965 type = remark ,
6966 gender = f ,
6967 Name-sg = Observación ,
6968 name-sg = observación ,
6969 Name-pl = Observaciones ,
6970 name-pl = observaciones ,
6971
6972 type = example ,
6973 gender = m ,
6974 Name-sg = Ejemplo ,
6975 name-sg = ejemplo ,
6976 Name-pl = Ejemplos ,
6977 name-pl = ejemplos ,
6978
6979 type = algorithm ,

```

```

6980   gender = m ,
6981   Name-sg = Algoritmo ,
6982   name-sg = algoritmo ,
6983   Name-pl = Algoritmos ,
6984   name-pl = algoritmos ,
6985
6986   type = listing ,
6987   gender = m ,
6988   Name-sg = Listado ,
6989   name-sg = listado ,
6990   Name-pl = Listados ,
6991   name-pl = listados ,
6992
6993   type = exercise ,
6994   gender = m ,
6995   Name-sg = Ejercicio ,
6996   name-sg = ejercicio ,
6997   Name-pl = Ejercicios ,
6998   name-pl = ejercicios ,
6999
7000   type = solution ,
7001   gender = f ,
7002   Name-sg = Solución ,
7003   name-sg = solución ,
7004   Name-pl = Soluciones ,
7005   name-pl = soluciones ,
7006 </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.

```

7007 <*package>
7008 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7009 </package>
7010 <*lang-dutch>
7011 namesep = {\nobreakspace} ,
7012 pairsep = {\~en\nobreakspace} ,
7013 listsep = { , ~ } ,
7014 lastsep = {\~en\nobreakspace} ,
7015 tpairsep = {\~en\nobreakspace} ,
7016 tlistsep = { , ~ } ,
7017 tlastsep = { , ~en\nobreakspace} ,
7018 notesep = { ~ } ,
7019 rangesep = {\~t/m\nobreakspace} ,
7020
7021 type = book ,
7022   gender = n ,
7023   Name-sg = Boek ,
7024   name-sg = boek ,
7025   Name-pl = Boeken ,
7026   name-pl = boeken ,

```

```

7027
7028 type = part ,
7029     gender = n ,
7030     Name-sg = Deel ,
7031     name-sg = deel ,
7032     Name-pl = Delen ,
7033     name-pl = delen ,
7034
7035 type = chapter ,
7036     gender = n ,
7037     Name-sg = Hoofdstuk ,
7038     name-sg = hoofdstuk ,
7039     Name-pl = Hoofdstukken ,
7040     name-pl = hoofdstukken ,
7041
7042 type = section ,
7043     gender = m ,
7044     Name-sg = Paragraaf ,
7045     name-sg = paragraaf ,
7046     Name-pl = Paragrafen ,
7047     name-pl = paragrafen ,
7048
7049 type = paragraph ,
7050     gender = f ,
7051     Name-sg = Alinea ,
7052     name-sg = alinea ,
7053     Name-pl = Alinea's ,
7054     name-pl = alinea's ,
7055

```

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.

```

7056 type = appendix ,
7057     gender = { f , m } ,
7058     Name-sg = Bijlage ,
7059     name-sg = bijlage ,
7060     Name-pl = Bijlagen ,
7061     name-pl = bijlagen ,
7062
7063 type = page ,
7064     gender = { f , m } ,
7065     Name-sg = Pagina ,
7066     name-sg = pagina ,
7067     Name-pl = Pagina's ,
7068     name-pl = pagina's ,
7069     rangesep = {\textendash} ,
7070     rangetopair = false ,
7071
7072 type = line ,
7073     gender = m ,
7074     Name-sg = Regel ,
7075     name-sg = regel ,
7076     Name-pl = Regels ,

```

```

7077 name-pl = regels ,
7078
7079 type = figure ,
7080 gender = { n , f , m } ,
7081 Name-sg = Figuur ,
7082 name-sg = figuur ,
7083 Name-pl = Figuren ,
7084 name-pl = figuren ,
7085
7086 type = table ,
7087 gender = { f , m } ,
7088 Name-sg = Tabel ,
7089 name-sg = tabel ,
7090 Name-pl = Tabellen ,
7091 name-pl = tabellen ,
7092
7093 type = item ,
7094 gender = n ,
7095 Name-sg = Punt ,
7096 name-sg = punt ,
7097 Name-pl = Punten ,
7098 name-pl = punten ,
7099
7100 type = footnote ,
7101 gender = { f , m } ,
7102 Name-sg = Voetnoot ,
7103 name-sg = voetnoot ,
7104 Name-pl = Voetnoten ,
7105 name-pl = voetnoten ,
7106
7107 type = endnote ,
7108 gender = { f , m } ,
7109 Name-sg = Eindnoot ,
7110 name-sg = eindnoot ,
7111 Name-pl = Eindnoten ,
7112 name-pl = eindnoten ,
7113
7114 type = note ,
7115 gender = f ,
7116 Name-sg = Opmerking ,
7117 name-sg = opmerking ,
7118 Name-pl = Opmerkingen ,
7119 name-pl = opmerkingen ,
7120
7121 type = equation ,
7122 gender = f ,
7123 Name-sg = Vergelijking ,
7124 name-sg = vergelijking ,
7125 Name-pl = Vergelijkingen ,
7126 name-pl = vergelijkingen ,
7127 Name-sg-ab = Vgl. ,
7128 name-sg-ab = vgl. ,
7129 Name-pl-ab = Vgl.'s ,
7130 name-pl-ab = vgl.'s ,

```



```

7131   refbounds-first-sg = {,(,)},
7132   refbounds = {(,,)} ,
7133
7134   type = theorem ,
7135   gender = f ,
7136   Name-sg = Stelling ,
7137   name-sg = stelling ,
7138   Name-pl = Stellingen ,
7139   name-pl = stellingen ,
7140

```

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".

```

7141   type = lemma ,
7142   gender = n ,
7143   Name-sg = Lemma ,
7144   name-sg = lemma ,
7145   Name-pl = Lemma's ,
7146   name-pl = lemma's ,
7147
7148   type = corollary ,
7149   gender = n ,
7150   Name-sg = Gevolg ,
7151   name-sg = gevolg ,
7152   Name-pl = Gevolgen ,
7153   name-pl = gevolgen ,
7154
7155   type = proposition ,
7156   gender = f ,
7157   Name-sg = Propositie ,
7158   name-sg = propositie ,
7159   Name-pl = Propositions ,
7160   name-pl = propositions ,
7161
7162   type = definition ,
7163   gender = f ,
7164   Name-sg = Definitie ,
7165   name-sg = definitie ,
7166   Name-pl = Definities ,
7167   name-pl = definities ,
7168
7169   type = proof ,
7170   gender = n ,
7171   Name-sg = Bewijs ,
7172   name-sg = bewijs ,
7173   Name-pl = Bewijzen ,
7174   name-pl = bewijzen ,
7175
7176   type = result ,
7177   gender = n ,
7178   Name-sg = Resultaat ,
7179   name-sg = resultaat ,
7180   Name-pl = Resultaten ,

```

```

7181   name-pl = resultaten ,
7182
7183   type = remark ,
7184   gender = f ,
7185   Name-sg = Opmerking ,
7186   name-sg = opmerking ,
7187   Name-pl = Opmerkingen ,
7188   name-pl = opmerkingen ,
7189
7190   type = example ,
7191   gender = n ,
7192   Name-sg = Voorbeeld ,
7193   name-sg = voorbeeld ,
7194   Name-pl = Voorbeelden ,
7195   name-pl = voorbeelden ,
7196

```

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".

```

7197   type = algorithm ,
7198   gender = { n , f , m } ,
7199   Name-sg = Algoritme ,
7200   name-sg = algoritme ,
7201   Name-pl = Algoritmen ,
7202   name-pl = algoritmen ,
7203

```

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

```

7204   type = listing ,
7205   gender = m ,
7206   Name-sg = Listing ,
7207   name-sg = listing ,
7208   Name-pl = Listings ,
7209   name-pl = listings ,
7210
7211   type = exercise ,
7212   gender = { f , m } ,
7213   Name-sg = Opgave ,
7214   name-sg = opgave ,
7215   Name-pl = Opgaven ,
7216   name-pl = opgaven ,
7217
7218   type = solution ,
7219   gender = f ,
7220   Name-sg = Oplossing ,
7221   name-sg = oplossing ,
7222   Name-pl = Oplossingen ,
7223   name-pl = oplossingen ,
7224 </lang-dutch>

```

10.8 Italian

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

```
7225 (*package)
7226 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7227 \endpackage

7228 (*lang-italian)

7229 namesep = {\nobreakspace} ,
7230 pairsep = {\nobreakspace} ,
7231 listsep = {,~} ,
7232 lastsep = {\nobreakspace} ,
7233 tpairsep = {\nobreakspace} ,
7234 tlistsep = {,~} ,
7235 tlastsep = {,~\nobreakspace} ,
7236 notesep = {~} ,
7237 rangesep = {\nobreakspace} ,
7238 +refbounds-rb = {da\nobreakspace,,} ,
7239
7240 type = book ,
7241   gender = m ,
7242   Name-sg = Libro ,
7243   name-sg = libro ,
7244   Name-pl = Libri ,
7245   name-pl = libri ,
7246
7247 type = part ,
7248   gender = f ,
7249   Name-sg = Parte ,
7250   name-sg = parte ,
7251   Name-pl = Parti ,
7252   name-pl = parti ,
7253
7254 type = chapter ,
7255   gender = m ,
7256   Name-sg = Capitolo ,
7257   name-sg = capitolo ,
7258   Name-pl = Capitoli ,
7259   name-pl = capitoli ,
7260
7261 type = section ,
7262   gender = m ,
7263   Name-sg = Paragrafo ,
7264   name-sg = paragrafo ,
7265   Name-pl = Paragrafi ,
7266   name-pl = paragrafi ,
7267
7268 type = paragraph ,
7269   gender = m ,
7270   Name-sg = Capoverso ,
7271   name-sg = capoverso ,
7272   Name-pl = Capoversi ,
```

```

7273 name-pl = capoversi ,
7274
7275 type = appendix ,
7276 gender = f ,
7277 Name-sg = Appendice ,
7278 name-sg = appendice ,
7279 Name-pl = Appendici ,
7280 name-pl = appendici ,
7281
7282 type = page ,
7283 gender = f ,
7284 Name-sg = Pagina ,
7285 name-sg = pagina ,
7286 Name-pl = Pagine ,
7287 name-pl = pagine ,
7288 Name-sg-ab = Pag. ,
7289 name-sg-ab = pag. ,
7290 Name-pl-ab = Pag. ,
7291 name-pl-ab = pag. ,
7292 rangesep = {\textendash} ,
7293 rangetopair = false ,
7294 +refbounds-rb = {,,} ,
7295
7296 type = line ,
7297 gender = f ,
7298 Name-sg = Riga ,
7299 name-sg = riga ,
7300 Name-pl = Righe ,
7301 name-pl = righe ,
7302
7303 type = figure ,
7304 gender = f ,
7305 Name-sg = Figura ,
7306 name-sg = figura ,
7307 Name-pl = Figure ,
7308 name-pl = figure ,
7309 Name-sg-ab = Fig. ,
7310 name-sg-ab = fig. ,
7311 Name-pl-ab = Fig. ,
7312 name-pl-ab = fig. ,
7313
7314 type = table ,
7315 gender = f ,
7316 Name-sg = Tabella ,
7317 name-sg = tabella ,
7318 Name-pl = Tabelle ,
7319 name-pl = tabelle ,
7320 Name-sg-ab = Tab. ,
7321 name-sg-ab = tab. ,
7322 Name-pl-ab = Tab. ,
7323 name-pl-ab = tab. ,
7324
7325 type = item ,
7326 gender = m ,

```

```

7327 Name-sg = Punto ,
7328 name-sg = punto ,
7329 Name-pl = Punti ,
7330 name-pl = punti ,
7331
7332 type = footnote ,
7333 gender = f ,
7334 Name-sg = Nota ,
7335 name-sg = nota ,
7336 Name-pl = Note ,
7337 name-pl = note ,
7338
7339 type = endnote ,
7340 gender = f ,
7341 Name-sg = Nota ,
7342 name-sg = nota ,
7343 Name-pl = Note ,
7344 name-pl = note ,
7345
7346 type = note ,
7347 gender = f ,
7348 Name-sg = Nota ,
7349 name-sg = nota ,
7350 Name-pl = Note ,
7351 name-pl = note ,
7352
7353 type = equation ,
7354 gender = f ,
7355 Name-sg = Equazione ,
7356 name-sg = equazione ,
7357 Name-pl = Equazioni ,
7358 name-pl = equazioni ,
7359 Name-sg-ab = Eq. ,
7360 name-sg-ab = eq. ,
7361 Name-pl-ab = Eq. ,
7362 name-pl-ab = eq. ,
7363 +refbounds-rb = {da\nobreakspace(,,)} ,
7364 refbounds-first-sg = {(,(),)} ,
7365 refbounds = {(,,)} ,
7366
7367 type = theorem ,
7368 gender = m ,
7369 Name-sg = Teorema ,
7370 name-sg = teorema ,
7371 Name-pl = Teoremi ,
7372 name-pl = teoremi ,
7373
7374 type = lemma ,
7375 gender = m ,
7376 Name-sg = Lemma ,
7377 name-sg = lemma ,
7378 Name-pl = Lemmi ,
7379 name-pl = lemmi ,
7380

```

```

7381 type = corollary ,
7382   gender = m ,
7383   Name-sg = Corollario ,
7384   name-sg = corollario ,
7385   Name-pl = Corollari ,
7386   name-pl = corollari ,
7387
7388 type = proposition ,
7389   gender = f ,
7390   Name-sg = Proposizione ,
7391   name-sg = proposizione ,
7392   Name-pl = Proposizioni ,
7393   name-pl = proposizioni ,
7394
7395 type = definition ,
7396   gender = f ,
7397   Name-sg = Definizione ,
7398   name-sg = definizione ,
7399   Name-pl = Definizioni ,
7400   name-pl = definizioni ,
7401
7402 type = proof ,
7403   gender = f ,
7404   Name-sg = Dimostrazione ,
7405   name-sg = dimostrazione ,
7406   Name-pl = Dimostrazioni ,
7407   name-pl = dimostrazioni ,
7408
7409 type = result ,
7410   gender = m ,
7411   Name-sg = Risultato ,
7412   name-sg = risultato ,
7413   Name-pl = Risultati ,
7414   name-pl = risultati ,
7415
7416 type = remark ,
7417   gender = f ,
7418   Name-sg = Osservazione ,
7419   name-sg = osservazione ,
7420   Name-pl = Osservazioni ,
7421   name-pl = osservazioni ,
7422
7423 type = example ,
7424   gender = m ,
7425   Name-sg = Esempio ,
7426   name-sg = esempio ,
7427   Name-pl = Esempi ,
7428   name-pl = esempi ,
7429
7430 type = algorithm ,
7431   gender = m ,
7432   Name-sg = Algoritmo ,
7433   name-sg = algoritmo ,
7434   Name-pl = Algoritmi ,

```

```

7435   name-pl = algoritmi ,
7436
7437   type = listing ,
7438   gender = m ,
7439   Name-sg = Listato ,
7440   name-sg = listato ,
7441   Name-pl = Listati ,
7442   name-pl = listati ,
7443
7444   type = exercise ,
7445   gender = m ,
7446   Name-sg = Esercizio ,
7447   name-sg = esercizio ,
7448   Name-pl = Esercizi ,
7449   name-pl = esercizi ,
7450
7451   type = solution ,
7452   gender = f ,
7453   Name-sg = Soluzione ,
7454   name-sg = soluzione ,
7455   Name-pl = Soluzioni ,
7456   name-pl = soluzioni ,
7457 </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.”.

