

The zref-clever package*

Code documentation

Gustavo Barros[†]

2023-02-21

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	8
4.1	Auxiliary	8
4.2	Messages	8
4.3	Data extraction	11
4.4	Option infra	12
4.5	Reference format	20
4.6	Languages	24
4.7	Language files	29
4.8	Options	42
5	Configuration	67
5.1	\zcsetup	67
5.2	\zcRefTypeSetup	67
5.3	\zcLanguageSetup	72
6	User interface	83
6.1	\zceref	83
6.2	\zcpageref	85
7	Sorting	85
8	Typesetting	92

*This file describes v0.3.6, released 2023-02-21.

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

9	Compatibility	126
9.1	appendix	126
9.2	appendices	127
9.3	memoir	128
9.4	amsmath	131
9.5	mathtools	133
9.6	breqn	134
9.7	listings	135
9.8	enumitem	136
9.9	subcaption	137
9.10	subfig	137
10	Language files	138
10.1	Localization guidelines	138
10.2	English	140
10.3	German	144
10.4	French	153
10.5	Portuguese	157
10.6	Spanish	162
10.7	Dutch	166
10.8	Italian	170
	Index	175

1 Initial setup

Start the DocStrip guards.

```

1 <*package>
   Identify the internal prefix (LATEX3 DocStrip convention).
2 <@@=zrefclever>
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`, even though I'd have loved to have used `\bool_case_true:...`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Finally, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (`ltxcmds`), with implications to the hook we add to `\appendix` (by Phelype Oleinik at <https://tex.stackexchange.com/q/617905> and <https://github.com/latex3/latex2e/pull/699>). Second, the support for `\@currentcounter` has been improved, including `\footnote` and `amsmath` (by Frank Mittelbach and Ulrike Fischer at <https://github.com/latex3/latex2e/issues/687>). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.

```

3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5   {}
6   {%
7     \PackageError{zref-clever}{LaTeX kernel too old}
```

```

8      {%
9      'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
10     \MessageBreak Loading will abort!%
11     }%
12   \endinput
13 }%

Identify the package.
14 \ProvidesExplPackage {zref-clever} {2023-02-21} {0.3.6}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

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

```

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

```

3 zref setup

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

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

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

```

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

```

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

```

22 \zref@newprop { thecounter }
23 {
24   \cs_if_exist:cTF { c@ \l_zrefclever_current_counter_tl }
25   { \use:c { the \l_zrefclever_current_counter_tl } }
26   {

```

```

27     \cs_if_exist:cT { c@ \@currentcounter }
28     { \use:c { the \@currentcounter } }
29   }
30 }
31 \zref@addprop \ZREF@mainlist { thecounter }

```

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

```

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

```

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

```

47 \zref@newprop { zc@cntval } [0]
48 {
49   \bool_lazy_and:nnTF
50   { ! \tl_if_empty_p:N \l__zrefclever_current_counter_tl }
51   { \cs_if_exist_p:c { c@ \l__zrefclever_current_counter_tl } }
52   { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
53   {
54     \bool_lazy_and:nnTF
55     { ! \tl_if_empty_p:N \@currentcounter }
56     { \cs_if_exist_p:c { c@ \@currentcounter } }
57     { \int_use:c { c@ \@currentcounter } }
58     { 0 }
59   }

```

```

60 }
61 \zref@addprop \ZREF@mainlist { zc@cntval }
62 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
63 \zref@addprop \ZREF@mainlist { zc@pgval }

```

However, since many counters (may) get reset along the document, we require more than just their numeric values. We need to know the reset chain of a given counter, in order to sort and compress a group of references. Also here, the “printed representation” is not enough, not only because it is easier to work with the numeric values but, given we occasionally group multiple counters within a single type, sorting this group requires to know the actual counter reset chain.

Furthermore, even if it is true that most of the definitions of counters, and hence of their reset behavior, is likely to be defined in the preamble, this is not necessarily true. Users can create counters, newtheorems mid-document, and alter their reset behavior along the way. Was that not the case, we could just store the desired information at `begindocument` in a variable and retrieve it when needed. But since it is, we need to store the information with the label, with the values as current when the label is set.

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of `\newcounter`, `\@addtoreset`, `\counterwithin`, and related infrastructure). The canonical optional argument of `\newcounter` establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “super-counter”, “parent counter” etc.). This information is somewhat tricky to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\c1@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see `ltxcounts.dtx` in `texdoc source2e`). Besides, there may be a chain of resetting counters, which must be taken into account: if `counterC` gets reset by `counterB`, and `counterB` gets reset by `counterA`, stepping the latter affects all three of them.

The procedure below examines a set of counters, those in `\l__zrefclever_counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\c1@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\c1@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification

has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

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

```

    \__zrefclever_get_enclosing_counters_value:n {<counter>}
64 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
65 {
66   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
67   {
68     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
69     \__zrefclever_get_enclosing_counters_value:e
70     { \__zrefclever_counter_reset_by:n {#1} }
71   }
72 }

```

Both `e` and `f` expansions work for this particular recursive call. I’ll stay with the `e` variant, since conceptually it is what I want (`x` itself is not expandable), and this package is anyway not compatible with older kernels for which the performance penalty of the `e` expansion would ensue (helpful comment by Enrico Gregorio, aka ‘egreg’ at https://tex.stackexchange.com/q/611370/#comment1529282_611385).

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

(End definition for `__zrefclever_get_enclosing_counters_value:n`.)

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

```

    \__zrefclever_counter_reset_by:n {<counter>}
74 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
75 {
76   \bool_if:nTF
77   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
78   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
79   {
80     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
81     { \__zrefclever_counter_reset_by_aux:nn {#1} }
82   }
83 }
84 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
85 {
86   \cs_if_exist:cT { c@ #2 }
87   {
88     \tl_if_empty:cF { c1@ #2 }

```

```

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

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

100 \zref@newprop { zc@enclval }
101 {
102     \__zrefclever_get_enclosing_counters_value:e
103     \l__zrefclever_current_counter_tl
104 }
105 \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

106 \tl_new:N \g__zrefclever_page_format_tl
107 \AddToHook { shipout / before }
108 {
109     \group_begin:
110     \int_set:Nn \c@page { 1 }
111     \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
112     \group_end:
113 }
114 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
115 \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.

```

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

```

(End definition for `__zrefclever_if_package_loaded:n` and `__zrefclever_if_class_loaded:n`.)

4.2 Messages

```

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

```



```

155     Can't set declension '#1' for unknown language '#2' \msg_line_context:..
156     See documentation for '\iow_char:N\zcDeclareLanguage' and
157     '\iow_char:N\zcDeclareLanguageAlias'.
158   }
159   \msg_new:nnn { zref-clever } { language-no-decl-ref }
160   {
161     Language '#1' has no declared declension cases \msg_line_context:..
162     Nothing to do with option 'd=#2'.
163   }
164   \msg_new:nnn { zref-clever } { language-no-gender }
165   {
166     Language '#1' has no declared gender \msg_line_context:..
167     Nothing to do with option '#2=#3'.
168   }
169   \msg_new:nnn { zref-clever } { language-no-decl-setup }
170   {
171     Language '#1' has no declared declension cases \msg_line_context:..
172     Nothing to do with option 'case=#2'.
173   }
174   \msg_new:nnn { zref-clever } { unknown-decl-case }
175   {
176     Declension case '#1' unknown for language '#2' \msg_line_context:..
177     Using default declension case.
178   }
179   \msg_new:nnn { zref-clever } { nudge-multitype }
180   {
181     Reference with multiple types \msg_line_context:..
182     You may wish to separate them or review language around it.
183   }
184   \msg_new:nnn { zref-clever } { nudge-comptosing }
185   {
186     Multiple labels have been compressed into singular type name
187     for type '#1' \msg_line_context:..
188   }
189   \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
190   {
191     Option 'sg' signals that a singular type name was expected
192     \msg_line_context:.. But type '#1' has plural type name.
193   }
194   \msg_new:nnn { zref-clever } { gender-not-declared }
195   { Language '#1' has no '#2' gender declared \msg_line_context:.. }
196   \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
197   {
198     Gender mismatch for type '#1' \msg_line_context:..
199     You've specified 'g=#2' but type name is '#3' for language '#4'.
200   }
201   \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
202   {
203     You've specified 'g=#1' \msg_line_context:..
204     But gender for type '#2' is not declared for language '#3'.
205   }
206   \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
207   { Unknown value '#1' for 'nudgeif' option \msg_line_context:.. }
208   \msg_new:nnn { zref-clever } { option-document-only }

```

```

209 { Option~'#1'~is-only-available-after~\iow_char:N\begin\{document\}. }
210 \msg_new:nnn { zref-clever } { langfile-loaded }
211 { Loaded~'#1'~language-file. }
212 \msg_new:nnn { zref-clever } { zref-property-undefined }
213 {
214   Option~'ref=#1'~requested~\msg_line_context:..~
215   But~the~property~'#1'~is-not-declared,-falling-back-to~'default'.
216 }
217 \msg_new:nnn { zref-clever } { endrange-property-undefined }
218 {
219   Option~'endrange=#1'~requested~\msg_line_context:..~
220   But~the~property~'#1'~is-not-declared,~'endrange'~not-set.
221 }
222 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
223 {
224   Option~'hyperref'~only~available-in~the~preamble~\msg_line_context:..~
225   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
226   '\iow_char:N\zcref'.
227 }
228 \msg_new:nnn { zref-clever } { missing-hyperref }
229 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
230 \msg_new:nnn { zref-clever } { option-preamble-only }
231 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:.. }
232 \msg_new:nnn { zref-clever } { unknown-compat-module }
233 {
234   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
235   Nothing~to~do.
236 }
237 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
238 {
239   The~value~of~option~'#1'~must~be~a~comma~separated~list~
240   of~four~items.~We~received~'#2'~items~\msg_line_context:..~
241   Option~not~set.
242 }
243 \msg_new:nnn { zref-clever } { missing-zref-check }
244 {
245   Option~'check'~requested~\msg_line_context:..~
246   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
247 }
248 \msg_new:nnn { zref-clever } { zref-check-too-old }
249 {
250   Option~'check'~requested~\msg_line_context:..~
251   But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
252 }
253 \msg_new:nnn { zref-clever } { missing-type }
254 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
255 \msg_new:nnn { zref-clever } { missing-property }
256 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:.. }
257 \msg_new:nnn { zref-clever } { missing-name }
258 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:.. }
259 \msg_new:nnn { zref-clever } { single-element-range }
260 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
261 \msg_new:nnn { zref-clever } { compat-package }
262 { Loaded~support~for~'#1'~package. }

```

```

263 \msg_new:nnn { zref-clever } { compat-class }
264 { Loaded-support-for-#1'-documentclass. }
265 \msg_new:nnn { zref-clever } { option-deprecated }
266 {
267   Option-#1'-has-been-deprecated-\msg_line_context:\iow_newline:
268   Use-#2'-instead.
269 }
270 \msg_new:nnn { zref-clever } { load-time-options }
271 {
272   'zref-clever'-does-not-accept-load-time-options.~
273   To-configure-package-options,~use-\iow_char:N\zcsetup'.
274 }

```

4.3 Data extraction

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

```

      \_zrefclever_extract_default:Nnnn {<tl var>}
      {<label>} {<prop>} {<default>}
275 \cs_new_protected:Npn \_zrefclever_extract_default:Nnnn #1#2#3#4
276 {
277   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
278   { \zref@extractdefault {#2} {#3} {#4} }
279 }
280 \cs_generate_variant:Nn \_zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

(End definition for `_zrefclever_extract_default:Nnnn`.)

`_zrefclever_extract_unexp:nnn` Extract property $\langle prop \rangle$ from $\langle label \rangle$. Ensure that, in the context of an x expansion, `\zref@extractdefault` is expanded exactly twice, but no further to retrieve the proper value. Thus, this is meant to be use in an x expansion context, not in other situations. In case the property is not found, leave $\langle default \rangle$ in the stream.

```

      \_zrefclever_extract_unexp:nnn{<label>}{<prop>}{<default>}
281 \cs_new:Npn \_zrefclever_extract_unexp:nnn #1#2#3
282 {
283   \exp_args:NNo \exp_args:No
284   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
285 }
286 \cs_generate_variant:Nn \_zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End definition for `_zrefclever_extract_unexp:nnn`.)

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

```

      \_zrefclever_extract:nnn{<label>}{<prop>}{<default>}
287 \cs_new:Npn \_zrefclever_extract:nnn #1#2#3
288 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for `_zrefclever_extract:nnn`.)

4.4 Option infra

This section provides the functions in which the variables naming scheme of the package options is embodied, and some basic general functions to query these option variables.

I had originally implemented the option handling of the package based on property lists, which are definitely very convenient. But as the number of options grew, I started to get concerned about the performance implications. That there was a toll was noticeable, even when we could live with it, of course. Indeed, at the time of writing, the typesetting of a reference queries about 24 different option values, most of them once per type-block, each of these queries can be potentially made in up to 5 option scope levels. Considering the size of the built-in language files is running at the hundreds, the package does have a lot of work to do in querying option values alone, and thus it is best to smooth things in this area as much as possible. This also gives me some peace of mind that the package will scale well in the long term. For some interesting discussion about alternative methods and their performance implications, see <https://tex.stackexchange.com/q/147966>. Phelype Oleinik also offered some insight on the matter at https://tex.stackexchange.com/questions/629946/#comment1571118_629946. The only real downside of this change is that we can no longer list the whole set of options in place at a given moment, which was useful for the purposes of regression testing, since we don't know what the whole set of active options is.

`_zrefclever_opt_varname_general:nn` Defines, and leaves in the input stream, the csname of the variable used to store the general $\langle option \rangle$. The data type of the variable must be specified (`tl`, `seq`, `bool`, etc.).

```
\_zrefclever_opt_varname_general:nn { $\langle option \rangle$ } { $\langle data type \rangle$ }
```

```
289 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
290 { l\_zrefclever_opt_general_ #1 _ #2 }
```

(End definition for `_zrefclever_opt_varname_general:nn`.)

`_zrefclever_opt_varname_type:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the type-specific $\langle option \rangle$ for $\langle ref type \rangle$.

```
\_zrefclever_opt_varname_type:nnn { $\langle ref type \rangle$ } { $\langle option \rangle$ } { $\langle data type \rangle$ }
```

```
291 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
292 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
293 \cs_generate_variant:Nn \_zrefclever_opt_varname_type:nnn { enn , een }
```

(End definition for `_zrefclever_opt_varname_type:nnn`.)

`_zrefclever_opt_varname_language:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the language $\langle option \rangle$ for $\langle lang \rangle$ (for general language options, those set with `\zcDeclareLanguage`). The “`lang_unknown`” branch should be guarded against, such as we normally should not get there, but this function *must* return some valid csname. The random part is there so that, in the circumstance this could not be avoided, we (hopefully) don't retrieve the value for an “unknown language” inadvertently.

```
\_zrefclever_opt_varname_language:nnn { $\langle lang \rangle$ } { $\langle option \rangle$ } { $\langle data type \rangle$ }
```

```

294 \cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
295 {
296   \__zrefclever_language_if_declared:nTF {#1}
297   {
298     g__zrefclever_opt_language_
299     \tl_use:c { \__zrefclever_language_varname:n {#1} }
300     _ #2 _ #3
301   }
302   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
303 }
304 \cs_generate_variant:Nn \__zrefclever_opt_varname_language:nnn { enn }

```

(End definition for `__zrefclever_opt_varname_language:nnn`.)

`__zrefclever_opt_varname_lang_default:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the language-specific default reference format $\langle option \rangle$ for $\langle lang \rangle$.

```

\__zrefclever_opt_varname_lang_default:nnn {<lang>} {<option>} {<data type>}
305 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
306 {
307   \__zrefclever_language_if_declared:nTF {#1}
308   {
309     g__zrefclever_opt_lang_
310     \tl_use:c { \__zrefclever_language_varname:n {#1} }
311     _default_ #2 _ #3
312   }
313   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
314 }
315 \cs_generate_variant:Nn \__zrefclever_opt_varname_lang_default:nnn { enn }

```

(End definition for `__zrefclever_opt_varname_lang_default:nnn`.)