```

7458 (*package)
7459 \zcDeclareLanguage
7460 [ variants = { n , a , g , d , i , p } , gender = { f , m , n } ]
7461 { russian }
7462 </package>
7463 (*lang-russian)
7464 namesep = {\nobreakspace} ,
7465 pairsep = {\~\nobreakspace} ,
7466 listsep = { , ~ } ,
7467 lastsep = {\~\nobreakspace} ,
7468 tpairsep = {\~\nobreakspace} ,
7469 tlistsep = { , ~ } ,
7470 tlastsep = { , ~\nobreakspace} ,
7471 notesep = { ~ } ,
7472 rangesep = { ~по\nobreakspace} ,
7473 +refbounds-rb = {c\nobreakspace,,,} ,
7474
7475 type = book ,
7476 gender = f ,
7477 variant = n ,
7478 Name-sg = Книга ,

```

7479 name-sg = книга ,
7480 Name-pl = Книги ,
7481 name-pl = книги ,
7482 variant = a ,
7483 Name-sg = Книгу ,
7484 name-sg = книгу ,
7485 Name-pl = Книги ,
7486 name-pl = книги ,
7487 variant = g ,
7488 Name-sg = Книги ,
7489 name-sg = книги ,
7490 Name-pl = Книг ,
7491 name-pl = книг ,
7492 variant = d ,
7493 Name-sg = Книге ,
7494 name-sg = книге ,
7495 Name-pl = Книгам ,
7496 name-pl = книгам ,
7497 variant = i ,
7498 Name-sg = Книгой ,
7499 name-sg = книгой ,
7500 Name-pl = Книгами ,
7501 name-pl = книгами ,
7502 variant = p ,
7503 Name-sg = Книге ,
7504 name-sg = книге ,
7505 Name-pl = Книгах ,
7506 name-pl = книгах ,
7507
7508 type = part ,
7509 gender = f ,
7510 variant = n ,
7511 Name-sg = Часть ,
7512 name-sg = часть ,
7513 Name-pl = Части ,
7514 name-pl = части ,
7515 Name-sg-ab = Ч. ,
7516 name-sg-ab = ч. ,
7517 Name-pl-ab = Чч. ,
7518 name-pl-ab = чч. ,
7519 variant = a ,
7520 Name-sg = Часть ,
7521 name-sg = часть ,
7522 Name-pl = Части ,
7523 name-pl = части ,
7524 Name-sg-ab = Ч. ,
7525 name-sg-ab = ч. ,
7526 Name-pl-ab = Чч. ,
7527 name-pl-ab = чч. ,
7528 variant = g ,
7529 Name-sg = Части ,
7530 name-sg = части ,
7531 Name-pl = Частей ,
7532 name-pl = частей ,

7533 Name-sg-ab = Ч. ,
7534 name-sg-ab = ч. ,
7535 Name-pl-ab = Чч. ,
7536 name-pl-ab = чч. ,
7537 variant = d ,
7538 Name-sg = Части ,
7539 name-sg = части ,
7540 Name-pl = Частям ,
7541 name-pl = частям ,
7542 Name-sg-ab = Ч. ,
7543 name-sg-ab = ч. ,
7544 Name-pl-ab = Чч. ,
7545 name-pl-ab = чч. ,
7546 variant = i ,
7547 Name-sg = Частью ,
7548 name-sg = частью ,
7549 Name-pl = Частями ,
7550 name-pl = частями ,
7551 Name-sg-ab = Ч. ,
7552 name-sg-ab = ч. ,
7553 Name-pl-ab = Чч. ,
7554 name-pl-ab = чч. ,
7555 variant = p ,
7556 Name-sg = Части ,
7557 name-sg = части ,
7558 Name-pl = Частях ,
7559 name-pl = частях ,
7560 Name-sg-ab = Ч. ,
7561 name-sg-ab = ч. ,
7562 Name-pl-ab = Чч. ,
7563 name-pl-ab = чч. ,
7564
7565 type = chapter ,
7566 gender = f ,
7567 variant = n ,
7568 Name-sg = Глава ,
7569 name-sg = глава ,
7570 Name-pl = Главы ,
7571 name-pl = главы ,
7572 Name-sg-ab = Гл. ,
7573 name-sg-ab = гл. ,
7574 Name-pl-ab = Гл. ,
7575 name-pl-ab = гл. ,
7576 variant = a ,
7577 Name-sg = Главу ,
7578 name-sg = главу ,
7579 Name-pl = Главы ,
7580 name-pl = главы ,
7581 Name-sg-ab = Гл. ,
7582 name-sg-ab = гл. ,
7583 Name-pl-ab = Гл. ,
7584 name-pl-ab = гл. ,
7585 variant = g ,
7586 Name-sg = Главы ,

7587 name-sg = главы ,
7588 Name-pl = Глав ,
7589 name-pl = глав ,
7590 Name-sg-ab = Гл. ,
7591 name-sg-ab = гл. ,
7592 Name-pl-ab = Гл. ,
7593 name-pl-ab = гл. ,
7594 variant = d ,
7595 Name-sg = Главе ,
7596 name-sg = главе ,
7597 Name-pl = Главам ,
7598 name-pl = главам ,
7599 Name-sg-ab = Гл. ,
7600 name-sg-ab = гл. ,
7601 Name-pl-ab = Гл. ,
7602 name-pl-ab = гл. ,
7603 variant = i ,
7604 Name-sg = Главой ,
7605 name-sg = главой ,
7606 Name-pl = Главами ,
7607 name-pl = главами ,
7608 Name-sg-ab = Гл. ,
7609 name-sg-ab = гл. ,
7610 Name-pl-ab = Гл. ,
7611 name-pl-ab = гл. ,
7612 variant = p ,
7613 Name-sg = Главе ,
7614 name-sg = главе ,
7615 Name-pl = Главах ,
7616 name-pl = главах ,
7617 Name-sg-ab = Гл. ,
7618 name-sg-ab = гл. ,
7619 Name-pl-ab = Гл. ,
7620 name-pl-ab = гл. ,
7621
7622 type = section ,
7623 gender = m ,
7624 variant = n ,
7625 Name-sg = Раздел ,
7626 name-sg = раздел ,
7627 Name-pl = Разделы ,
7628 name-pl = разделы ,
7629 variant = a ,
7630 Name-sg = Раздел ,
7631 name-sg = раздел ,
7632 Name-pl = Разделы ,
7633 name-pl = разделы ,
7634 variant = g ,
7635 Name-sg = Раздела ,
7636 name-sg = раздела ,
7637 Name-pl = Разделов ,
7638 name-pl = разделов ,
7639 variant = d ,
7640 Name-sg = Разделу ,

7641 name-sg = разделу ,
7642 Name-pl = Разделам ,
7643 name-pl = разделам ,
7644 variant = i ,
7645 Name-sg = Разделом ,
7646 name-sg = разделом ,
7647 Name-pl = Разделами ,
7648 name-pl = разделами ,
7649 variant = p ,
7650 Name-sg = Разделе ,
7651 name-sg = разделе ,
7652 Name-pl = Разделах ,
7653 name-pl = разделах ,
7654
7655 type = paragraph ,
7656 gender = m ,
7657 variant = n ,
7658 Name-sg = Абзац ,
7659 name-sg = абзац ,
7660 Name-pl = Абзацы ,
7661 name-pl = абзацы ,
7662 variant = a ,
7663 Name-sg = Абзац ,
7664 name-sg = абзац ,
7665 Name-pl = Абзацы ,
7666 name-pl = абзацы ,
7667 variant = g ,
7668 Name-sg = Абзаца ,
7669 name-sg = абзаца ,
7670 Name-pl = Абзацев ,
7671 name-pl = абзацев ,
7672 variant = d ,
7673 Name-sg = Абзацу ,
7674 name-sg = абзацу ,
7675 Name-pl = Абзацам ,
7676 name-pl = абзацам ,
7677 variant = i ,
7678 Name-sg = Абзацем ,
7679 name-sg = абзацем ,
7680 Name-pl = Абзацами ,
7681 name-pl = абзацами ,
7682 variant = p ,
7683 Name-sg = Абзаце ,
7684 name-sg = абзаце ,
7685 Name-pl = Абзацах ,
7686 name-pl = абзацах ,
7687
7688 type = appendix ,
7689 gender = n ,
7690 variant = n ,
7691 Name-sg = Приложение ,
7692 name-sg = приложение ,
7693 Name-pl = Приложения ,
7694 name-pl = приложения ,

7695 variant = a ,
7696 Name-sg = Приложение ,
7697 name-sg = приложение ,
7698 Name-pl = Приложения ,
7699 name-pl = приложения ,
7700 variant = g ,
7701 Name-sg = Приложения ,
7702 name-sg = приложения ,
7703 Name-pl = Приложений ,
7704 name-pl = приложений ,
7705 variant = d ,
7706 Name-sg = Приложению ,
7707 name-sg = приложению ,
7708 Name-pl = Приложениям ,
7709 name-pl = приложениям ,
7710 variant = i ,
7711 Name-sg = Приложением ,
7712 name-sg = приложением ,
7713 Name-pl = Приложениями ,
7714 name-pl = приложениями ,
7715 variant = p ,
7716 Name-sg = Приложении ,
7717 name-sg = приложении ,
7718 Name-pl = Приложениях ,
7719 name-pl = приложениях ,
7720
7721 type = page ,
7722 gender = f ,
7723 variant = n ,
7724 Name-sg = Страница ,
7725 name-sg = страница ,
7726 Name-pl = Страницы ,
7727 name-pl = страницы ,
7728 Name-sg-ab = С. ,
7729 name-sg-ab = с. ,
7730 Name-pl-ab = Сс. ,
7731 name-pl-ab = сс. ,
7732 variant = a ,
7733 Name-sg = Страницу ,
7734 name-sg = страницу ,
7735 Name-pl = Страницы ,
7736 name-pl = страницы ,
7737 Name-sg-ab = С. ,
7738 name-sg-ab = с. ,
7739 Name-pl-ab = Сс. ,
7740 name-pl-ab = сс. ,
7741 variant = g ,
7742 Name-sg = Страницы ,
7743 name-sg = страницы ,
7744 Name-pl = Страниц ,
7745 name-pl = страниц ,
7746 Name-sg-ab = С. ,
7747 name-sg-ab = с. ,
7748 Name-pl-ab = Сс. ,

```

7749     name-pl-ab = сс. ,
7750     variant = d ,
7751     Name-sg = Странице ,
7752     name-sg = странице ,
7753     Name-pl = Страницам ,
7754     name-pl = страницам ,
7755     Name-sg-ab = С. ,
7756     name-sg-ab = с. ,
7757     Name-pl-ab = Сс. ,
7758     name-pl-ab = сс. ,
7759     variant = i ,
7760     Name-sg = Страницей ,
7761     name-sg = страницей ,
7762     Name-pl = Страницами ,
7763     name-pl = страницами ,
7764     Name-sg-ab = С. ,
7765     name-sg-ab = с. ,
7766     Name-pl-ab = Сс. ,
7767     name-pl-ab = сс. ,
7768     variant = p ,
7769     Name-sg = Странице ,
7770     name-sg = странице ,
7771     Name-pl = Страницах ,
7772     name-pl = страницах ,
7773     Name-sg-ab = С. ,
7774     name-sg-ab = с. ,
7775     Name-pl-ab = Сс. ,
7776     name-pl-ab = сс. ,
7777     rangesep = {\textendash} ,
7778     rangetopair = false ,
7779     +refbounds-rb = {,,} ,
7780
7781     type = line ,
7782     gender = f ,
7783     variant = n ,
7784     Name-sg = Строка ,
7785     name-sg = строка ,
7786     Name-pl = Строки ,
7787     name-pl = строки ,
7788     variant = a ,
7789     Name-sg = Строку ,
7790     name-sg = строку ,
7791     Name-pl = Строки ,
7792     name-pl = строки ,
7793     variant = g ,
7794     Name-sg = Строки ,
7795     name-sg = строки ,
7796     Name-pl = Строк ,
7797     name-pl = строк ,
7798     variant = d ,
7799     Name-sg = Строке ,
7800     name-sg = строке ,
7801     Name-pl = Строкам ,
7802     name-pl = строкам ,

```