`__zrefclever_opt_varname_lang_type:nnnn` Defines, and leaves in the input stream, the csname of the variable used to store the language- and type-specific reference format $\langle option \rangle$ for $\langle lang \rangle$ and $\langle ref type \rangle$.

```

\__zrefclever_opt_varname_lang_type:nnnn {<lang>} {<ref type>}
{<option>} {<data type>}
316 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
317 {
318   \__zrefclever_language_if_declared:nTF {#1}
319   {
320     g__zrefclever_opt_lang_
321     \tl_use:c { \__zrefclever_language_varname:n {#1} }
322     _type_ #2 _ #3 _ #4
323   }
324   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
325 }
326 \cs_generate_variant:Nn
327 \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }

```

(End definition for `__zrefclever_opt_varname_lang_type:nnnn`.)

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

```

    \_zrefclever_opt_varname_fallback:nn {<option>} {<data type>}
328 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
329 { c\_zrefclever_opt_fallback_ #1 _ #2 }

```

(End 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>}
330 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
331 { \cs_to_str:N #1 _is_set_bool }

```

(End definition for _zrefclever_opt_var_set_bool:n.)

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

```

```

357 }
358 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset:Nn { cn }
359 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear:N #1
360 {
361   \tl_if_exist:NF #1
362     { \tl_new:N #1 }
363   \tl_gclear:N #1
364 }
365 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear:N { c }

```

(End definition for `__zrefclever_opt_tl_set:Nn` and others.)

`__zrefclever_opt_tl_unset:N` Unset $\langle option\ tl \rangle$.

```

\__zrefclever_opt_tl_unset:N { $\langle option\ tl \rangle$ }
366 \cs_new_protected:Npn \__zrefclever_opt_tl_unset:N #1
367 {
368   \tl_if_exist:NT #1
369   {
370     \tl_clear:N #1 % ?
371     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
372       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
373       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
374   }
375 }
376 \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End definition for `__zrefclever_opt_tl_unset:N`.)

`__zrefclever_opt_tl_if_set:NTF` This conditional *defines* what means to be unset for a token list option. Note that the “set bool” not existing signals that the variable *is set*, that would be the case of all global option variables (language-specific ones). But this means care should be taken to always define and set the “set bool” for local variables.

```

\__zrefclever_opt_tl_if_set:N(TF) { $\langle option\ tl \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }
377 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
378 {
379   \tl_if_exist:NTF #1
380   {
381     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
382     {
383       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
384         { \prg_return_true: }
385         { \prg_return_false: }
386     }
387     { \prg_return_true: }
388   }
389   { \prg_return_false: }
390 }

```

(End definition for `__zrefclever_opt_tl_if_set:NTF`.)

```

\__zrefclever_opt_tl_gset_if_new:Nn      \__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\__zrefclever_opt_tl_gclear_if_new:N     \__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
391 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
392 {
393   \__zrefclever_opt_tl_if_set:NF #1
394   {
395     \tl_if_exist:NF #1
396     { \tl_new:N #1 }
397     \tl_gset:Nn #1 {#2}
398   }
399 }
400 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
401 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
402 {
403   \__zrefclever_opt_tl_if_set:NF #1
404   {
405     \tl_if_exist:NF #1
406     { \tl_new:N #1 }
407     \tl_gclear:N #1
408   }
409 }
410 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }

```

(End definition for __zrefclever_opt_tl_gset_if_new:Nn and __zrefclever_opt_tl_gclear_if_new:N.)

```

\__zrefclever_opt_tl_get:NNTF           \__zrefclever_opt_tl_get:NN(TF) {<option tl to get>} {<tl var to set>}
                                         {<true>} {<false>}
411 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
412 {
413   \__zrefclever_opt_tl_if_set:NTF #1
414   {
415     \tl_set_eq:NN #2 #1
416     \prg_return_true:
417   }
418   { \prg_return_false: }
419 }
420 \prg_generate_conditional_variant:Nnn
421 \__zrefclever_opt_tl_get:NN { cN } { F }

```

(End definition for __zrefclever_opt_tl_get:NNTF.)

```

\__zrefclever_opt_seq_set_clist_split:Nn \__zrefclever_opt_seq_set_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_gset_clist_split:Nn \__zrefclever_opt_seq_gset_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_set_eq:NN         \__zrefclever_opt_seq_set_eq:NN {<option seq>} {<seq var>}
\__zrefclever_opt_seq_gset_eq:NN       \__zrefclever_opt_seq_gset_eq:NN {<option seq>} {<seq var>}
422 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
423 { \seq_set_split:Nnn #1 { , } {#2} }
424 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
425 { \seq_gset_split:Nnn #1 { , } {#2} }
426 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
427 {
428   \seq_if_exist:NF #1
429   { \seq_new:N #1 }

```



```

430 \seq_set_eq:NN #1 #2
431 \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
432 { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
433 \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
434 }
435 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
436 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
437 {
438 \seq_if_exist:NF #1
439 { \seq_new:N #1 }
440 \seq_gset_eq:NN #1 #2
441 }
442 \cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }

```

(End definition for __zrefclever_opt_seq_set_clist_split:Nn and others.)

__zrefclever_opt_seq_unset:N Unset *<option seq>*.

```

\__zrefclever_opt_seq_unset:N {<option seq>}
443 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
444 {
445 \seq_if_exist:NT #1
446 {
447 \seq_clear:N #1 % ?
448 \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
449 { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
450 { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
451 }
452 }
453 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

(End definition for __zrefclever_opt_seq_unset:N.)

__zrefclever_opt_seq_if_set:NTF This conditional *defines* what means to be unset for a sequence option.

```

\__zrefclever_opt_seq_if_set:N(TF) {<option seq>} {<true>} {<false>}
454 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
455 {
456 \seq_if_exist:NTF #1
457 {
458 \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
459 {
460 \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
461 { \prg_return_true: }
462 { \prg_return_false: }
463 }
464 { \prg_return_true: }
465 }
466 { \prg_return_false: }
467 }
468 \prg_generate_conditional_variant:Nnn
469 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_seq_if_set:NTF.)

```

_zrefclever_opt_seq_get:NTF      \_zrefclever_opt_seq_get:NN(TF) {<option seq to get>} {<seq var to set>}
                                {<true>} {<false>}
470 \prg_new_protected_conditional:Npnn \_zrefclever_opt_seq_get:NN #1#2 { F }
471 {
472   \_zrefclever_opt_seq_if_set:NTF #1
473   {
474     \seq_set_eq:NN #2 #1
475     \prg_return_true:
476   }
477   { \prg_return_false: }
478 }
479 \prg_generate_conditional_variant:Nnn
480 \_zrefclever_opt_seq_get:NN { cN } { F }

(End definition for \_zrefclever_opt_seq_get:NTF.)

```

_zrefclever_opt_bool_unset:N Unset <option bool>.

```

                                \_zrefclever_opt_bool_unset:N {<option bool>}
481 \cs_new_protected:Npn \_zrefclever_opt_bool_unset:N #1
482 {
483   \bool_if_exist:NT #1
484   {
485     % \bool_set_false:N #1 % ?
486     \bool_if_exist:cTF { \_zrefclever_opt_var_set_bool:n {#1} }
487     { \bool_set_false:c { \_zrefclever_opt_var_set_bool:n {#1} } }
488     { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
489   }
490 }
491 \cs_generate_variant:Nn \_zrefclever_opt_bool_unset:N { c }

(End definition for \_zrefclever_opt_bool_unset:N.)

```

_zrefclever_opt_bool_if_set:NTF This conditional *defines* what means to be unset for a boolean option.