```

7803 variant = i ,
7804     Name-sg = Строкой ,
7805     name-sg = строкой ,
7806     Name-pl = Строками ,
7807     name-pl = строками ,
7808 variant = p ,
7809     Name-sg = Строке ,
7810     name-sg = строке ,
7811     Name-pl = Строках ,
7812     name-pl = строках ,
7813
7814 type = figure ,
7815     gender = m ,
7816     variant = n ,
7817     Name-sg = Рисунок ,
7818     name-sg = рисунок ,
7819     Name-pl = Рисунки ,
7820     name-pl = рисунки ,
7821     Name-sg-ab = Рис. ,
7822     name-sg-ab = рис. ,
7823     Name-pl-ab = Рис. ,
7824     name-pl-ab = рис. ,
7825     variant = a ,
7826     Name-sg = Рисунок ,
7827     name-sg = рисунок ,
7828     Name-pl = Рисунки ,
7829     name-pl = рисунки ,
7830     Name-sg-ab = Рис. ,
7831     name-sg-ab = рис. ,
7832     Name-pl-ab = Рис. ,
7833     name-pl-ab = рис. ,
7834     variant = g ,
7835     Name-sg = Рисунка ,
7836     name-sg = рисунок ,
7837     Name-pl = Рисунков ,
7838     name-pl = рисунков ,
7839     Name-sg-ab = Рис. ,
7840     name-sg-ab = рис. ,
7841     Name-pl-ab = Рис. ,
7842     name-pl-ab = рис. ,
7843     variant = d ,
7844     Name-sg = Рисунку ,
7845     name-sg = рисунок ,
7846     Name-pl = Рисункам ,
7847     name-pl = рисункам ,
7848     Name-sg-ab = Рис. ,
7849     name-sg-ab = рис. ,
7850     Name-pl-ab = Рис. ,
7851     name-pl-ab = рис. ,
7852     variant = i ,
7853     Name-sg = Рисунком ,
7854     name-sg = рисунком ,
7855     Name-pl = Рисунками ,
7856     name-pl = рисунками ,

```

7857 Name-sg-ab = Рис. ,
7858 name-sg-ab = рис. ,
7859 Name-pl-ab = Рис. ,
7860 name-pl-ab = рис. ,
7861 variant = p ,
7862 Name-sg = Рисунке ,
7863 name-sg = рисунке ,
7864 Name-pl = Рисунках ,
7865 name-pl = рисунках ,
7866 Name-sg-ab = Рис. ,
7867 name-sg-ab = рис. ,
7868 Name-pl-ab = Рис. ,
7869 name-pl-ab = рис. ,
7870
7871 type = table ,
7872 gender = f ,
7873 variant = n ,
7874 Name-sg = Таблица ,
7875 name-sg = таблица ,
7876 Name-pl = Таблицы ,
7877 name-pl = таблицы ,
7878 Name-sg-ab = Табл. ,
7879 name-sg-ab = табл. ,
7880 Name-pl-ab = Табл. ,
7881 name-pl-ab = табл. ,
7882 variant = a ,
7883 Name-sg = Таблицу ,
7884 name-sg = таблицу ,
7885 Name-pl = Таблицы ,
7886 name-pl = таблицы ,
7887 Name-sg-ab = Табл. ,
7888 name-sg-ab = табл. ,
7889 Name-pl-ab = Табл. ,
7890 name-pl-ab = табл. ,
7891 variant = g ,
7892 Name-sg = Таблицы ,
7893 name-sg = таблицы ,
7894 Name-pl = Таблиц ,
7895 name-pl = таблиц ,
7896 Name-sg-ab = Табл. ,
7897 name-sg-ab = табл. ,
7898 Name-pl-ab = Табл. ,
7899 name-pl-ab = табл. ,
7900 variant = d ,
7901 Name-sg = Таблице ,
7902 name-sg = таблице ,
7903 Name-pl = Таблицам ,
7904 name-pl = таблицам ,
7905 Name-sg-ab = Табл. ,
7906 name-sg-ab = табл. ,
7907 Name-pl-ab = Табл. ,
7908 name-pl-ab = табл. ,
7909 variant = i ,
7910 Name-sg = Таблицей ,

7911 name-sg = таблицей ,
7912 Name-pl = Таблицами ,
7913 name-pl = таблицами ,
7914 Name-sg-ab = Табл. ,
7915 name-sg-ab = табл. ,
7916 Name-pl-ab = Табл. ,
7917 name-pl-ab = табл. ,
7918 variant = p ,
7919 Name-sg = Таблице ,
7920 name-sg = таблице ,
7921 Name-pl = Таблицах ,
7922 name-pl = таблицях ,
7923 Name-sg-ab = Табл. ,
7924 name-sg-ab = табл. ,
7925 Name-pl-ab = Табл. ,
7926 name-pl-ab = табл. ,
7927
7928 type = item ,
7929 gender = m ,
7930 variant = n ,
7931 Name-sg = Пункт ,
7932 name-sg = пункт ,
7933 Name-pl = Пункты ,
7934 name-pl = пункты ,
7935 Name-sg-ab = П. ,
7936 name-sg-ab = п. ,
7937 Name-pl-ab = Пп. ,
7938 name-pl-ab = пп. ,
7939 variant = a ,
7940 Name-sg = Пункт ,
7941 name-sg = пункт ,
7942 Name-pl = Пункты ,
7943 name-pl = пункты ,
7944 Name-sg-ab = П. ,
7945 name-sg-ab = п. ,
7946 Name-pl-ab = Пп. ,
7947 name-pl-ab = пп. ,
7948 variant = g ,
7949 Name-sg = Пункта ,
7950 name-sg = пункта ,
7951 Name-pl = Пунктов ,
7952 name-pl = пунктов ,
7953 Name-sg-ab = П. ,
7954 name-sg-ab = п. ,
7955 Name-pl-ab = Пп. ,
7956 name-pl-ab = пп. ,
7957 variant = d ,
7958 Name-sg = Пункту ,
7959 name-sg = пункту ,
7960 Name-pl = Пунктам ,
7961 name-pl = пунктам ,
7962 Name-sg-ab = П. ,
7963 name-sg-ab = п. ,
7964 Name-pl-ab = Пп. ,

7965 name-pl-ab = пп. ,
7966 variant = i ,
7967 Name-sg = Пунктом ,
7968 name-sg = пунктом ,
7969 Name-pl = Пунктами ,
7970 name-pl = пунктами ,
7971 Name-sg-ab = П. ,
7972 name-sg-ab = п. ,
7973 Name-pl-ab = Пп. ,
7974 name-pl-ab = пп. ,
7975 variant = p ,
7976 Name-sg = Пункте ,
7977 name-sg = пункте ,
7978 Name-pl = Пунктах ,
7979 name-pl = пунктах ,
7980 Name-sg-ab = П. ,
7981 name-sg-ab = п. ,
7982 Name-pl-ab = Пп. ,
7983 name-pl-ab = пп. ,
7984
7985 type = footnote ,
7986 gender = f ,
7987 variant = n ,
7988 Name-sg = Сноска ,
7989 name-sg = сноска ,
7990 Name-pl = Сноски ,
7991 name-pl = сноски ,
7992 variant = a ,
7993 Name-sg = Сноску ,
7994 name-sg = сноску ,
7995 Name-pl = Сноски ,
7996 name-pl = сноски ,
7997 variant = g ,
7998 Name-sg = Сноски ,
7999 name-sg = сноски ,
8000 Name-pl = Сносок ,
8001 name-pl = сносок ,
8002 variant = d ,
8003 Name-sg = Сноске ,
8004 name-sg = сноске ,
8005 Name-pl = Сноскам ,
8006 name-pl = сноскам ,
8007 variant = i ,
8008 Name-sg = Сноской ,
8009 name-sg = сноской ,
8010 Name-pl = Сносками ,
8011 name-pl = сносками ,
8012 variant = p ,
8013 Name-sg = Сноске ,
8014 name-sg = сноске ,
8015 Name-pl = Сносках ,
8016 name-pl = сносках ,
8017
8018 type = endnote ,

```

8019 gender = f ,
8020 variant = n ,
8021   Name-sg = Сноска ,
8022   name-sg = сноска ,
8023   Name-pl = Сноски ,
8024   name-pl = сноски ,
8025 variant = a ,
8026   Name-sg = Сноску ,
8027   name-sg = сноску ,
8028   Name-pl = Сноски ,
8029   name-pl = сноски ,
8030 variant = g ,
8031   Name-sg = Сноски ,
8032   name-sg = сноски ,
8033   Name-pl = Сносок ,
8034   name-pl = сносок ,
8035 variant = d ,
8036   Name-sg = Сноске ,
8037   name-sg = сноске ,
8038   Name-pl = Сноскам ,
8039   name-pl = сноскам ,
8040 variant = i ,
8041   Name-sg = Сноской ,
8042   name-sg = сноской ,
8043   Name-pl = Сносками ,
8044   name-pl = сносками ,
8045 variant = p ,
8046   Name-sg = Сноске ,
8047   name-sg = сноске ,
8048   Name-pl = Сносках ,
8049   name-pl = сносках ,
8050
8051 type = note ,
8052 gender = f ,
8053 variant = n ,
8054   Name-sg = Заметка ,
8055   name-sg = заметка ,
8056   Name-pl = Заметки ,
8057   name-pl = заметки ,
8058 variant = a ,
8059   Name-sg = Заметку ,
8060   name-sg = заметку ,
8061   Name-pl = Заметки ,
8062   name-pl = заметки ,
8063 variant = g ,
8064   Name-sg = Заметки ,
8065   name-sg = заметки ,
8066   Name-pl = Заметок ,
8067   name-pl = заметок ,
8068 variant = d ,
8069   Name-sg = Заметке ,
8070   name-sg = заметке ,
8071   Name-pl = Заметкам ,
8072   name-pl = заметкам ,

```

```

8073 variant = i ,
8074     Name-sg = Заметкой ,
8075     name-sg = заметкой ,
8076     Name-pl = Заметками ,
8077     name-pl = заметками ,
8078 variant = p ,
8079     Name-sg = Заметке ,
8080     name-sg = заметке ,
8081     Name-pl = Заметках ,
8082     name-pl = заметках ,
8083
8084 type = equation ,
8085     gender = n ,
8086     variant = n ,
8087     Name-sg = Уравнение ,
8088     name-sg = уравнение ,
8089     Name-pl = Уравнения ,
8090     name-pl = уравнения ,
8091     Name-sg-ab = Ур. ,
8092     name-sg-ab = ур. ,
8093     Name-pl-ab = Ур. ,
8094     name-pl-ab = ур. ,
8095     variant = a ,
8096     Name-sg = Уравнение ,
8097     name-sg = уравнение ,
8098     Name-pl = Уравнения ,
8099     name-pl = уравнения ,
8100     Name-sg-ab = Ур. ,
8101     name-sg-ab = ур. ,
8102     Name-pl-ab = Ур. ,
8103     name-pl-ab = ур. ,
8104     variant = g ,
8105     Name-sg = Уравнения ,
8106     name-sg = уравнения ,
8107     Name-pl = Уравнений ,
8108     name-pl = уравнений ,
8109     Name-sg-ab = Ур. ,
8110     name-sg-ab = ур. ,
8111     Name-pl-ab = Ур. ,
8112     name-pl-ab = ур. ,
8113     variant = d ,
8114     Name-sg = Уравнению ,
8115     name-sg = уравнению ,
8116     Name-pl = Уравнениям ,
8117     name-pl = уравнениям ,
8118     Name-sg-ab = Ур. ,
8119     name-sg-ab = ур. ,
8120     Name-pl-ab = Ур. ,
8121     name-pl-ab = ур. ,
8122     variant = i ,
8123     Name-sg = Уравнением ,
8124     name-sg = уравнением ,
8125     Name-pl = Уравнениями ,
8126     name-pl = уравнениями ,

```

```

8127     Name-sg-ab = Ур. ,
8128     name-sg-ab = ур. ,
8129     Name-pl-ab = Ур. ,
8130     name-pl-ab = ур. ,
8131     variant = p ,
8132     Name-sg = Уравнении ,
8133     name-sg = уравнении ,
8134     Name-pl = Уравнениях ,
8135     name-pl = уравнениях ,
8136     Name-sg-ab = Ур. ,
8137     name-sg-ab = ур. ,
8138     Name-pl-ab = Ур. ,
8139     name-pl-ab = ур. ,
8140     +refbounds-rb = {c\nobreakspace(,,)} ,
8141     refbounds-first-sg = {(,)}, ,
8142     refbounds = {(,,)} ,
8143
8144     type = theorem ,
8145     gender = f ,
8146     variant = n ,
8147     Name-sg = Теорема ,
8148     name-sg = теорема ,
8149     Name-pl = Теоремы ,
8150     name-pl = теоремы ,
8151     Name-sg-ab = Теор. ,
8152     name-sg-ab = теор. ,
8153     Name-pl-ab = Теор. ,
8154     name-pl-ab = теор. ,
8155     variant = a ,
8156     Name-sg = Теорему ,
8157     name-sg = теорему ,
8158     Name-pl = Теоремы ,
8159     name-pl = теоремы ,
8160     Name-sg-ab = Теор. ,
8161     name-sg-ab = теор. ,
8162     Name-pl-ab = Теор. ,
8163     name-pl-ab = теор. ,
8164     variant = g ,
8165     Name-sg = Теоремы ,
8166     name-sg = теоремы ,
8167     Name-pl = Теорем ,
8168     name-pl = теорем ,
8169     Name-sg-ab = Теор. ,
8170     name-sg-ab = теор. ,
8171     Name-pl-ab = Теор. ,
8172     name-pl-ab = теор. ,
8173     variant = d ,
8174     Name-sg = Теореме ,
8175     name-sg = теореме ,
8176     Name-pl = Теоремам ,
8177     name-pl = теоремам ,
8178     Name-sg-ab = Теор. ,
8179     name-sg-ab = теор. ,
8180     Name-pl-ab = Теор. ,

```

8181 name-pl-ab = теор. ,
8182 variant = i ,
8183 Name-sg = Теоремой ,
8184 name-sg = теоремой ,
8185 Name-pl = Теоремами ,
8186 name-pl = теоремами ,
8187 Name-sg-ab = Теор. ,
8188 name-sg-ab = теор. ,
8189 Name-pl-ab = Теор. ,
8190 name-pl-ab = теор. ,
8191 variant = p ,
8192 Name-sg = Теореме ,
8193 name-sg = теореме ,
8194 Name-pl = Теоремах ,
8195 name-pl = теоремах ,
8196 Name-sg-ab = Теор. ,
8197 name-sg-ab = теор. ,
8198 Name-pl-ab = Теор. ,
8199 name-pl-ab = теор. ,
8200
8201 type = lemma ,
8202 gender = f ,
8203 variant = n ,
8204 Name-sg = Лемма ,
8205 name-sg = лемма ,
8206 Name-pl = Леммы ,
8207 name-pl = леммы ,
8208 variant = a ,
8209 Name-sg = Лемму ,
8210 name-sg = лемму ,
8211 Name-pl = Леммы ,
8212 name-pl = леммы ,
8213 variant = g ,
8214 Name-sg = Леммы ,
8215 name-sg = леммы ,
8216 Name-pl = Лемм ,
8217 name-pl = лемм ,
8218 variant = d ,
8219 Name-sg = Лемме ,
8220 name-sg = лемме ,
8221 Name-pl = Леммам ,
8222 name-pl = леммам ,
8223 variant = i ,
8224 Name-sg = Леммой ,
8225 name-sg = леммой ,
8226 Name-pl = Леммами ,
8227 name-pl = леммами ,
8228 variant = p ,
8229 Name-sg = Лемме ,
8230 name-sg = лемме ,
8231 Name-pl = Леммах ,
8232 name-pl = леммах ,
8233
8234 type = corollary ,