```

                                \_zrefclever_opt_bool_if_set:N(TF) {<option bool>} {<true>} {<false>}
492 \prg_new_conditional:Npnn \_zrefclever_opt_bool_if_set:N #1 { F , TF }
493 {
494   \bool_if_exist:NTF #1
495   {
496     \bool_if_exist:cTF { \_zrefclever_opt_var_set_bool:n {#1} }
497     {
498       \bool_if:cTF { \_zrefclever_opt_var_set_bool:n {#1} }
499       { \prg_return_true: }
500       { \prg_return_false: }
501     }
502     { \prg_return_true: }
503   }
504   { \prg_return_false: }
505 }
506 \prg_generate_conditional_variant:Nnn
507 \_zrefclever_opt_bool_if_set:N { c } { F , TF }

(End definition for \_zrefclever_opt_bool_if_set:NTF.)

```

```

    \_zrefclever_opt_bool_set_true:N {<option bool>}
    \_zrefclever_opt_bool_set_false:N {<option bool>}
    \_zrefclever_opt_bool_gset_true:N {<option bool>}
    \_zrefclever_opt_bool_gset_false:N {<option bool>}
508 \cs_new_protected:Npn \_zrefclever_opt_bool_set_true:N #1
509 {
510   \bool_if_exist:NF #1
511   { \bool_new:N #1 }
512   \bool_set_true:N #1
513   \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
514   { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
515   \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} }
516 }
517 \cs_generate_variant:Nn \_zrefclever_opt_bool_set_true:N { c }
518 \cs_new_protected:Npn \_zrefclever_opt_bool_set_false:N #1
519 {
520   \bool_if_exist:NF #1
521   { \bool_new:N #1 }
522   \bool_set_false:N #1
523   \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
524   { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
525   \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} }
526 }
527 \cs_generate_variant:Nn \_zrefclever_opt_bool_set_false:N { c }
528 \cs_new_protected:Npn \_zrefclever_opt_bool_gset_true:N #1
529 {
530   \bool_if_exist:NF #1
531   { \bool_new:N #1 }
532   \bool_gset_true:N #1
533 }
534 \cs_generate_variant:Nn \_zrefclever_opt_bool_gset_true:N { c }
535 \cs_new_protected:Npn \_zrefclever_opt_bool_gset_false:N #1
536 {
537   \bool_if_exist:NF #1
538   { \bool_new:N #1 }
539   \bool_gset_false:N #1
540 }
541 \cs_generate_variant:Nn \_zrefclever_opt_bool_gset_false:N { c }

```

(End definition for _zrefclever_opt_bool_set_true:N and others.)

```

\_zrefclever_opt_bool_get:NNTF \_zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
  {<true>} {<false>}
542 \prg_new_protected_conditional:Npnn \_zrefclever_opt_bool_get:NN #1#2 { F }
543 {
544   \_zrefclever_opt_bool_if_set:NTF #1
545   {
546     \bool_set_eq:NN #2 #1
547     \prg_return_true:
548   }
549   { \prg_return_false: }
550 }
551 \prg_generate_conditional_variant:Nnn
552 \_zrefclever_opt_bool_get:NN { cN } { F }

```

(End definition for `_zrefclever_opt_bool_get:NNTF`.)

```
\_zrefclever_opt_bool_if:NTF      \_zrefclever_opt_bool_if:N(TF) {\option bool} {\true} {\false}
553 \prg_new_conditional:Npnn \_zrefclever_opt_bool_if:N #1 { T , F , TF }
554 {
555   \_zrefclever_opt_bool_if_set:NTF #1
556   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
557   { \prg_return_false: }
558 }
559 \prg_generate_conditional_variant:Nnn
560 \_zrefclever_opt_bool_if:N { c } { T , F , TF }
```

(End definition for `_zrefclever_opt_bool_if:NTF`.)

4.5 Reference format

For a general discussion on the precedence rules for reference format options, see Section “Reference format” in the User manual. Internally, these precedence rules are handled / enforced in `_zrefclever_get_rf_opt_tl:nnnN`, `_zrefclever_get_rf_opt_seq:nnnN`, `_zrefclever_get_rf_opt_bool:nnnnN`, and `_zrefclever_type_name_setup`: which are the basic functions to retrieve proper values for reference format settings.

The fact that we have multiple scopes to set reference format options has some implications for how we handle these options, and for the resulting UI. Since there is a clear precedence rule between the different levels, setting an option at a high priority level shadows everything below it. Hence, it may be relevant to be able to “unset” these options too, so as to be able go back to the lower precedence level of the language-specific options at any given point. However, since many of these options are token lists, or clists, for which “empty” is a legitimate value, we cannot rely on emptiness to distinguish that particular intention. How to deal with it, depends on the kind of option (its data type, to be precise). For token lists and clists/sequences, we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit in `\keys_define:nn` by means of the `.default:o` property of the key. For the technique, by Jonathan P. Spratte, aka ‘Skillmon’, and some discussion about it, including further insights by Phelype Oleinik, see <https://tex.stackexchange.com/q/614690> and <https://github.com/latex3/latex3/pull/988>. However, Joseph Wright seems to particularly dislike this use and the general idea of a “key with no value” being somehow meaningful for `l3keys` (e.g. his comments on the previous question, and https://tex.stackexchange.com/q/632157/#comment1576404_632157), which does make it somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the “key with no value” is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value “unset” for this purpose. And similarly for “choice” options.

However, “unsetting” options is only supported at the general and reference type levels, that is, at `\zcsetup`, at `\zcref`, and at `\zcRefTypeSetup`. For language-specific options – in the language files or at `\zcLanguageSetup` – there is no unsetting, an option which has been set can there only be changed to another value. This for two reasons. First, these are low precedence levels, so it is less meaningful to be able to unset these options. Second, these settings can only be done in the preamble (or the package itself).

They are meant to be global. So, do it once, do it right, and if you need to locally change something along the document, use a higher precedence level.

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

```

\l_zrefclever_setup_language_tl
\l_zrefclever_lang_decl_case_tl
\l_zrefclever_lang_declension_seq
\l_zrefclever_lang_gender_seq
561 \tl_new:N \l__zrefclever_setup_type_tl
562 \tl_new:N \l__zrefclever_setup_language_tl
563 \tl_new:N \l__zrefclever_lang_decl_case_tl
564 \seq_new:N \l__zrefclever_lang_declension_seq
565 \seq_new:N \l__zrefclever_lang_gender_seq

```

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

`zrefclever_rf_opts_tl_not_type_specific_seq` 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.

```

566 \seq_new:N \g__zrefclever_rf_opts_tl_not_type_specific_seq
567 \seq_gset_from_clist:Nn
568   \g__zrefclever_rf_opts_tl_not_type_specific_seq
569   {
570     tpairsep ,
571     tlistsep ,
572     tlastsep ,
573     notesep ,
574   }
575 \seq_new:N \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
576 \seq_gset_from_clist:Nn
577   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
578   {
579     namesep ,
580     pairsep ,
581     listsep ,
582     lastsep ,
583     rangesep ,
584     namefont ,
585     reffont ,
586   }
587 \seq_new:N \g__zrefclever_rf_opts_seq_refbounds_seq
588 \seq_gset_from_clist:Nn
589   \g__zrefclever_rf_opts_seq_refbounds_seq
590   {
591     refbounds-first ,
592     refbounds-first-sg ,
593     refbounds-first-pb ,
594     refbounds-first-rb ,
595     refbounds-mid ,
596     refbounds-mid-rb ,

```

```

597     refbounds-mid-re ,
598     refbounds-last ,
599     refbounds-last-pe ,
600     refbounds-last-re ,
601   }
602 \seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
603 \seq_gset_from_clist:Nn
604   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
605   {
606     cap ,
607     abbrev ,
608     rangetopair ,
609   }

```

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:`.

```

610 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
611 \seq_gset_from_clist:Nn
612   \g__zrefclever_rf_opts_tl_type_names_seq
613   {
614     Name-sg ,
615     name-sg ,
616     Name-pl ,
617     name-pl ,
618     Name-sg-ab ,
619     name-sg-ab ,
620     Name-pl-ab ,
621     name-pl-ab ,
622   }

```

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

```

623 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
624 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
625   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
626   \g__zrefclever_rf_opts_tl_type_names_seq
627 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
628 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
629   \g__zrefclever_rf_opts_tl_not_type_specific_seq
630   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

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

We set here also the “derived” `refbounds` options, which are (almost) the same for every option scope.

```

631 \clist_map_inline:nn
632   {
633     reference ,
634     typesetup ,
635     langsetup ,
636     langfile ,
637   }
638   {
639     \keys_define:nn { zref-clever/ #1 }
640     {

```

```

641 +refbounds-first .meta:n =
642 {
643   refbounds-first = {##1} ,
644   refbounds-first-sg = {##1} ,
645   refbounds-first-pb = {##1} ,
646   refbounds-first-rb = {##1} ,
647 } ,
648 +refbounds-mid .meta:n =
649 {
650   refbounds-mid = {##1} ,
651   refbounds-mid-rb = {##1} ,
652   refbounds-mid-re = {##1} ,
653 } ,
654 +refbounds-last .meta:n =
655 {
656   refbounds-last = {##1} ,
657   refbounds-last-pe = {##1} ,
658   refbounds-last-re = {##1} ,
659 } ,
660 +refbounds-rb .meta:n =
661 {
662   refbounds-first-rb = {##1} ,
663   refbounds-mid-rb = {##1} ,
664 } ,
665 +refbounds-re .meta:n =
666 {
667   refbounds-mid-re = {##1} ,
668   refbounds-last-re = {##1} ,
669 } ,
670 +refbounds .meta:n =
671 {
672   +refbounds-first = {##1} ,
673   +refbounds-mid = {##1} ,
674   +refbounds-last = {##1} ,
675 } ,
676 refbounds .meta:n = { +refbounds = {##1} } ,
677 }
678 }
679 \clist_map_inline:nn
680 {
681   reference ,
682   typesetup ,
683 }
684 {
685   \keys_define:nn { zref-clever/ #1 }
686   {
687     +refbounds-first .default:o = \c_novalue_tl ,
688     +refbounds-mid .default:o = \c_novalue_tl ,
689     +refbounds-last .default:o = \c_novalue_tl ,
690     +refbounds-rb .default:o = \c_novalue_tl ,
691     +refbounds-re .default:o = \c_novalue_tl ,
692     +refbounds .default:o = \c_novalue_tl ,
693     refbounds .default:o = \c_novalue_tl ,
694   }

```

```

695 }
696 \clist_map_inline:nn
697 {
698   langsetup ,
699   langfile ,
700 }
701 {
702   \keys_define:nn { zref-clever/ #1 }
703   {
704     +refbounds-first .value_required:n = true ,
705     +refbounds-mid .value_required:n = true ,
706     +refbounds-last .value_required:n = true ,
707     +refbounds-rb .value_required:n = true ,
708     +refbounds-re .value_required:n = true ,
709     +refbounds .value_required:n = true ,
710     refbounds .value_required:n = true ,
711   }
712 }

```

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.

```

713 \tl_new:N \l__zrefclever_ref_language_tl
714 \tl_new:N \l__zrefclever_current_language_tl
715 \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.

```

716 \tl_new:N \l_zrefclever_ref_language_tl
717 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```

(End definition for `\l_zrefclever_ref_language_tl`. This function is documented on page ??.)

`_zrefclever_language_varname:n` Defines, and leaves in the input stream, the csname of the variable used to store the *⟨base language⟩* (as the value of this variable) for a *⟨language⟩* declared for zref-clever.

```

       \__zrefclever_language_varname:n {⟨language⟩}
718 \cs_new:Npn \__zrefclever_language_varname:n #1
719   { g_zrefclever_declared_language_ #1 _tl }

```

(End definition for `_zrefclever_language_varname:n`.)

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

```

720 \cs_set_eq:NN \zrefclever_language_varname:n
721   \__zrefclever_language_varname:n

```

(End definition for `\zrefclever_language_varname:n`. This function is documented on page ??.)

`_zrefclever_language_if_declared:nTF` A language is considered to be declared for zref-clever if it passes this conditional, which requires that a variable with `_zrefclever_language_varname:n{<language>}` exists.

```

\zrefclever_language_if_declared:nTF {<language>}
722 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
723 {
724   \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
725   { \prg_return_true: }
726   { \prg_return_false: }
727 }
728 \prg_generate_conditional_variant:Nnn
729 \_zrefclever_language_if_declared:n { x } { T , F , TF }

(End definition for \_zrefclever_language_if_declared:nTF.)

```

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

```

730 \prg_set_eq_conditional:NnN \zrefclever_language_if_declared:n
731 \_zrefclever_language_if_declared:n { TF }

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

```

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

```

\zcDeclareLanguage [<options>] {<language>}

732 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
733 {
734   \group_begin:
735   \tl_if_empty:nF {#2}
736   {
737     \_zrefclever_language_if_declared:nTF {#2}
738     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
739     {
740       \tl_new:c { \_zrefclever_language_varname:n {#2} }
741       \tl_gset:cn { \_zrefclever_language_varname:n {#2} } {#2}
742       \tl_set:Nn \l_zrefclever_setup_language_tl {#2}
743       \keys_set:nn { zref-clever/declarelang } {#1}
744     }
745   }

```

```

746     \group_end:
747   }
748 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

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

```

\zcDeclareLanguageAlias {⟨language alias⟩} {⟨aliased language⟩}

749 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
750 {
751   \tl_if_empty:nF {#1}
752   {
753     \__zrefclever_language_if_declared:nTF {#2}
754     {
755       \tl_new:c { \__zrefclever_language_varname:n {#1} }
756       \tl_gset:cx { \__zrefclever_language_varname:n {#1} }
757       { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
758     }
759     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
760   }
761 }
762 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

763 \keys_define:nn { zref-clever/declarelang }
764 {
765   declension .code:n =
766   {
767     \seq_new:c
768     {
769       \__zrefclever_opt_varname_language:enn
770       { \l__zrefclever_setup_language_tl } { declension } { seq }
771     }
772     \seq_gset_from_clist:cn
773     {
774       \__zrefclever_opt_varname_language:enn
775       { \l__zrefclever_setup_language_tl } { declension } { seq }
776     }
777     {#1}
778   } ,
779   declension .value_required:n = true ,
780   gender .code:n =
781   {
782     \seq_new:c
783     {
784       \__zrefclever_opt_varname_language:enn
785       { \l__zrefclever_setup_language_tl } { gender } { seq }
786     }
787     \seq_gset_from_clist:cn
788     {
789       \__zrefclever_opt_varname_language:enn

```

```

790         { \l__zrefclever_setup_language_tl } { gender } { seq }
791     }
792     {#1}
793 } ,
794 gender .value_required:n = true ,
795 allcaps .choices:nn =
796 { true , false }
797 {
798     \bool_new:c
799     {
800         \__zrefclever_opt_varname_language:enn
801         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
802     }
803     \use:c { bool_gset_ \l_keys_choice_tl :c }
804     {
805         \__zrefclever_opt_varname_language:enn
806         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
807     }
808 } ,
809 allcaps .default:n = true ,
810 }

```

`__zrefclever_process_language_settings:`

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

```

811 \cs_new_protected:Npn \__zrefclever_process_language_settings:
812 {
813     \__zrefclever_language_if_declared:xF
814     { \l__zrefclever_ref_language_tl }
815     {

```

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

```

816         \__zrefclever_opt_seq_get:cNF
817         {
818             \__zrefclever_opt_varname_language:enn
819             { \l__zrefclever_ref_language_tl } { declension } { seq }
820         }
821         \l__zrefclever_lang_declension_seq
822         { \seq_clear:N \l__zrefclever_lang_declension_seq }
823     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
824     {
825         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
826         {

```

```

827         \msg_warning:nxxx { zref-clever }
828         { language-no-decl-ref }
829         { \l__zrefclever_ref_language_tl }
830         { \l__zrefclever_ref_decl_case_tl }
831         \tl_clear:N \l__zrefclever_ref_decl_case_tl
832     }
833 }
834 {
835     \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
836     {
837         \seq_get_left:NN \l__zrefclever_lang_declension_seq
838         \l__zrefclever_ref_decl_case_tl
839     }
840     {
841         \seq_if_in:NVF \l__zrefclever_lang_declension_seq
842         \l__zrefclever_ref_decl_case_tl
843         {
844             \msg_warning:nxxx { zref-clever }
845             { unknown-decl-case }
846             { \l__zrefclever_ref_decl_case_tl }
847             { \l__zrefclever_ref_language_tl }
848             \seq_get_left:NN \l__zrefclever_lang_declension_seq
849             \l__zrefclever_ref_decl_case_tl
850         }
851     }
852 }

```

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.