```

8235 gender = m ,
8236 variant = n ,
8237   Name-sg = Вывод ,
8238   name-sg = вывод ,
8239   Name-pl = Выводы ,
8240   name-pl = выводы ,
8241 variant = a ,
8242   Name-sg = Вывод ,
8243   name-sg = вывод ,
8244   Name-pl = Выводы ,
8245   name-pl = выводы ,
8246 variant = g ,
8247   Name-sg = Вывода ,
8248   name-sg = вывода ,
8249   Name-pl = Выводов ,
8250   name-pl = выводов ,
8251 variant = d ,
8252   Name-sg = Выводу ,
8253   name-sg = выводу ,
8254   Name-pl = Выводам ,
8255   name-pl = выводам ,
8256 variant = i ,
8257   Name-sg = Выводом ,
8258   name-sg = выводом ,
8259   Name-pl = Выводами ,
8260   name-pl = выводами ,
8261 variant = p ,
8262   Name-sg = Выводе ,
8263   name-sg = выводе ,
8264   Name-pl = Выводах ,
8265   name-pl = выводах ,
8266
8267 type = proposition ,
8268   gender = n ,
8269   variant = n ,
8270     Name-sg = Предложение ,
8271     name-sg = предложение ,
8272     Name-pl = Предложения ,
8273     name-pl = предложения ,
8274     Name-sg-ab = Предл. ,
8275     name-sg-ab = предл. ,
8276     Name-pl-ab = Предл. ,
8277     name-pl-ab = предл. ,
8278   variant = a ,
8279     Name-sg = Предложение ,
8280     name-sg = предложение ,
8281     Name-pl = Предложения ,
8282     name-pl = предложения ,
8283     Name-sg-ab = Предл. ,
8284     name-sg-ab = предл. ,
8285     Name-pl-ab = Предл. ,
8286     name-pl-ab = предл. ,
8287   variant = g ,
8288     Name-sg = Предложения ,

```

8289 name-sg = предложения ,
8290 Name-pl = Предложений ,
8291 name-pl = предложений ,
8292 Name-sg-ab = Предл. ,
8293 name-sg-ab = предл. ,
8294 Name-pl-ab = Предл. ,
8295 name-pl-ab = предл. ,
8296 variant = d ,
8297 Name-sg = Предложению ,
8298 name-sg = предложению ,
8299 Name-pl = Предложениям ,
8300 name-pl = предложениям ,
8301 Name-sg-ab = Предл. ,
8302 name-sg-ab = предл. ,
8303 Name-pl-ab = Предл. ,
8304 name-pl-ab = предл. ,
8305 variant = i ,
8306 Name-sg = Предложением ,
8307 name-sg = предложением ,
8308 Name-pl = Предложениями ,
8309 name-pl = предложениями ,
8310 Name-sg-ab = Предл. ,
8311 name-sg-ab = предл. ,
8312 Name-pl-ab = Предл. ,
8313 name-pl-ab = предл. ,
8314 variant = p ,
8315 Name-sg = Предложении ,
8316 name-sg = предложении ,
8317 Name-pl = Предложениях ,
8318 name-pl = предложениях ,
8319 Name-sg-ab = Предл. ,
8320 name-sg-ab = предл. ,
8321 Name-pl-ab = Предл. ,
8322 name-pl-ab = предл. ,
8323
8324 type = definition ,
8325 gender = n ,
8326 variant = n ,
8327 Name-sg = Определение ,
8328 name-sg = определение ,
8329 Name-pl = Определения ,
8330 name-pl = определения ,
8331 Name-sg-ab = Опр. ,
8332 name-sg-ab = опр. ,
8333 Name-pl-ab = Опр. ,
8334 name-pl-ab = опр. ,
8335 variant = a ,
8336 Name-sg = Определение ,
8337 name-sg = определение ,
8338 Name-pl = Определения ,
8339 name-pl = определения ,
8340 Name-sg-ab = Опр. ,
8341 name-sg-ab = опр. ,
8342 Name-pl-ab = Опр. ,

8343 name-pl-ab = опр. ,
8344 variant = g ,
8345 Name-sg = Определения ,
8346 name-sg = определения ,
8347 Name-pl = Определений ,
8348 name-pl = определений ,
8349 Name-sg-ab = Опр. ,
8350 name-sg-ab = опр. ,
8351 Name-pl-ab = Опр. ,
8352 name-pl-ab = опр. ,
8353 variant = d ,
8354 Name-sg = Определению ,
8355 name-sg = определению ,
8356 Name-pl = Определениям ,
8357 name-pl = определениям ,
8358 Name-sg-ab = Опр. ,
8359 name-sg-ab = опр. ,
8360 Name-pl-ab = Опр. ,
8361 name-pl-ab = опр. ,
8362 variant = i ,
8363 Name-sg = Определением ,
8364 name-sg = определением ,
8365 Name-pl = Определениями ,
8366 name-pl = определениями ,
8367 Name-sg-ab = Опр. ,
8368 name-sg-ab = опр. ,
8369 Name-pl-ab = Опр. ,
8370 name-pl-ab = опр. ,
8371 variant = p ,
8372 Name-sg = Определении ,
8373 name-sg = определении ,
8374 Name-pl = Определениях ,
8375 name-pl = определениях ,
8376 Name-sg-ab = Опр. ,
8377 name-sg-ab = опр. ,
8378 Name-pl-ab = Опр. ,
8379 name-pl-ab = опр. ,
8380
8381 type = proof ,
8382 gender = n ,
8383 variant = n ,
8384 Name-sg = Доказательство ,
8385 name-sg = доказательство ,
8386 Name-pl = Доказательства ,
8387 name-pl = доказательства ,
8388 variant = a ,
8389 Name-sg = Доказательство ,
8390 name-sg = доказательство ,
8391 Name-pl = Доказательства ,
8392 name-pl = доказательства ,
8393 variant = g ,
8394 Name-sg = Доказательства ,
8395 name-sg = доказательства ,
8396 Name-pl = Доказательств ,

8397 name-pl = доказательств ,
8398 variant = d ,
8399 Name-sg = Доказательству ,
8400 name-sg = доказательству ,
8401 Name-pl = Доказательствам ,
8402 name-pl = доказательствам ,
8403 variant = i ,
8404 Name-sg = Доказательством ,
8405 name-sg = доказательством ,
8406 Name-pl = Доказательствами ,
8407 name-pl = доказательствами ,
8408 variant = p ,
8409 Name-sg = Доказательстве ,
8410 name-sg = доказательстве ,
8411 Name-pl = Доказательствах ,
8412 name-pl = доказательствах ,
8413
8414 type = result ,
8415 gender = m ,
8416 variant = n ,
8417 Name-sg = Результат ,
8418 name-sg = результат ,
8419 Name-pl = Результаты ,
8420 name-pl = результаты ,
8421 variant = a ,
8422 Name-sg = Результат ,
8423 name-sg = результат ,
8424 Name-pl = Результаты ,
8425 name-pl = результаты ,
8426 variant = g ,
8427 Name-sg = Результата ,
8428 name-sg = результата ,
8429 Name-pl = Результатов ,
8430 name-pl = результатов ,
8431 variant = d ,
8432 Name-sg = Результату ,
8433 name-sg = результату ,
8434 Name-pl = Результатам ,
8435 name-pl = результатам ,
8436 variant = i ,
8437 Name-sg = Результатом ,
8438 name-sg = результатом ,
8439 Name-pl = Результатами ,
8440 name-pl = результатами ,
8441 variant = p ,
8442 Name-sg = Результате ,
8443 name-sg = результате ,
8444 Name-pl = Результатах ,
8445 name-pl = результатах ,
8446
8447 type = remark ,
8448 gender = n ,
8449 variant = n ,
8450 Name-sg = Примечание ,

8451 name-sg = примечание ,
8452 Name-pl = Примечания ,
8453 name-pl = примечания ,
8454 Name-sg-ab = Прим. ,
8455 name-sg-ab = прим. ,
8456 Name-pl-ab = Прим. ,
8457 name-pl-ab = прим. ,
8458 variant = a ,
8459 Name-sg = Примечание ,
8460 name-sg = примечание ,
8461 Name-pl = Примечания ,
8462 name-pl = примечания ,
8463 Name-sg-ab = Прим. ,
8464 name-sg-ab = прим. ,
8465 Name-pl-ab = Прим. ,
8466 name-pl-ab = прим. ,
8467 variant = g ,
8468 Name-sg = Примечания ,
8469 name-sg = примечания ,
8470 Name-pl = Примечаний ,
8471 name-pl = примечаний ,
8472 Name-sg-ab = Прим. ,
8473 name-sg-ab = прим. ,
8474 Name-pl-ab = Прим. ,
8475 name-pl-ab = прим. ,
8476 variant = d ,
8477 Name-sg = Примечанию ,
8478 name-sg = примечанию ,
8479 Name-pl = Примечаниям ,
8480 name-pl = примечаниям ,
8481 Name-sg-ab = Прим. ,
8482 name-sg-ab = прим. ,
8483 Name-pl-ab = Прим. ,
8484 name-pl-ab = прим. ,
8485 variant = i ,
8486 Name-sg = Примечанием ,
8487 name-sg = примечанием ,
8488 Name-pl = Примечаниями ,
8489 name-pl = примечаниями ,
8490 Name-sg-ab = Прим. ,
8491 name-sg-ab = прим. ,
8492 Name-pl-ab = Прим. ,
8493 name-pl-ab = прим. ,
8494 variant = p ,
8495 Name-sg = Примечании ,
8496 name-sg = примечании ,
8497 Name-pl = Примечаниях ,
8498 name-pl = примечаниях ,
8499 Name-sg-ab = Прим. ,
8500 name-sg-ab = прим. ,
8501 Name-pl-ab = Прим. ,
8502 name-pl-ab = прим. ,
8503
8504 type = example ,

```

8505 gender = m ,
8506 variant = n ,
8507     Name-sg = Пример ,
8508     name-sg = пример ,
8509     Name-pl = Примеры ,
8510     name-pl = примеры ,
8511 variant = a ,
8512     Name-sg = Пример ,
8513     name-sg = пример ,
8514     Name-pl = Примеры ,
8515     name-pl = примеры ,
8516 variant = g ,
8517     Name-sg = Примера ,
8518     name-sg = примера ,
8519     Name-pl = Примеров ,
8520     name-pl = примеров ,
8521 variant = d ,
8522     Name-sg = Примеру ,
8523     name-sg = примеру ,
8524     Name-pl = Примерам ,
8525     name-pl = примерам ,
8526 variant = i ,
8527     Name-sg = Примером ,
8528     name-sg = примером ,
8529     Name-pl = Примерами ,
8530     name-pl = примерами ,
8531 variant = p ,
8532     Name-sg = Примере ,
8533     name-sg = примере ,
8534     Name-pl = Примерах ,
8535     name-pl = примерах ,
8536
8537 type = algorithm ,
8538 gender = m ,
8539 variant = n ,
8540     Name-sg = Алгоритм ,
8541     name-sg = алгоритм ,
8542     Name-pl = Алгоритмы ,
8543     name-pl = алгоритмы ,
8544 variant = a ,
8545     Name-sg = Алгоритм ,
8546     name-sg = алгоритм ,
8547     Name-pl = Алгоритмы ,
8548     name-pl = алгоритмы ,
8549 variant = g ,
8550     Name-sg = Алгоритма ,
8551     name-sg = алгоритма ,
8552     Name-pl = Алгоритмов ,
8553     name-pl = алгоритмов ,
8554 variant = d ,
8555     Name-sg = Алгоритму ,
8556     name-sg = алгоритму ,
8557     Name-pl = Алгоритмам ,
8558     name-pl = алгоритмам ,

```

```

8559 variant = i ,
8560     Name-sg = Алгоритмом ,
8561     name-sg = алгоритмом ,
8562     Name-pl = Алгоритмами ,
8563     name-pl = алгоритмами ,
8564 variant = p ,
8565     Name-sg = Алгоритме ,
8566     name-sg = алгоритме ,
8567     Name-pl = Алгоритмах ,
8568     name-pl = алгоритмах ,
8569
8570 type = listing ,
8571     gender = m ,
8572     variant = n ,
8573     Name-sg = Листинг ,
8574     name-sg = листинг ,
8575     Name-pl = Листинги ,
8576     name-pl = листинги ,
8577     variant = a ,
8578     Name-sg = Листинг ,
8579     name-sg = листинг ,
8580     Name-pl = Листинги ,
8581     name-pl = листинги ,
8582     variant = g ,
8583     Name-sg = Листинга ,
8584     name-sg = листинга ,
8585     Name-pl = Листингов ,
8586     name-pl = листингов ,
8587     variant = d ,
8588     Name-sg = Листингу ,
8589     name-sg = листингу ,
8590     Name-pl = Листингам ,
8591     name-pl = листингам ,
8592     variant = i ,
8593     Name-sg = Листингом ,
8594     name-sg = листингм ,
8595     Name-pl = Листингами ,
8596     name-pl = листингами ,
8597     variant = p ,
8598     Name-sg = Листинге ,
8599     name-sg = листинге ,
8600     Name-pl = Листингах ,
8601     name-pl = листингах ,
8602
8603 type = exercise ,
8604     gender = n ,
8605     variant = n ,
8606     Name-sg = Упражнение ,
8607     name-sg = упражнение ,
8608     Name-pl = Упражнения ,
8609     name-pl = упражнения ,
8610     Name-sg-ab = Упр. ,
8611     name-sg-ab = упр. ,
8612     Name-pl-ab = Упр. ,

```

8613 name-pl-ab = упр. ,
8614 variant = a ,
8615 Name-sg = Упражнение ,
8616 name-sg = упражнение ,
8617 Name-pl = Упражнения ,
8618 name-pl = упражнения ,
8619 Name-sg-ab = Упр. ,
8620 name-sg-ab = упр. ,
8621 Name-pl-ab = Упр. ,
8622 name-pl-ab = упр. ,
8623 variant = g ,
8624 Name-sg = Упражнения ,
8625 name-sg = упражнения ,
8626 Name-pl = Упражнений ,
8627 name-pl = упражнений ,
8628 Name-sg-ab = Упр. ,
8629 name-sg-ab = упр. ,
8630 Name-pl-ab = Упр. ,
8631 name-pl-ab = упр. ,
8632 variant = d ,
8633 Name-sg = Упражнению ,
8634 name-sg = упражнению ,
8635 Name-pl = Упражнениям ,
8636 name-pl = упражнениям ,
8637 Name-sg-ab = Упр. ,
8638 name-sg-ab = упр. ,
8639 Name-pl-ab = Упр. ,
8640 name-pl-ab = упр. ,
8641 variant = i ,
8642 Name-sg = Упражнением ,
8643 name-sg = упражнением ,
8644 Name-pl = Упражнениями ,
8645 name-pl = упражнениями ,
8646 Name-sg-ab = Упр. ,
8647 name-sg-ab = упр. ,
8648 Name-pl-ab = Упр. ,
8649 name-pl-ab = упр. ,
8650 variant = p ,
8651 Name-sg = Упражнении ,
8652 name-sg = упражнении ,
8653 Name-pl = Упражнениях ,
8654 name-pl = упражнениях ,
8655 Name-sg-ab = Упр. ,
8656 name-sg-ab = упр. ,
8657 Name-pl-ab = Упр. ,
8658 name-pl-ab = упр. ,
8659
8660 type = solution ,
8661 gender = n ,
8662 variant = n ,
8663 Name-sg = Решение ,
8664 name-sg = решение ,
8665 Name-pl = Решения ,
8666 name-pl = решения ,

```

8667 variant = a ,
8668     Name-sg = Решение ,
8669     name-sg = решение ,
8670     Name-pl = Решения ,
8671     name-pl = решения ,
8672 variant = g ,
8673     Name-sg = Решения ,
8674     name-sg = решения ,
8675     Name-pl = Решений ,
8676     name-pl = решений ,
8677 variant = d ,
8678     Name-sg = Решению ,
8679     name-sg = решению ,
8680     Name-pl = Решениям ,
8681     name-pl = решениям ,
8682 variant = i ,
8683     Name-sg = Решением ,
8684     name-sg = решением ,
8685     Name-pl = Решениями ,
8686     name-pl = решениями ,
8687 variant = p ,
8688     Name-sg = Решении ,
8689     name-sg = решении ,
8690     Name-pl = Решениях ,
8691     name-pl = решениях ,
8692 </lang-russian>

```

10.10 Swedish

Swedish language file initially contributed by ‘Timmyfox’ (issue [#35](#)).

```

8693 (*package)
8694 \zcDeclareLanguage { swedish }
8695 </package>
8696 (*lang-swedish)
8697 namesep = {\nobreakspace} ,
8698 pairsep = {\~och\nobreakspace} ,
8699 listsep = {,~} ,
8700 lastsep = {\~och\nobreakspace} ,
8701 tpairsep = {\~och\nobreakspace} ,
8702 tlistsep = {,~} ,
8703 tlastsep = {\~och\nobreakspace} ,
8704 notesep = {\~} ,
8705 rangesep = {\textendash} ,
8706 rangetopair = false ,
8707
8708 type = book ,
8709     Name-sg = Bok ,
8710     name-sg = bok ,
8711     Name-pl = Bok ,
8712     name-pl = bok ,
8713
8714 type = part ,

```

```

8715 Name-sg = Del ,
8716 name-sg = del ,
8717 Name-pl = Del ,
8718 name-pl = del ,
8719
8720 type = chapter ,
8721 Name-sg = Kapitel ,
8722 name-sg = kapitel ,
8723 Name-pl = Kapitel ,
8724 name-pl = kapitel ,
8725
8726 type = section ,
8727 Name-sg = Avsnitt ,
8728 name-sg = avsnitt ,
8729 Name-pl = Avsnitt ,
8730 name-pl = avsnitt ,
8731
8732 type = paragraph ,
8733 Name-sg = Paragraf ,
8734 name-sg = paragraf ,
8735 Name-pl = Paragraf ,
8736 name-pl = paragraf ,
8737
8738 type = appendix ,
8739 Name-sg = Bilaga ,
8740 name-sg = bilaga ,
8741 Name-pl = Bilaga ,
8742 name-pl = bilaga ,
8743
8744 type = page ,
8745 Name-sg = Sida ,
8746 name-sg = sida ,
8747 Name-pl = Sida ,
8748 name-pl = sida ,
8749
8750 type = line ,
8751 Name-sg = Rad ,
8752 name-sg = rad ,
8753 Name-pl = Rad ,
8754 name-pl = rad ,
8755
8756 type = figure ,
8757 Name-sg = Figur ,
8758 name-sg = figur ,
8759 Name-pl = Figur ,
8760 name-pl = figur ,
8761 Name-sg-ab = Fig. ,
8762 name-sg-ab = fig. ,
8763 Name-pl-ab = Fig. ,
8764 name-pl-ab = fig. ,
8765
8766 type = table ,
8767 Name-sg = Tabell ,
8768 name-sg = tabell ,

```

```

8769 Name-pl = Tabell ,
8770 name-pl = tabell ,
8771 Name-sg-ab = Tab. ,
8772 name-sg-ab = tab. ,
8773 Name-pl-ab = Tab. ,
8774 name-pl-ab = tab. ,
8775
8776 type = item ,
8777 Name-sg = Punkt ,
8778 name-sg = punkt ,
8779 Name-pl = Punkt ,
8780 name-pl = punkt ,
8781
8782 type = footnote ,
8783 Name-sg = Fotnot ,
8784 name-sg = fotnot ,
8785 Name-pl = Fotnot ,
8786 name-pl = fotnot ,
8787
8788 type = endnote ,
8789 Name-sg = Slutnot ,
8790 name-sg = slutnot ,
8791 Name-pl = Slutnot ,
8792 name-pl = slutnot ,
8793
8794 type = note ,
8795 Name-sg = Not ,
8796 name-sg = not ,
8797 Name-pl = Not ,
8798 name-pl = not ,
8799
8800 type = equation ,
8801 Name-sg = Ekvation ,
8802 name-sg = ekvation ,
8803 Name-pl = Ekvation ,
8804 name-pl = ekvation ,
8805 Name-sg-ab = Ekv. ,
8806 name-sg-ab = ekv. ,
8807 Name-pl-ab = Ekv. ,
8808 name-pl-ab = ekv. ,
8809 refbounds-first-sg = {,(,)}, ,
8810 refbounds = {(,,)} ,
8811
8812 type = theorem ,
8813 Name-sg = Sats ,
8814 name-sg = sats ,
8815 Name-pl = Sats ,
8816 name-pl = sats ,
8817
8818 type = lemma ,
8819 Name-sg = Hjälp-sats ,
8820 name-sg = hjälpsats ,
8821 Name-pl = Hjälp-sats ,
8822 name-pl = hjälpsats ,

```



```

8823
8824 type = corollary ,
8825   Name-sg = Följdsats ,
8826   name-sg = följsats ,
8827   Name-pl = Följdsats ,
8828   name-pl = följsats ,
8829
8830 type = proposition ,
8831   Name-sg = Påstående ,
8832   name-sg = påstående ,
8833   Name-pl = Påstående ,
8834   name-pl = påstående ,
8835
8836 type = definition ,
8837   Name-sg = Definition ,
8838   name-sg = definition ,
8839   Name-pl = Definition ,
8840   name-pl = definition ,
8841
8842 type = proof ,
8843   Name-sg = Bevis ,
8844   name-sg = bevis ,
8845   Name-pl = Bevis ,
8846   name-pl = bevis ,
8847
8848 type = result ,
8849   Name-sg = Resultat ,
8850   name-sg = resultat ,
8851   Name-pl = Resultat ,
8852   name-pl = resultat ,
8853
8854 type = remark ,
8855   Name-sg = Anmärkning ,
8856   name-sg = anmärkning ,
8857   Name-pl = Anmärkning ,
8858   name-pl = anmärkning ,
8859
8860 type = example ,
8861   Name-sg = Exempel ,
8862   name-sg = exempel ,
8863   Name-pl = Exempel ,
8864   name-pl = exempel ,
8865
8866 type = algorithm ,
8867   Name-sg = Algoritm ,
8868   name-sg = algoritm ,
8869   Name-pl = Algoritm ,
8870   name-pl = algoritm ,
8871
8872 type = listing ,
8873   Name-sg = Kod ,
8874   name-sg = kod ,
8875   Name-pl = Kod ,
8876   name-pl = kod ,

```