```

853     \__zrefclever_opt_seq_get:cNF
854     {
855         \__zrefclever_opt_varname_language:enn
856         { \l__zrefclever_ref_language_tl } { gender } { seq }
857     }
858     \l__zrefclever_lang_gender_seq
859     { \seq_clear:N \l__zrefclever_lang_gender_seq }
860     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
861     {
862         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
863         {
864             \msg_warning:nxxxx { zref-clever }
865             { language-no-gender }
866             { \l__zrefclever_ref_language_tl }
867             { g }
868             { \l__zrefclever_ref_gender_tl }
869             \tl_clear:N \l__zrefclever_ref_gender_tl
870         }
871     }
872     {
873         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
874         {
875             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
876             \l__zrefclever_ref_gender_tl

```

```

877         {
878             \msg_warning:nxxx { zref-clever }
879             { gender-not-declared }
880             { \l__zrefclever_ref_language_tl }
881             { \l__zrefclever_ref_gender_tl }
882             \tl_clear:N \l__zrefclever_ref_gender_tl
883         }
884     }
885 }

```

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

```

886 \__zrefclever_opt_bool_if:cT
887 {
888     \__zrefclever_opt_varname_language:enn
889     { \l__zrefclever_ref_language_tl } { allcaps } { bool }
890 }
891 { \keys_set:mn { zref-clever/reference } { cap = true } }
892 }
893 {

```

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

```

894 \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
895 {
896     \msg_warning:nxxx { zref-clever } { unknown-language-decl }
897     { \l__zrefclever_ref_decl_case_tl }
898     { \l__zrefclever_ref_language_tl }
899     \tl_clear:N \l__zrefclever_ref_decl_case_tl
900 }
901 \tl_if_empty:NF \l__zrefclever_ref_gender_tl
902 {
903     \msg_warning:nxxxx { zref-clever }
904     { language-no-gender }
905     { \l__zrefclever_ref_language_tl }
906     { g }
907     { \l__zrefclever_ref_gender_tl }
908     \tl_clear:N \l__zrefclever_ref_gender_tl
909 }
910 }
911 }

```

(End definition for `__zrefclever_process_language_settings:.`)

4.7 Language files

Contrary to general options and type options, which are always *local*, language-specific settings are always *global*. Hence, the loading of built-in language files, as well as settings done with `\zcLanguageSetup`, should set the relevant variables globally.

The built-in language files and their related infrastructure are designed to perform “on the fly” loading of the language files, “lazily” as needed. Much like `babel` does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of parsimony, of “loading the least possible”. Therefore, we load at `begindocument` one

single language (see `lang option`), as specified by the user in the preamble with the `lang` option or, failing any specification, the current language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

This design decision has also implications to the *form* the language files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`. This includes `translator`, `translations`, but also `babel`'s `.ldf` files, and `biblatex`'s `.ltx` files. I'm not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`'s "on the fly" functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble "configuration files" of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load language files on the fly robustly mid-document, without getting paranoid with the last bit of white-space in them, and without introducing any undue content on the stream when we cannot afford to do it. Hence, `zref-clever`'s built-in language files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/langfile}` by `__zrefclever_provide_langfile:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The language file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

`__zrefclever_provide_langfile:n` is only meant to load the built-in language files. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a corresponding variables. Hence, there is no need to "load" anything in this case: definitions and assignments made by the user are performed immediately.

`\g__zrefclever_loaded_langfiles_seq` Used to keep track of whether a language file has already been loaded or not.

```
912 \seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(End definition for `\g__zrefclever_loaded_langfiles_seq`.)

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

```
\__zrefclever_provide_langfile:n {<language>}
```

```
913 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
914 {
915   \group_begin:
916   \@bsphack
917   \__zrefclever_language_if_declared:nT {#1}
918   {
919     \seq_if_in:NxF
920     \g__zrefclever_loaded_langfiles_seq
921     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
922     {
923       \exp_args:Nx \file_get:nnNTF
924       {
925         zref-clever-
```

```

926         \tl_use:c { \__zrefclever_language_varname:n {#1} }
927         .lang
928     }
929     { \ExplSyntaxOn }
930     \l_tmpa_tl
931     {
932         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
933         \tl_clear:N \l__zrefclever_setup_type_tl
934         \__zrefclever_opt_seq_get:cNF
935         {
936             \__zrefclever_opt_varname_language:nnn
937             {#1} { declension } { seq }
938         }
939         \l__zrefclever_lang_declension_seq
940         { \seq_clear:N \l__zrefclever_lang_declension_seq }
941         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
942         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
943         {
944             \seq_get_left:NN \l__zrefclever_lang_declension_seq
945             \l__zrefclever_lang_decl_case_tl
946         }
947         \__zrefclever_opt_seq_get:cNF
948         {
949             \__zrefclever_opt_varname_language:nnn
950             {#1} { gender } { seq }
951         }
952         \l__zrefclever_lang_gender_seq
953         { \seq_clear:N \l__zrefclever_lang_gender_seq }
954         \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
955         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
956         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
957         \msg_info:nnx { zref-clever } { langfile-loaded }
958         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
959     }
960     {

```

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.

```

961         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
962         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
963     }
964 }
965 }
966 \@esphack
967 \group_end:
968 }
969 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }

```

(End definition for __zrefclever_provide_langfile:n.)

The set of keys for `zref-clever/langfile`, which is used to process the language files in `__zrefclever_provide_langfile:n`. The no-op cases for each category have their messages sent to "info". These messages should not occur, as long as the language

files are well formed, but they're placed there nevertheless, and can be leveraged in regression tests.

```

970 \keys_define:nn { zref-clever/langfile }
971 {
972   type .code:n =
973   {
974     \tl_if_empty:nTF {#1}
975     { \tl_clear:N \l__zrefclever_setup_type_tl }
976     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
977   } ,
978
979   case .code:n =
980   {
981     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
982     {
983       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
984       { \l__zrefclever_setup_language_tl } {#1}
985     }
986     {
987       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
988       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
989       {
990         \msg_info:nxxx { zref-clever } { unknown-decl-case }
991         {#1} { \l__zrefclever_setup_language_tl }
992         \seq_get_left:NN \l__zrefclever_lang_declension_seq
993         \l__zrefclever_lang_decl_case_tl
994       }
995     }
996   } ,
997   case .value_required:n = true ,
998
999   gender .value_required:n = true ,
1000   gender .code:n =
1001   {
1002     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1003     {
1004       \msg_info:nxxxx { zref-clever } { language-no-gender }
1005       { \l__zrefclever_setup_language_tl } { gender } {#1}
1006     }
1007     {
1008       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1009       {
1010         \msg_info:nnn { zref-clever }
1011         { option-only-type-specific } { gender }
1012       }
1013       {
1014         \seq_clear:N \l_tmpa_seq
1015         \clist_map_inline:nn {#1}
1016         {
1017           \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1018           { \seq_put_right:Nn \l_tmpa_seq {##1} }
1019           {
1020             \msg_info:nxxx { zref-clever }
1021             { gender-not-declared }