```

8877
8878 type = exercise ,
8879   Name-sg = Uppgift ,
8880   name-sg = uppgift ,
8881   Name-pl = Uppgift ,
8882   name-pl = uppgift ,
8883
8884 type = solution ,
8885   Name-sg = Lösning ,
8886   name-sg = lösning ,
8887   Name-pl = Lösning ,
8888   name-pl = lösning ,
8889 </lang-swedish>

```

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	
<code>\AddToHook</code>	120, 2053, 2097, 2120, 2150, 2152, 2191, 2266, 2308, 2462, 2475, 2483, 5360, 5381, 5397, 5421, 5436, 5461, 5479, 5513, 5549, 5574
<code>\AddToHookWithArguments</code> .	2437, 5681, 5685
<code>\alph</code>	5494, 5565
<code>\appendix</code>	2, <i>128</i> , <i>129</i> , <i>131</i> , <i>139</i>
<code>\appendixname</code>	<i>128</i> , <i>140</i>
<code>\AtEndOfPackage</code>	2473
B	
<code>\babelname</code>	2107
<code>\babelprovide</code>	30, <i>56</i>
bool commands:	
<code>\bool_gset_false:N</code>	559
<code>\bool_gset_true:N</code>	552, 2449
<code>\bool_if:NTF</code>	403, 480, 518, 576, 1771, 1822, 2057, 2061, 2439, 2485, 3458, 3853, 3991, 4117, 4153, 4187, 4253, 4266, 4278, 4328, 4344, 4354, 4359, 4403, 4406, 4462, 4485, 4492, 4498, 4509, 4515, 4543, 4588, 4610, 4639, 4788, 4955, 4957, 5516, 5577
<code>\bool_if:nTF</code>	86, 3569, 3579, 3602, 3619, 3634, 3699, 3707, 4337, 4709, 4750, 4831, 4848, 4899
<code>\bool_if_exist:NTF</code>	357, 367, 391, 401, 451, 468, 478, 503, 506, 514, 516, 530, 533, 540, 543, 550, 557
<code>\bool_lazy_all:nTF</code>	4430, 4967
<code>\bool_lazy_and:nnTF</code> . .	49, 54, 2609, 3429, 3450, 4221, 4291, 4662, 4938, 4982
<code>\bool_lazy_any:nTF</code>	5118, 5127
<code>\bool_lazy_or:nnTF</code>	1393, 1426, 1978, 2555, 2818, 3317, 3342, 3433, 4925, 5535
<code>\bool_new:N</code> .	138, 358, 368, 393, 452, 470, 508, 531, 534, 541, 544, 551, 558, 820, 1590, 1591, 1617, 1641, 2001, 2008, 2015, 2028, 2029, 2201, 2202, 2203, 2204, 2205, 2301, 2302, 2430, 2442, 3466, 3481, 3739, 3740, 3751, 3752, 3759, 3761, 3762, 3775, 3776, 3777, 3789, 3790, 5478, 5523, 5548
<code>\bool_set:Nn</code>	3426
<code>\bool_set_eq:NN</code>	566
<code>\bool_set_false:N</code> . .	392, 469, 505, 507, 542, 1603, 1607, 1774, 1872, 1920, 1965, 2036, 2045, 2046, 2063, 2070, 2213, 2217, 2224, 2232, 2233, 2234, 2326, 2338, 3471, 3562, 3806, 3807, 3822, 3859, 3870, 4265, 4452, 4453, 4460, 4461, 4692, 4906, 5125, 5142
<code>\bool_set_true:N</code>	359, 369, 453, 532, 535, 545, 1597, 1598, 1602, 1608, 1796, 1818, 1884, 1886, 1924, 1926, 1938, 1940, 1969, 1971, 1990, 1992, 2035, 2040, 2041, 2211, 2218, 2223, 2240, 2242, 2244, 2247, 2248, 2249, 2314, 2319, 3575, 3585, 3589, 3610, 3625, 3640, 3664, 3830, 3854, 3860, 3864, 3871, 4014, 4024, 4035, 4085, 4121, 4157, 4191, 4208, 4288, 4322, 4487, 4547, 4593, 4615, 4644, 4907, 4913, 4920, 5044, 5103, 5141, 5179, 5186,

5187, 5205, 5222, 5224, 5492, 5528, 5562	\exp_args:NNo 297, 303
\bool_until_do:Nn	\exp_args:No 303
. 1873, 1927, 1972, 3600, 3823	\exp_not:N
\bool_while_do:nn 5610	. 114, 4361, 4374, 4377, 4380, 4721, 4725, 4733, 4737, 4762, 4765, 4773, 4776, 4804, 4807, 4811, 4816, 4822, 4825, 4837, 4840, 4868, 4873, 4883, 4888
C	\exp_not:n 304, 4029, 4052, 4060, 4078, 4091, 4095, 4130, 4139, 4161, 4169, 4176, 4200, 4214, 4231, 4241, 4282, 4305, 4315, 4362, 4373, 4378, 4379, 4531, 4554, 4566, 4600, 4622, 4631, 4651, 4672, 4682, 4722, 4734, 4763, 4764, 4774, 4775, 4805, 4806, 4808, 4812, 4823, 4824, 4826, 4834, 4838, 4839, 4842, 4869, 4884
\chaptername 140	\ExplSyntaxOn 31, 951
clist commands:	F
\clist_map_inline:nn 651, 699, 716, 1035, 2235, 2451, 2957, 5495, 5567	\figurename 140
\counterwithin 5, 62	file commands:
\counterwithout 63	\file_get:nnNTF 945
\crefstripprefix 46	\fmtversion 5
cs commands:	\footnote 2
\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, 991, 1855, 1894, 1948, 2000, 2617, 4744, 4782, 5162, 5266, 5299, 5332	G
\cs_if_exist:NTF 24, 27, 66, 75, 96, 5336, 5348, 5400, 5597	group commands:
\cs_if_exist_p:N	\group_begin: 754, 937, 1773, 1860, 1908, 1953, 2881, 3423, 3437, 3469, 4361, 4377, 4721, 4733, 4762, 4773, 4804, 4811, 4822, 4837, 4868, 4883
. 51, 56, 4223, 4293, 4664, 5612	\group_end: 766, 989, 1837, 1889, 1943, 1995, 2909, 3440, 3463, 3473, 4374, 4380, 4725, 4737, 4765, 4776, 4807, 4816, 4825, 4840, 4873, 4888
\cs_if_exist_use:N 5460, 5650	H
\cs_new:Npn 64, 73, 84, 94, 105, 301, 307, 309, 311, 314, 325, 336, 348, 350, 738, 4705, 4745, 4783, 5152	\hyperlink 5149
\cs_new_protected:Npn 295, 352, 362, 372, 379, 386, 411, 421, 442, 444, 446, 456, 463, 501, 528, 538, 548, 555, 833, 935, 1539, 1558, 1757, 1856, 1904, 1949, 2481, 2615, 3421, 3485, 3527, 3538, 3550, 3677, 3729, 3791, 3998, 4456, 4701, 4703, 4897, 5146, 5163, 5234, 5267, 5300, 5529	\hypertarget 138
\cs_set_eq:NN 740	I
\cs_to_str:N 351	\IfBooleanT 3470
D	\IfClassLoadedTF 133
\DeclareHookRule 5683	\ifdraft 2216
\def 3	\IfFormatAtLeastTF 5, 6
E	\ifoptionfinal 2222
\edef 5687	\IfPackageAtLeastTF 2312
\eqref 133, 134	\IfPackageLoadedTF 131
exp commands:	\input 30, 31
\exp_args:Ne 945	int commands:
\exp_args:Nee	\int_case:nnTF
. 1775, 1787, 1799, 1809, 1875, 1929, 1974, 5165, 5171, 5193, 5197, 5212 4001, 4042, 4104, 4409, 4522, 4579
\exp_args:NNe 36, 39	\int_compare:nNnTF
\exp_args:NNNo 297 3611, 3626, 3641, 3653, 3665, 3685, 3687, 3731, 3899, 4020, 4048,
\exp_args:NNNV 1836, 1888, 1942, 1994	

P	
<code>\PackageError</code>	9
<code>\pagename</code>	140
<code>\pagenote</code>	131
<code>\pagenumbering</code>	7
<code>\pageref</code>	86
<code>\PagesCheckSetup</code>	46
<code>\paragraph</code>	62
<code>\part</code>	139
<code>\partname</code>	140
prg commands:	
<code>\prg_generate_conditional_-variant:Nnn</code>	440, 488, 499, 526, 571, 579, 748, 1902
<code>\prg_new_conditional:Npnn</code>	130, 132, 397, 474, 512, 573, 742
<code>\prg_new_protected_conditional:Npnn</code>	431, 490, 562, 1895
<code>\prg_return_false:</code>	131, 133, 405, 409, 438, 482, 486, 497, 520, 524, 569, 576, 577, 746, 1900
<code>\prg_return_true:</code>	131, 133, 404, 407, 436, 481, 484, 495, 519, 522, 567, 576, 745, 1899
<code>\prg_set_eq_conditional:NNn</code>	750
prop commands:	
<code>\prop_if_in:NnTF</code>	36
<code>\prop_if_in_p:Nn</code>	87
<code>\prop_item:Nn</code>	39, 88
<code>\prop_new:N</code>	2353, 2399
<code>\prop_put:Nnn</code>	1562
<code>\prop_remove:Nn</code>	1561
<code>\providecommand</code>	5
<code>\ProvidesExplPackage</code>	14
<code>\ProvidesFile</code>	30
R	
<code>\ref</code>	138
<code>\refstepcounter</code>	3, 4, 132, 134, 135
regex commands:	
<code>\regex_match:nnTF</code>	1898
<code>\renewlist</code>	136
<code>\RequirePackage</code>	16, 17, 18, 19, 2058
<code>\restoreapp</code>	129
<code>\roman</code>	7
S	
<code>\scantokens</code>	129
seq commands:	
<code>\seq_clear:N</code>	467, 844, 881, 962, 975, 1034, 1638, 2552, 2815, 2892, 2905, 2956, 3487, 5293
<code>\seq_const_from_clist:Nn</code>	22
<code>\seq_count:N</code>	1397, 1411, 1430, 1445, 2559, 2569, 2822, 2835, 3321, 3335, 3346, 3360
<code>\seq_gclear:N</code> ..	1390, 1423, 3314, 3339
<code>\seq_gconcat:NNN</code>	644, 648
<code>\seq_get_left:NN</code>	859, 870, 966, 1013, 2896, 2933, 3833
<code>\seq_gput_right:Nn</code> ...	977, 983, 2455
<code>\seq_gremove_all:Nn</code>	2487
<code>\seq_gset_eq:NN</code>	460, 1062
<code>\seq_gset_from_clist:Nn</code>	587, 596, 608, 623, 631, 792, 809
<code>\seq_gset_split:Nnn</code>	445
<code>\seq_if_empty:NnTF</code> ..	845, 882, 963, 1002, 1022, 2893, 2922, 2944, 3827, 4994
<code>\seq_if_exist:NnTF</code> ..	448, 458, 465, 476
<code>\seq_if_in:NnTF</code> ..	863, 897, 941, 1008, 1037, 2453, 2486, 2928, 2959, 3531, 4990
<code>\seq_item:Nn</code> ...	4715, 4720, 4726, 4728, 4731, 4732, 4738, 4739, 4756, 4761, 4766, 4768, 4771, 4772, 4777, 4778, 4809, 4810, 4817, 4819, 4854, 4866, 4874, 4877, 4881, 4882, 4889, 4890
<code>\seq_map_break:n</code> ...	108, 3720, 3723
<code>\seq_map_function:NN</code>	3490
<code>\seq_map_indexed_inline:Nn</code> ..	45, 3681
<code>\seq_map_inline:Nn</code>	1076, 1101, 1335, 1374, 1452, 2477, 2490, 2538, 2575, 2624, 2636, 2798, 2841, 2980, 3004, 3264, 3304, 3366, 3717, 5531
<code>\seq_map_tokens:Nn</code>	90
<code>\seq_new:N</code>	136, 137, 449, 459, 584, 585, 586, 595, 607, 622, 630, 643, 647, 787, 804, 934, 1054, 1626, 2382, 2443, 3465, 3484, 3738, 3755, 3778, 3779, 3780, 3781, 3782, 3783, 3784, 3785, 3786, 3787, 3788
<code>\seq_pop_left:NN</code>	3825
<code>\seq_put_right:Nn</code> ...	1038, 2960, 3534
<code>\seq_reverse:N</code>	1632
<code>\seq_set_eq:NN</code>	450, 494, 3793, 4012, 4022, 4033, 4083, 4119, 4155, 4189, 4205, 4286, 4320, 4330, 4545, 4590, 4612, 4641
<code>\seq_set_from_clist:Nn</code> ..	1631, 2386, 3425
<code>\seq_set_split:Nnn</code>	443
<code>\seq_sort:Nn</code>	88, 3493
<code>\seq_use:Nn</code>	5008
<code>\setcounter</code> ..	5362, 5383, 5398, 5423, 5437
<code>\sidefootnote</code>	131
sort commands:	
<code>\sort_return_same:</code>	89, 93, 3500, 3505, 3576, 3596, 3616, 3631, 3645, 3670, 3705, 3720, 3736

<code>\tl_if_empty_p:N</code>	50, 55, 2611, 4222, 4292, 4663, 4970, 4984, 5122, 5132, 5136
<code>\tl_if_empty_p:n</code>	1394, 1427, 2556, 2819, 3318, 3343, 3571, 3572, 3581, 3582, 3606, 3607, 3622, 3637
<code>\tl_if_eq:NNTF</code>	122, 3544, 3565, 3867
<code>\tl_if_eq:NnTF</code>	1785, 3488, 3520, 3691, 3694, 3719, 3722, 3836, 3882, 4916, 5169
<code>\tl_if_eq:nnTF</code>	1775, 1787, 1799, 1809, 1875, 1929, 1974, 3683, 5165, 5171, 5193, 5197, 5212
<code>\tl_if_exist:NTF</code>	354, 364, 374, 381, 388, 399, 415, 425, 744, 5651
<code>\tl_if_exist_p:N</code>	2610
<code>\tl_if_novalue:nTF</code>	2498, 2546, 2644, 2806
<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, 1564, 2183, 2206, 2274, 2295, 2346, 2423, 3475, 3476, 3477, 3478, 3479, 3480, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 3753, 3754, 3758, 3760, 3763, 3764, 3765, 3766, 3767, 3768, 3769, 3770, 3771, 3772, 3773, 3774
<code>\tl_put_left:Nn</code>	4340, 4347, 4390, 5029, 5030, 5072, 5074, 5076, 5078
<code>\tl_put_right:Nn</code>	4027, 4050, 4058, 4076, 4089, 4128, 4137, 4159, 4167, 4174, 4198, 4212, 4229, 4239, 4529, 4552, 4564, 4598, 4620, 4629, 4649, 4670, 4680, 4936, 4937, 4948, 5653
<code>\tl_reverse:N</code>	3554, 3557
<code>\tl_set:Nn</code>	297, 356, 737, 762, 954, 998, 1009, 1573, 1580, 1582, 1766, 1834, 1838, 1851, 1879, 1881, 1890, 1892, 1933, 1935, 1944, 1946, 1985, 1987, 1996, 1998, 2101, 2102, 2107, 2108, 2111, 2112, 2126, 2131, 2136, 2162, 2167, 2172, 2620, 2885, 2918, 2929, 3658, 3660, 3838, 3839, 4008, 4010, 4280, 4303, 4313, 4357, 4476, 4478, 4488, 4505, 4931, 4932, 4946, 5533
<code>\tl_set_eq:NN</code>	435, 4440
<code>\tl_show:N</code>	4404
<code>\tl_tail:N</code>	3659, 3661
<code>\tl_tail:n</code>	1880, 1882, 1934, 1936, 1986, 1988
<code>\tl_use:N</code>	319, 330, 341, 777, 943, 948, 978, 980, 984
	U
	use commands:
<code>\use:N</code>	25, 28, 825, 4225, 4299, 4666, 5326
<code>\UseHook</code>	1861, 1909, 1954
	X
<code>\xdef</code>	5682
	Z
<code>\zcDeclareLanguage</code>	13, 26, 752, 5692, 5897, 6350, 6565, 6794, 7008, 7226, 7459, 8694
<code>\zcDeclareLanguageAlias</code>	26, 769, 5693, 5694, 5695, 5696, 5697, 5698, 5699, 5900, 5901, 5902, 5903, 5904, 6351, 6566, 6567, 6568
<code>\zcLanguageSetup</code>	21, 30, 31, 69, 74, 75, 141, 2879
<code>\zcpageref</code>	86, 3467
<code>\zcref</code>	21, 66, 68, 84–88, 94, 96, 129, 133, 134, 3419, 3472
<code>\zcRefTypeSetup</code>	21, 69, 2618
<code>\zcsetup</code>	21, 55, 66, 68, 69, 2613
<code>\zlabel</code>	2, 64, 131, 132, 134
	zrefcheck commands:
<code>\zrefcheck_zcref_beg_label:</code>	3432
<code>\zrefcheck_zcref_end_label_-maybe:</code>	3454
<code>\zrefcheck_zcref_run_checks_on_-labels:n</code>	3455
	zrefclever commands:
<code>\zrefclever_language_if_declared:n</code>	750
<code>\zrefclever_language_if_declared:nTF</code>	750
<code>\zrefclever_language_varname:n</code>	740, 740
<code>\l_zrefclever_ref_language_tl</code>	736
	zrefclever internal commands:
<code>\l__zrefclever_abbrev_bool</code>	3763, 3944, 4939
<code>\l__zrefclever_amsmath_subequations_bool</code>	5478, 5492, 5516
<code>\l__zrefclever_breqn_dgroup_bool</code>	5548, 5562, 5577
<code>\l__zrefclever_cap_bool</code>	3763, 3940, 4926
<code>\l__zrefclever_capfirst_bool</code>	2008, 2011, 4928
<code>__zrefclever_compat_module:nn</code>	66, 2481, 2481, 5333, 5377, 5441, 5474, 5524, 5544, 5584, 5605, 5632, 5658, 5677
<code>__zrefclever_counter_reset_by:n</code>	6, 62, 63, 66, 68, 70, 75, 77, 79, 84, 84, 5342, 5354, 5406, 5416, 5486, 5556

__zrefclever_counter_reset_by_- aux:nn	4079, 4092, 4096, 4131, 4140, 4162, 4170, 4177, 4201, 4215, 4242, 4283,
__zrefclever_counter_reset_by_- aux:nnn	4316, 4349, 4532, 4555, 4567, 4601, 4623, 4632, 4652, 4683, <u>4705</u> , 4705, 4744
\l__zrefclever_counter_resetby_- prop	__zrefclever_get_ref_endrange:nnN <u>46</u> , <u>114</u> , <u>115</u> , 4232, 4306, 4673, <u>4745</u> , 4745, 4782
\l__zrefclever_counter_resetters_- seq	__zrefclever_get_ref_first: <u>113</u> , <u>114</u> , <u>118</u> , 4341, 4391, <u>4783</u> , 4783
\l__zrefclever_counter_type_prop	__zrefclever_get_rf_opt_bool:nN <u>127</u>
\l__zrefclever_current_counter_- tl	__zrefclever_get_rf_opt_- bool:nnnnN
\l__zrefclever_current_language_- tl <u>21</u> , 3937, 3941, 3945, <u>5300</u> , 5300, 5332
\l__zrefclever_endrangefunc_tl	__zrefclever_get_rf_opt_- seq:nnnN
..... <u>21</u> , <u>126</u> , 3949, 3953, 3957, 3961, 3965, 3969, 3973, 3977, 3981, 3985, 4986, <u>5267</u> , 5267, 5299
.....	__zrefclever_get_rf_opt_tl:nnnN . <u>21</u> , <u>23</u> , <u>46</u> , <u>125</u> , 3443, 3809, 3813, 3817, 3901, 3905, 3909, 3913, 3917, 3921, 3925, 3929, 3933, <u>5234</u> , 5234, 5266
.....	__zrefclever_hyperlink:nnn . <u>123</u> , 4363, 4716, 4757, 4794, 4856, <u>5146</u> , 5146
\l__zrefclever_endrangeprop_tl	\l__zrefclever_hyperlink_bool 2028, 2035, 2040, 2045, 2057, 2063, 2070, 3471, 4711, 4752, 4850, 5120
.....	\l__zrefclever_hyperref_warn_- bool . . 2029, 2036, 2041, 2046, 2061
__zrefclever_extract:nnn	__zrefclever_if_class_loaded:n <u>130</u> , <u>132</u>
.....	__zrefclever_if_class_loaded:nTF 5443, 5679
.....	__zrefclever_if_package_- loaded:n
..... <u>130</u> , <u>130</u>
__zrefclever_extract_default:Nnnn	__zrefclever_if_package_- loaded:nTF
..... 2055, 2099, 2105, 2310, 5379, 5476, 5526, 5546, 5586, 5607, 5634, 5660
.....	__zrefclever_is_integer_rgx:n <u>1895</u> , 1896, 1903
.....	__zrefclever_is_integer_rgx:nTF 1921, 1923, 1966, 1968
__zrefclever_extract_unexp:nnn	\l__zrefclever_label_a_tl
..... <u>93</u> , 3744, 3826, 3844, 3856, 3877, 3879, 3885, 3888, 3894, 4009, 4030, 4053, 4061, 4096, 4162, 4177, 4227, 4233, 4242, 4273, 4283, 4301, 4307, 4316, 4464, 4468, 4477, 4489, 4506, 4532, 4568, 4632, 4668, 4674, 4683
.....	\l__zrefclever_label_b_tl . . . <u>93</u> , <u>3744</u> , 3829, 3834, 3847, 3858, 3863, 4468
__zrefclever_extract_url_- unexp:n	\l__zrefclever_label_count_int <u>94</u> , <u>3741</u> , 3801, 3899, 4001, 4447, 4472, 4699, 4973
.....	
__zrefclever_get_enclosing_- counters:n	
....	
__zrefclever_get_enclosing_- counters_value:n	
<u>6</u> , <u>64</u> , <u>64</u> , <u>69</u> , <u>82</u> , <u>117</u>	
__zrefclever_get_endrange_- pagecomp:nnN	
.....	
__zrefclever_get_endrange_- pagecomptwo:nnN	
.....	
__zrefclever_get_endrange_- property:nnN	
.....	
__zrefclever_get_endrange_- stripprefix:nnN	
.....	
__zrefclever_get_ref:nN	
.....	

`\l_zrefclever_label_enclval_a_tl` 3475, 3552, 3554, 3606, 3622, 3642, 3654, 3658, 3659, 3666
`\l_zrefclever_label_enclval_b_tl` 3475, 3555, 3557, 3607, 3629, 3637, 3656, 3660, 3661, 3668
`\l_zrefclever_label_extdoc_a_tl` ... 3475, 3558, 3566, 3571, 3581, 3594
`\l_zrefclever_label_extdoc_b_tl` ... 3475, 3560, 3567, 3572, 3582, 3593
`\l_zrefclever_label_type_a_tl` 3444, 3475, 3530, 3532, 3535, 3541, 3545, 3691, 3719, 3810, 3814, 3818, 3838, 3843, 3868, 3882, 3902, 3906, 3910, 3914, 3918, 3922, 3926, 3930, 3934, 3938, 3942, 3946, 3950, 3954, 3958, 3962, 3966, 3970, 3974, 3978, 3982, 3986, 4011, 4479
`\l_zrefclever_label_type_b_tl` 3475, 3543, 3546, 3694, 3722, 3839, 3846, 3869
`_zrefclever_label_type_put_new_right:n` 87, 88, 3491, 3527, 3527
`\l_zrefclever_label_types_seq` 88, 3484, 3487, 3531, 3534, 3717
`\l_zrefclever_labelhook_bool` 2430, 2433, 2439
`_zrefclever_labels_in_sequence:mn` 48, 94, 124, 4271, 4467, 5163, 5163
`\l_zrefclever_lang_gender_seq` 581, 880, 881, 882, 897, 974, 975, 1022, 1037, 2904, 2905, 2944, 2959
`\l_zrefclever_lang_variant_tl` 581, 964, 967, 1009, 1014, 1349, 1366, 2894, 2897, 2929, 2934, 3278, 3295
`\l_zrefclever_lang_variants_seq` 581, 843, 844, 845, 859, 863, 870, 961, 962, 963, 966, 1002, 1008, 1013, 2891, 2892, 2893, 2896, 2922, 2928, 2933
`_zrefclever_language_if_declared:n` 26, 742, 749, 751
`_zrefclever_language_if_declared:n(TF)` 25
`_zrefclever_language_if_declared:nTF` 316, 327, 338, 742, 757, 773, 835, 939, 2137, 2173, 2882
`_zrefclever_language_varname:n` 25, 319, 330, 341, 738, 738, 741, 744, 760, 761, 775, 776, 777, 943, 948, 978, 980, 984
`\l_zrefclever_last_of_type_bool` 94, 3738, 3854, 3859, 3860, 3864, 3870, 3871, 3991
`\l_zrefclever_lastsep_tl` 3763, 3916, 4060, 4095, 4139, 4176, 4214
`\l_zrefclever_link_star_bool` 3426, 3465, 4712, 4753, 4851, 5121
`\l_zrefclever_listsep_tl` 3763, 3912, 4091, 4169, 4531, 4554, 4566, 4600, 4622, 4631, 4651
`\g_zrefclever_loaded_langfiles_seq` 934, 942, 977, 983
`\l_zrefclever_main_language_tl` 25, 55, 735, 2102, 2108, 2112, 2132, 2168
`\l_zrefclever_mathtools_loaded_bool` 3458, 5523, 5528
`_zrefclever_mathtools_showonlyrefs:n` 3460, 5529
`_zrefclever_name_default:` 4701, 4703, 4833
`\l_zrefclever_name_format_fallback_tl` 3750, 4946, 4950, 5016, 5065, 5077, 5079, 5097
`\l_zrefclever_name_format_tl` 3750, 4931, 4932, 4936, 4937, 4947, 4948, 5022, 5029, 5030, 5038, 5046, 5056, 5073, 5074, 5087, 5107
`\l_zrefclever_name_in_link_bool` 116, 118, 3750, 4359, 4788, 4906, 5125, 5141, 5142
`\l_zrefclever_namefont_tl` 3763, 3924, 4362, 4378, 4805, 4823, 4838
`\l_zrefclever_nameinlink_str` 2073, 2078, 2080, 2082, 2084, 5123, 5129, 5131, 5135
`\l_zrefclever_namesep_tl` 3763, 3904, 4808, 4826, 4834, 4842
`\l_zrefclever_next_is_same_bool` 94, 124, 3756, 4461, 4492, 4509, 4515, 5187, 5225
`\l_zrefclever_next_maybe_range_bool` 94, 124, 3756, 4265, 4278, 4460, 4485, 4498, 5179, 5186, 5205, 5223
`\l_zrefclever_noabbrev_first_bool` 2015, 2018, 4943
`\g_zrefclever_nocompat_bool` 2442, 2449, 2485
`\l_zrefclever_nocompat_bool` 65
`\g_zrefclever_nocompat_modules_seq` 2443, 2453, 2456, 2477, 2486, 2487
`\l_zrefclever_nocompat_modules_seq` 65
`\l_zrefclever_nudge_comptosing_bool` 2203, 2233, 2242, 2248, 4969
`\l_zrefclever_nudge_enabled_bool` 2201, 2211, 2213, 2217, 2218, 2223, 2224, 4432, 4955

```

\l_zrefclever_nudge_gender_bool
..... 2205, 2234, 2244, 2249, 4983
\l_zrefclever_nudge_multitype_
bool .. 2202, 2232, 2240, 2247, 4433
\l_zrefclever_nudge_singular_
bool ..... 2204, 2260, 4957
\__zrefclever_opt_bool_get:NN ..
..... 562, 572
\__zrefclever_opt_bool_get:NN(TF)
..... 20
\__zrefclever_opt_bool_get:NNTF
.... 562, 5303, 5308, 5313, 5318, 5323
\__zrefclever_opt_bool_gset_
false:N ..... 19,
528, 555, 561, 1507, 1524, 3397, 3405
\__zrefclever_opt_bool_gset_
true:N ..... 19,
528, 548, 554, 1469, 1486, 3376, 3384
\__zrefclever_opt_bool_if:N . 573, 580
\__zrefclever_opt_bool_if:N(TF) . 20
\__zrefclever_opt_bool_if:NNTF ..
..... 573, 908
\__zrefclever_opt_bool_if_set:N
..... 512, 527
\__zrefclever_opt_bool_if_
set:N(TF) ..... 19
\__zrefclever_opt_bool_if_
set:NNTF .....
. 512, 564, 575, 1462, 1478, 1500, 1516
\__zrefclever_opt_bool_set_
false:N 19, 528, 538, 547, 2588, 2858
\__zrefclever_opt_bool_set_
true:N . 19, 528, 528, 537, 2583, 2849
\__zrefclever_opt_bool_unset:N .
..... 18, 501, 501, 511, 2593, 2867
\__zrefclever_opt_seq_get:NN 490, 500
\__zrefclever_opt_seq_get:NN(TF) 18
\__zrefclever_opt_seq_get:NNTF .
..... 490, 838, 875, 956, 969,
2886, 2899, 5270, 5275, 5280, 5285, 5290
\__zrefclever_opt_seq_gset_
clist_split:Nn .....
.. 17, 442, 444, 1391, 1424, 3315, 3340
\__zrefclever_opt_seq_gset_eq:NN
..... 17, 442,
456, 462, 1400, 1433, 2967, 3324, 3349
\__zrefclever_opt_seq_if_set:N .
..... 474, 489
\__zrefclever_opt_seq_if_
set:N(TF) ..... 18
\__zrefclever_opt_seq_if_set:NNTF
..... 474, 492, 1045, 1384, 1416
\__zrefclever_opt_seq_set_clist_
split:Nn ... 17, 442, 442, 2553, 2816
\__zrefclever_opt_seq_set_eq:NN
..... 17, 442, 446, 455, 2562, 2825
\__zrefclever_opt_seq_unset:N ..
..... 17, 463, 463, 473, 2548, 2808
\__zrefclever_opt_tl_clear:N ...
15, 352, 362, 371, 1659, 1664, 1678,
1692, 1706, 2671, 2676, 2690, 2704, 2718
\__zrefclever_opt_tl_cset_
fallback:n ..... 1539, 1546
\__zrefclever_opt_tl_gclear:N ..
15, 352, 379, 385, 3045, 3051, 3059,
3066, 3086, 3102, 3122, 3138, 3158, 3174
\__zrefclever_opt_tl_gclear_if_
new:N ..... 16,
411, 421, 430, 1143, 1149, 1157,
1164, 1184, 1200, 1220, 1236, 1256, 1272
\__zrefclever_opt_tl_get:NN . 431, 441
\__zrefclever_opt_tl_get:NN(TF) . 17
\__zrefclever_opt_tl_get:NNTF ..
..... 431, 5018, 5033, 5052, 5061,
5082, 5092, 5237, 5242, 5247, 5252, 5257
\__zrefclever_opt_tl_gset:N ..... 15
\__zrefclever_opt_tl_gset:Nn ...
..... 352, 372, 378, 2990, 3014,
3022, 3079, 3094, 3115, 3130, 3151,
3166, 3199, 3206, 3215, 3223, 3280, 3290
\__zrefclever_opt_tl_gset_if_
new:Nn ..... 16,
411, 411, 420, 1086, 1111, 1120,
1177, 1192, 1213, 1228, 1249, 1264,
1297, 1304, 1313, 1321, 1351, 1361
\__zrefclever_opt_tl_if_set:N .. 397
\__zrefclever_opt_tl_if_set:N(TF)
..... 16
\__zrefclever_opt_tl_if_set:NNTF
..... 397, 413, 423, 433
\__zrefclever_opt_tl_set:N ..... 15
\__zrefclever_opt_tl_set:Nn ....
..... 352, 352,
361, 1672, 1686, 1700, 1739, 1745,
2504, 2653, 2684, 2698, 2712, 2751, 2758
\__zrefclever_opt_tl_unset:N ...
..... 15, 386, 386,
396, 1714, 1719, 2500, 2646, 2726, 2731
\__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:n .....
.. 14, 348, 348, 1542, 5258, 5291, 5324

```