```



```

1022         { \l__zrefclever_setup_language_tl } {##1}
1023     }
1024 }
1025 \__zrefclever_opt_seq_if_set:cF
1026 {
1027     \__zrefclever_opt_varname_lang_type:eenn
1028     { \l__zrefclever_setup_language_tl }
1029     { \l__zrefclever_setup_type_tl }
1030     { gender }
1031     { seq }
1032 }
1033 {
1034     \seq_new:c
1035     {
1036         \__zrefclever_opt_varname_lang_type:eenn
1037         { \l__zrefclever_setup_language_tl }
1038         { \l__zrefclever_setup_type_tl }
1039         { gender }
1040         { seq }
1041     }
1042     \seq_gset_eq:cN
1043     {
1044         \__zrefclever_opt_varname_lang_type:eenn
1045         { \l__zrefclever_setup_language_tl }
1046         { \l__zrefclever_setup_type_tl }
1047         { gender }
1048         { seq }
1049     }
1050     \l_tmpa_seq
1051 }
1052 }
1053 } ,
1054 } ,
1055 }
1056 \seq_map_inline:Nn
1057 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1058 {
1059     \keys_define:nn { zref-clever/langfile }
1060     {
1061         #1 .value_required:n = true ,
1062         #1 .code:n =
1063         {
1064             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1065             {
1066                 \__zrefclever_opt_tl_gset_if_new:cn
1067                 {
1068                     \__zrefclever_opt_varname_lang_default:enn
1069                     { \l__zrefclever_setup_language_tl }
1070                     {##1} { tl }
1071                 }
1072                 {##1}
1073             }
1074         }
1075         \msg_info:nnn { zref-clever }

```

```

1076         { option-not-type-specific } {#1}
1077     }
1078 } ,
1079 }
1080 }
1081 \seq_map_inline:Nn
1082 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1083 {
1084     \keys_define:nn { zref-clever/langfile }
1085     {
1086         #1 .value_required:n = true ,
1087         #1 .code:n =
1088         {
1089             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1090             {
1091                 \__zrefclever_opt_tl_gset_if_new:cn
1092                 {
1093                     \__zrefclever_opt_varname_lang_default:enn
1094                     { \l__zrefclever_setup_language_tl }
1095                     {#1} { tl }
1096                 }
1097                 {##1}
1098             }
1099             {
1100                 \__zrefclever_opt_tl_gset_if_new:cn
1101                 {
1102                     \__zrefclever_opt_varname_lang_type:eenn
1103                     { \l__zrefclever_setup_language_tl }
1104                     { \l__zrefclever_setup_type_tl }
1105                     {#1} { tl }
1106                 }
1107                 {##1}
1108             }
1109         } ,
1110     }
1111 }
1112 \keys_define:nn { zref-clever/langfile }
1113 {
1114     endrange .value_required:n = true ,
1115     endrange .code:n =
1116     {
1117         \str_case:nnF {#1}
1118         {
1119             { ref }
1120             {
1121                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1122                 {
1123                     \__zrefclever_opt_tl_gclear_if_new:c
1124                     {
1125                         \__zrefclever_opt_varname_lang_default:enn
1126                         { \l__zrefclever_setup_language_tl }
1127                         { endrangefunc } { tl }
1128                     }
1129                     \__zrefclever_opt_tl_gclear_if_new:c

```

```

1130         {
1131             \__zrefclever_opt_varname_lang_default:enn
1132             { \l__zrefclever_setup_language_tl }
1133             { endrangeprop } { tl }
1134         }
1135     }
1136     {
1137         \__zrefclever_opt_tl_gclear_if_new:c
1138         {
1139             \__zrefclever_opt_varname_lang_type:eenn
1140             { \l__zrefclever_setup_language_tl }
1141             { \l__zrefclever_setup_type_tl }
1142             { endrangefunc } { tl }
1143         }
1144         \__zrefclever_opt_tl_gclear_if_new:c
1145         {
1146             \__zrefclever_opt_varname_lang_type:eenn
1147             { \l__zrefclever_setup_language_tl }
1148             { \l__zrefclever_setup_type_tl }
1149             { endrangeprop } { tl }
1150         }
1151     }
1152 }
1153
1154 { stripprefix }
1155 {
1156     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1157     {
1158         \__zrefclever_opt_tl_gset_if_new:cn
1159         {
1160             \__zrefclever_opt_varname_lang_default:enn
1161             { \l__zrefclever_setup_language_tl }
1162             { endrangefunc } { tl }
1163         }
1164         { __zrefclever_get_endrange_stripprefix }
1165         \__zrefclever_opt_tl_gclear_if_new:c
1166         {
1167             \__zrefclever_opt_varname_lang_default:enn
1168             { \l__zrefclever_setup_language_tl }
1169             { endrangeprop } { tl }
1170         }
1171     }
1172     {
1173         \__zrefclever_opt_tl_gset_if_new:cn
1174         {
1175             \__zrefclever_opt_varname_lang_type:eenn
1176             { \l__zrefclever_setup_language_tl }
1177             { \l__zrefclever_setup_type_tl }
1178             { endrangefunc } { tl }
1179         }
1180         { __zrefclever_get_endrange_stripprefix }
1181         \__zrefclever_opt_tl_gclear_if_new:c
1182         {
1183             \__zrefclever_opt_varname_lang_type:eenn

```

```

1184         { \l__zrefclever_setup_language_tl }
1185         { \l__zrefclever_setup_type_tl }
1186         { endrangeprop } { tl }
1187     }
1188 }
1189 }
1190
1191 { pagecomp }
1192 {
1193     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1194     {
1195         \__zrefclever_opt_tl_gset_if_new:cn
1196         {
1197             \__zrefclever_opt_varname_lang_default:enn
1198             { \l__zrefclever_setup_language_tl }
1199             { endrangefunc } { tl }
1200         }
1201         { __zrefclever_get_endrange_pagecomp }
1202         \__zrefclever_opt_tl_gclear_if_new:c
1203         {
1204             \__zrefclever_opt_varname_lang_default:enn
1205             { \l__zrefclever_setup_language_tl }
1206             { endrangeprop } { tl }
1207         }
1208     }
1209     {
1210         \__zrefclever_opt_tl_gset_if_new:cn
1211         {
1212             \__zrefclever_opt_varname_lang_type:eenn
1213             { \l__zrefclever_setup_language_tl }
1214             { \l__zrefclever_setup_type_tl }
1215             { endrangefunc } { tl }
1216         }
1217         { __zrefclever_get_endrange_pagecomp }
1218         \__zrefclever_opt_tl_gclear_if_new:c
1219         {
1220             \__zrefclever_opt_varname_lang_type:eenn
1221             { \l__zrefclever_setup_language_tl }
1222             { \l__zrefclever_setup_type_tl }
1223             { endrangeprop } { tl }
1224         }
1225     }
1226 }
1227
1228 { pagecomp2 }
1229 {
1230     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1231     {
1232         \__zrefclever_opt_tl_gset_if_new:cn
1233         {
1234             \__zrefclever_opt_varname_lang_default:enn
1235             { \l__zrefclever_setup_language_tl }
1236             { endrangefunc } { tl }
1237         }

```

```

1238         { __zrefclever_get_endrange_pagecomptwo }
1239     \__zrefclever_opt_tl_gclear_if_new:c
1240     {
1241         \__zrefclever_opt_varname_lang_default:enn
1242         { \l__zrefclever_setup_language_tl }
1243         { endrangeprop } { tl }
1244     }
1245 }
1246 {
1247     \__zrefclever_opt_tl_gset_if_new:cn
1248     {
1249         \__zrefclever_opt_varname_lang_type:eenn
1250         { \l__zrefclever_setup_language_tl }
1251         { \l__zrefclever_setup_type_tl }
1252         { endrangefunc } { tl }
1253     }
1254     { __zrefclever_get_endrange_pagecomptwo }
1255     \__zrefclever_opt_tl_gclear_if_new:c
1256     {
1257         \__zrefclever_opt_varname_lang_type:eenn
1258         { \l__zrefclever_setup_language_tl }
1259         { \l__zrefclever_setup_type_tl }
1260         { endrangeprop } { tl }
1261     }
1262 }
1263 }
1264 }
1265 {
1266     \tl_if_empty:nTF {#1}
1267     {
1268         \msg_info:nnn { zref-clever }
1269         { endrange-property-undefined } {#1}
1270     }
1271     {
1272         \zref@ifpropundefined {#1}
1273         {
1274             \msg_info:nnn { zref-clever }
1275             { endrange-property-undefined } {#1}
1276         }
1277         {
1278             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1279             {
1280                 \__zrefclever_opt_tl_gset_if_new:cn
1281                 {
1282                     \__zrefclever_opt_varname_lang_default:enn
1283                     { \l__zrefclever_setup_language_tl }
1284                     { endrangefunc } { tl }
1285                 }
1286                 { __zrefclever_get_endrange_property }
1287                 \__zrefclever_opt_tl_gset_if_new:cn
1288                 {
1289                     \__zrefclever_opt_varname_lang_default:enn
1290                     { \l__zrefclever_setup_language_tl }
1291                     { endrangeprop } { tl }