```

\__zrefclever_opt_varname_-
  general:nn ..... 13,
  309, 309, 1661, 1666, 1674, 1680,
  1688, 1694, 1702, 1708, 1716, 1721,
  1741, 1747, 2501, 2505, 2549, 2563,
  2584, 2589, 2594, 5238, 5271, 5304
\__zrefclever_opt_varname_lang_-
  default:nnn ..... 13, 325, 325,
  335, 1088, 1113, 1145, 1151, 1179,
  1186, 1215, 1222, 1251, 1258, 1299,
  1306, 1386, 1402, 1464, 1471, 1502,
  1509, 2992, 3016, 3047, 3053, 3081,
  3088, 3117, 3124, 3153, 3160, 3201,
  3208, 3326, 3378, 3399, 5253, 5286, 5319
\__zrefclever_opt_varname_lang_-
  type:nnnn ..... 14,
  336, 336, 347, 1047, 1056, 1064,
  1122, 1159, 1166, 1194, 1202, 1230,
  1238, 1266, 1274, 1315, 1323, 1353,
  1363, 1418, 1435, 1480, 1488, 1518,
  1526, 2969, 3024, 3061, 3068, 3096,
  3104, 3132, 3140, 3168, 3176, 3217,
  3225, 3282, 3292, 3351, 3386, 3407,
  5035, 5084, 5094, 5248, 5281, 5314
\__zrefclever_opt_varname_-
  language:nnn ..... 13, 314,
  314, 324, 789, 794, 806, 811, 822,
  827, 840, 877, 910, 958, 971, 2888, 2901
\__zrefclever_opt_varname_-
  type:nnn ..... 13, 311, 311, 313,
  2648, 2655, 2673, 2678, 2686, 2692,
  2700, 2706, 2714, 2720, 2728, 2733,
  2753, 2760, 2810, 2827, 2851, 2860,
  2869, 5020, 5054, 5063, 5243, 5276, 5309
\g__zrefclever_page_format_int .
  ..... 118, 124, 128
\l__zrefclever_pairsep_tl .....
  ..... 3763, 3908,
  4029, 4052, 4078, 4130, 4161, 4200, 4282
\g__zrefclever_prev_page_format_-
  tl ..... 8, 119, 122, 125
\__zrefclever_process_language_-
  settings: ... 58, 59, 833, 833, 3428
\__zrefclever_prop_put_non_-
  empty:Nnn 43, 1558, 1558, 2364, 2410
\__zrefclever_provide_langfile:n
  21, 31, 32, 85, 935, 935, 991, 2143, 3427
\l__zrefclever_range_beg_is_-
  first_bool .....
  .... 3756, 3806, 4117, 4153, 4187,
  4452, 4487, 4543, 4588, 4610, 4639, 4692
\l__zrefclever_range_beg_label_-
  tl 94, 3756, 3799, 4080, 4093, 4132,
  4141, 4171, 4202, 4216, 4226, 4445,
  4488, 4505, 4556, 4602, 4624, 4653, 4667
\l__zrefclever_range_count_int .
  . 94, 3756, 3804, 4042, 4106, 4450,
  4491, 4502, 4508, 4514, 4522, 4581, 4694
\l__zrefclever_range_end_ref_tl
  ..... 3756, 3800, 4228, 4234,
  4302, 4308, 4446, 4490, 4507, 4669, 4675
\l__zrefclever_range_same_count_-
  int ..... 94,
  3756, 3805, 4020, 4071, 4107, 4451,
  4493, 4510, 4516, 4561, 4582, 4695
\l__zrefclever_rangeseptl 3763,
  3920, 4231, 4241, 4305, 4315, 4672, 4682
\l__zrefclever_rangetopair_bool
  ..... 3763, 3948, 4266
\l__zrefclever_ref_count_int 3741,
  3803, 4048, 4126, 4196, 4448, 4480,
  4528, 4551, 4563, 4597, 4619, 4628, 4648
\__zrefclever_ref_default: ....
  4701, 4701, 4742, 4748, 4786, 4827, 4893
\l__zrefclever_ref_gender_tl ...
  ..... 29, 884, 890,
  891, 895, 898, 903, 904, 923, 929,
  930, 2206, 2270, 4984, 4992, 4998, 5006
\l__zrefclever_ref_language_tl .
  ..... 25, 28, 55, 733,
  737, 836, 841, 851, 869, 878, 888,
  902, 911, 920, 927, 2126, 2131, 2136,
  2144, 2162, 2167, 2172, 3427, 3445,
  3811, 3815, 3819, 3903, 3907, 3911,
  3915, 3919, 3923, 3927, 3931, 3935,
  3939, 3943, 3947, 3951, 3955, 3959,
  3963, 3967, 3971, 3975, 3979, 3983,
  3987, 4988, 5000, 5011, 5036, 5085, 5095
\l__zrefclever_ref_property_tl .
  ..... 43, 48, 1564,
  1573, 1580, 1582, 1761, 1785, 1829,
  1846, 1858, 1865, 1870, 1906, 1913,
  1918, 1951, 1958, 1963, 3488, 3520,
  3836, 3889, 3893, 4707, 4792, 4846, 5169
\l__zrefclever_ref_typeset_font_-
  tl ..... 2274, 2276, 3438
\l__zrefclever_ref_variant_tl ..
  ..... 28, 847, 852, 853, 857,
  860, 864, 868, 871, 916, 919, 921,
  2183, 2195, 5027, 5031, 5070, 5075, 5080
\l__zrefclever_refbounds_first_-
  pb_seq ..... 3778,
  3960, 4034, 4084, 4156, 4207, 4287
\l__zrefclever_refbounds_first_-
  rb_seq . 3778, 3964, 4190, 4321, 4643
\l__zrefclever_refbounds_first_-
  seq 3778, 3952, 4331, 4546, 4592, 4614

```

\l__zrefclever_refbounds_first_- sg_seq .	3177, 3202, 3209, 3218, 3226, 3283, 3293, 3327, 3352, 3379, 3387, 3400, 3408
\l__zrefclever_refbounds_last_- pe_seq	\l__zrefclever_setup_type_tl 581, 955, 997, 998, 1028, 1049, 1058, 1066, 1084, 1109, 1124, 1141, 1161, 1168, 1175, 1196, 1204, 1211, 1232, 1240, 1247, 1268, 1276, 1295, 1317, 1325, 1343, 1355, 1365, 1382, 1420, 1437, 1460, 1482, 1490, 1498, 1520, 1528, 2620, 2622, 2649, 2656, 2674, 2679, 2687, 2693, 2701, 2707, 2715, 2721, 2729, 2734, 2754, 2761, 2811, 2828, 2852, 2861, 2870, 2884, 2917, 2918, 2950, 2971, 2988, 3012, 3026, 3043, 3063, 3070, 3077, 3098, 3106, 3113, 3134, 3142, 3149, 3170, 3178, 3197, 3219, 3227, 3272, 3284, 3294, 3312, 3353, 3374, 3388, 3395, 3409
\l__zrefclever_refbounds_last_- re_seq	
\l__zrefclever_refbounds_last_- seq	3778, 3988, 4235, 4243, 4309, 4317
\l__zrefclever_refbounds_mid_rb_- seq	3778, 3972, 4203, 4217, 4654
\l__zrefclever_refbounds_mid_re_- seq	3778, 3976, 4676, 4684
\l__zrefclever_refbounds_mid_seq	3778, 3968, 4094, 4172, 4533, 4557, 4569, 4603, 4625, 4633
\l__zrefclever_reffont_tl	3763, 3928, 4722, 4734, 4763, 4774, 4812, 4869, 4884
\l__zrefclever_reftype_override_- tl	34, 44, 2346, 2349
\g__zrefclever_rf_opts_bool_- maybe_type_specific_seq	53, 586, 1453, 2576, 2842, 3367
\g__zrefclever_rf_opts_seq_- refbounds_seq	586, 1375, 2539, 2799, 3305
\g__zrefclever_rf_opts_tl_maybe_- type_specific_seq	586, 1102, 3005
\g__zrefclever_rf_opts_tl_not_- type_specific_seq	586, 1077, 2625, 2981
\g__zrefclever_rf_opts_tl_- reference_seq	586, 2491
\g__zrefclever_rf_opts_tl_type_- names_seq	586, 1336, 3265
\g__zrefclever_rf_opts_tl_- typesetup_seq	586, 2637
\l__zrefclever_setup_language_tl	581, 762, 790, 795, 807, 812, 823, 828, 954, 1005, 1012, 1025, 1042, 1048, 1057, 1065, 1089, 1114, 1123, 1146, 1152, 1160, 1167, 1180, 1187, 1195, 1203, 1216, 1223, 1231, 1239, 1252, 1259, 1267, 1275, 1300, 1307, 1316, 1324, 1354, 1364, 1387, 1403, 1419, 1436, 1465, 1472, 1481, 1489, 1503, 1510, 1519, 1527, 2885, 2925, 2932, 2947, 2964, 2970, 2993, 3017, 3025, 3048, 3054, 3062, 3069, 3082, 3089, 3097, 3105, 3118, 3125, 3133, 3141, 3154, 3161, 3169,
\l__zrefclever_sort_decided_bool	3481, 3562, 3575, 3585, 3589, 3600, 3610, 3625, 3640, 3664
__zrefclever_sort_default:nn	89, 3522, 3538, 3538
__zrefclever_sort_default_- different_types:nn	45, 87, 92, 3548, 3677, 3677
__zrefclever_sort_default_same_- type:nn	87, 89, 3547, 3550, 3550
__zrefclever_sort_labels:	87-89, 93, 3436, 3485, 3485
__zrefclever_sort_page:nn	93, 3521, 3729, 3729
\l__zrefclever_sort_prior_a_int	3482, 3679, 3685, 3686, 3692, 3702, 3710
\l__zrefclever_sort_prior_b_int	3482, 3680, 3687, 3688, 3695, 3703, 3711
\l__zrefclever_tlastsep_tl	3763, 3820, 4426
\l__zrefclever_tlistsep_tl	3763, 3816, 4399
\l__zrefclever_tmpa_bool 134, 1774, 1796, 1818, 1822, 1872, 1873, 1884, 1886, 1920, 1924, 1926, 1927, 1938, 1940, 1965, 1969, 1971, 1972, 1990, 1992	
\l__zrefclever_tmpa_int	134, 5609, 5613, 5620, 5621, 5624, 5626, 5628
\g__zrefclever_tmpa_seq 134, 1390, 1392, 1397, 1406, 1411, 1423, 1425, 1430, 1440, 1445, 3314, 3316, 3321, 3330, 3335, 3339, 3341, 3346, 3355, 3360	
\l__zrefclever_tmpa_seq	134, 1034, 1038, 1070, 2552, 2554, 2559, 2564, 2569, 2815, 2817, 2822, 2830, 2835, 2956, 2960, 2975

<code>\l__zrefclever_tmpa_tl</code>	<code>\l__zrefclever_typeset_last_bool</code>
. 134 , 952 , 976 , 94 ,
1862 , 1876 , 1879 , 1880 , 1910 , 1921 ,	3738 , 3822 , 3823 , 3830 , 3853 , 4406 , 5137
1930 , 1933 , 1934 , 1955 , 1966 , 1975 ,	<code>\l__zrefclever_typeset_name_bool</code>
1985 , 1986 , 3446 , 3447 , 5533 , 5536 , 5537	1591 , 1598 , 1603 , 1608 , 4338 , 4354 , 4900
<code>\l__zrefclever_tmpb_tl</code> 134 ,	<code>\l__zrefclever_typeset_queue_-</code>
1824 , 1831 , 1834 , 1838 , 1867 , 1877 ,	curr_tl . . . 94 , 96 , 114 , 118 , 3744 ,
1881 , 1882 , 1883 , 1890 , 1915 , 1923 ,	3796 , 4027 , 4050 , 4058 , 4076 , 4089 ,
1931 , 1935 , 1936 , 1937 , 1944 , 1960 ,	4128 , 4137 , 4159 , 4167 , 4174 , 4198 ,
1968 , 1976 , 1979 , 1982 , 1987 , 1988 , 1996	4212 , 4229 , 4239 , 4255 , 4280 , 4303 ,
<code>\l__zrefclever_tpairsep_tl</code>	4313 , 4340 , 4347 , 4357 , 4390 , 4404 ,
. 3763 , 3812 , 4419	4414 , 4420 , 4427 , 4441 , 4442 , 4529 ,
<code>\l__zrefclever_type_count_int</code> . .	4552 , 4564 , 4598 , 4620 , 4629 , 4649 ,
. 94 , 118 , 3741 , 3802 , 4396 ,	4670 , 4680 , 4935 , 4959 , 4970 , 5132 , 5136
4398 , 4409 , 4434 , 4449 , 4929 , 4942 , 5138	<code>\l__zrefclever_typeset_queue_-</code>
<code>\l__zrefclever_type_first_label_-</code>	prev_tl . . . 94 , 3744 , 3795 , 4400 , 4440
tl 94 , 116 , 3744 , 3797 , 4008 , 4257 ,	<code>\l__zrefclever_typeset_range_-</code>
4268 , 4272 , 4300 , 4349 , 4366 , 4370 ,	bool . . . 1771 , 2001 , 2004 , 3435 , 4253
4443 , 4476 , 4785 , 4791 , 4797 , 4801 ,	<code>\l__zrefclever_typeset_ref_bool</code>
4814 , 4845 , 4859 , 4863 , 4871 , 4886 , 4910	1590 , 1597 , 1602 , 1607 , 4338 , 4344 , 4900
<code>\l__zrefclever_type_first_label_-</code>	<code>__zrefclever_typeset_refs:</code>
type_tl 94 , 118 , 3744 , 93-96 , 3439 , 3791 , 3791
3798 , 4010 , 4261 , 4444 , 4478 , 4917 ,	<code>__zrefclever_typeset_refs_last_-</code>
4963 , 4979 , 4987 , 4999 , 5005 , 5021 ,	of_type:
5037 , 5047 , 5055 , 5064 , 5086 , 5096 , 5108	. . . 100 , 114 , 116 , 118 , 3993 , 3998 , 3998
<code>\l__zrefclever_type_first_-</code>	<code>__zrefclever_typeset_refs_not_-</code>
refbounds_seq	last_of_type:
. 3778 , 4012 , 4022 , 4033 ,	. . . 94 , 100 , 114 , 124 , 3995 , 4456 , 4456
4083 , 4119 , 4155 , 4189 , 4206 , 4286 ,	<code>\l__zrefclever_typeset_sort_bool</code>
4320 , 4330 , 4350 , 4545 , 4591 , 4613 , 1617 , 1620 , 3434
4642 , 4809 , 4810 , 4817 , 4819 , 4855 ,	<code>\l__zrefclever_typesort_seq</code>
4867 , 4875 , 4878 , 4881 , 4882 , 4889 , 4890	. . . 45 , 92 , 1626 , 1631 , 1632 , 1638 , 3681
<code>\l__zrefclever_type_first_-</code>	<code>\l__zrefclever_verbose_testing_-</code>
refbounds_set_bool	bool 3790 , 4403
. 3778 , 3807 , 4014 , 4024 , 4035 ,	<code>__zrefclever_zcref:nnn</code>
4086 , 4122 , 4158 , 4192 , 4209 , 4288 , 28 , 57 , 3420 , 3421
4322 , 4328 , 4453 , 4548 , 4594 , 4616 , 4645	<code>__zrefclever_zcref:nnnn</code> 85 , 87 , 3421
<code>\l__zrefclever_type_name_gender_-</code>	<code>\l__zrefclever_zcref_labels_seq</code>
seq 3750 , 4989 , 4991 , 4994 , 5009 87 , 88 ,
<code>\l__zrefclever_type_name_-</code>	3425 , 3456 , 3461 , 3465 , 3490 , 3493 , 3794
missing_bool 118 ,	<code>\l__zrefclever_zcref_note_tl</code>
3750 , 4831 , 4907 , 4913 , 4920 , 5044 , 5104 2295 , 2298 , 3441 , 3448
<code>__zrefclever_type_name_setup: .</code>	<code>\l__zrefclever_zcref_with_check_-</code>
. 21 , 23 , 116 , 4336 , 4897 , 4897	bool 2302 , 2319 , 3431 , 3452
<code>\l__zrefclever_type_name_tl</code>	<code>__zrefclever_zcsetup:n</code>
. 116 , 118 , 69 , 2614 , 2615 , 2615 ,
3750 , 4373 , 4379 , 4806 , 4824 , 4839 ,	2617 , 5338 , 5350 , 5363 , 5384 , 5402 ,
4841 , 4905 , 4912 , 4919 , 5025 , 5041 ,	5412 , 5424 , 5445 , 5463 , 5481 , 5515 ,
5043 , 5059 , 5068 , 5090 , 5100 , 5102 , 5122	5551 , 5576 , 5588 , 5600 , 5616 , 5636 , 5662
<code>\l__zrefclever_typeset_compress_-</code>	<code>\l__zrefclever_zrefcheck_-</code>
bool 1641 , 1644 , 4462	available_bool
<code>\l__zrefclever_typeset_labels_-</code>	. . . 2301 , 2314 , 2326 , 2338 , 3430 , 3451
seq . . . 93 , 3738 , 3793 , 3825 , 3827 , 3833	