```

```

1292         }
1293         {#1}
1294     }
1295     {
1296         \__zrefclever_opt_tl_gset_if_new:cn
1297         {
1298             \__zrefclever_opt_varname_lang_type:eenn
1299             { \l__zrefclever_setup_language_tl }
1300             { \l__zrefclever_setup_type_tl }
1301             { endrangefunc } { tl }
1302         }
1303         { __zrefclever_get_endrange_property }
1304         \__zrefclever_opt_tl_gset_if_new:cn
1305         {
1306             \__zrefclever_opt_varname_lang_type:eenn
1307             { \l__zrefclever_setup_language_tl }
1308             { \l__zrefclever_setup_type_tl }
1309             { endrangeprop } { tl }
1310         }
1311         {#1}
1312     }
1313 }
1314 }
1315 }
1316 } ,
1317 }
1318 \seq_map_inline:Nn
1319 \g__zrefclever_rf_opts_tl_type_names_seq
1320 {
1321     \keys_define:nm { zref-clever/langfile }
1322     {
1323         #1 .value_required:n = true ,
1324         #1 .code:n =
1325         {
1326             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1327             {
1328                 \msg_info:nnn { zref-clever }
1329                 { option-only-type-specific } {#1}
1330             }
1331             {
1332                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1333                 {
1334                     \__zrefclever_opt_tl_gset_if_new:cn
1335                     {
1336                         \__zrefclever_opt_varname_lang_type:eenn
1337                         { \l__zrefclever_setup_language_tl }
1338                         { \l__zrefclever_setup_type_tl }
1339                         {#1} { tl }
1340                     }
1341                     {##1}
1342                 }
1343                 {
1344                     \__zrefclever_opt_tl_gset_if_new:cn
1345                     {

```

```

1346         \_zrefclever_opt_varname_lang_type:eeen
1347         { \l_zrefclever_setup_language_tl }
1348         { \l_zrefclever_setup_type_tl }
1349         { \l_zrefclever_lang_decl_case_tl - #1 } { tl }
1350     }
1351     {##1}
1352 }
1353 }
1354 } ,
1355 }
1356 }
1357 \seq_map_inline:Nn
1358 \g_zrefclever_rf_opts_seq_refbounds_seq
1359 {
1360     \keys_define:nm { zref-clever/langfile }
1361     {
1362         #1 .value_required:n = true ,
1363         #1 .code:n =
1364         {
1365             \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1366             {
1367                 \_zrefclever_opt_seq_if_set:cF
1368                 {
1369                     \_zrefclever_opt_varname_lang_default:enn
1370                     { \l_zrefclever_setup_language_tl } {#1} { seq }
1371                 }
1372                 {
1373                     \seq_gclear:N \g_tmpa_seq
1374                     \_zrefclever_opt_seq_gset_clist_split:Nn
1375                     \g_tmpa_seq {##1}
1376                     \bool_lazy_or:nnTF
1377                     { \tl_if_empty_p:n {##1} }
1378                     {
1379                         \int_compare_p:nNn
1380                         { \seq_count:N \g_tmpa_seq } = { 4 }
1381                     }
1382                     {
1383                         \_zrefclever_opt_seq_gset_eq:cN
1384                         {
1385                             \_zrefclever_opt_varname_lang_default:enn
1386                             { \l_zrefclever_setup_language_tl }
1387                             {#1} { seq }
1388                         }
1389                         \g_tmpa_seq
1390                     }
1391                     {
1392                         \msg_info:nxxx { zref-clever }
1393                         { refbounds-must-be-four }
1394                         {#1} { \seq_count:N \g_tmpa_seq }
1395                     }
1396                 }
1397             }
1398         }
1399         \_zrefclever_opt_seq_if_set:cF

```

```

1400     {
1401         \__zrefclever_opt_varname_lang_type:eenn
1402         { \l__zrefclever_setup_language_tl }
1403         { \l__zrefclever_setup_type_tl } {#1} { seq }
1404     }
1405     {
1406         \seq_gclear:N \g_tmpa_seq
1407         \__zrefclever_opt_seq_gset_clist_split:Nn
1408         \g_tmpa_seq {##1}
1409         \bool_lazy_or:nnTF
1410         { \tl_if_empty_p:n {##1} }
1411         {
1412             \int_compare_p:nNn
1413             { \seq_count:N \g_tmpa_seq } = { 4 }
1414         }
1415         {
1416             \__zrefclever_opt_seq_gset_eq:cN
1417             {
1418                 \__zrefclever_opt_varname_lang_type:eenn
1419                 { \l__zrefclever_setup_language_tl }
1420                 { \l__zrefclever_setup_type_tl }
1421                 {#1} { seq }
1422             }
1423             \g_tmpa_seq
1424         }
1425         {
1426             \msg_info:nxxx { zref-clever }
1427             { refbounds-must-be-four }
1428             {#1} { \seq_count:N \g_tmpa_seq }
1429         }
1430     }
1431     }
1432 } ,
1433 }
1434 }
1435 \seq_map_inline:Nn
1436 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1437 {
1438     \keys_define:nn { zref-clever/langfile }
1439     {
1440         #1 .choice: ,
1441         #1 / true .code:n =
1442         {
1443             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1444             {
1445                 \__zrefclever_opt_bool_if_set:cF
1446                 {
1447                     \__zrefclever_opt_varname_lang_default:enn
1448                     { \l__zrefclever_setup_language_tl }
1449                     {#1} { bool }
1450                 }
1451                 {
1452                     \__zrefclever_opt_bool_gset_true:c
1453                     {

```



```

1454         \_zrefclever_opt_varname_lang_default:enn
1455         { \l__zrefclever_setup_language_tl }
1456         {#1} { bool }
1457     }
1458 }
1459 }
1460 {
1461     \_zrefclever_opt_bool_if_set:cF
1462     {
1463         \_zrefclever_opt_varname_lang_type:eenn
1464         { \l__zrefclever_setup_language_tl }
1465         { \l__zrefclever_setup_type_tl }
1466         {#1} { bool }
1467     }
1468     {
1469         \_zrefclever_opt_bool_gset_true:c
1470         {
1471             \_zrefclever_opt_varname_lang_type:eenn
1472             { \l__zrefclever_setup_language_tl }
1473             { \l__zrefclever_setup_type_tl }
1474             {#1} { bool }
1475         }
1476     }
1477 }
1478 } ,
1479 #1 / false .code:n =
1480 {
1481     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1482     {
1483         \_zrefclever_opt_bool_if_set:cF
1484         {
1485             \_zrefclever_opt_varname_lang_default:enn
1486             { \l__zrefclever_setup_language_tl }
1487             {#1} { bool }
1488         }
1489         {
1490             \_zrefclever_opt_bool_gset_false:c
1491             {
1492                 \_zrefclever_opt_varname_lang_default:enn
1493                 { \l__zrefclever_setup_language_tl }
1494                 {#1} { bool }
1495             }
1496         }
1497     }
1498 }
1499 \_zrefclever_opt_bool_if_set:cF
1500 {
1501     \_zrefclever_opt_varname_lang_type:eenn
1502     { \l__zrefclever_setup_language_tl }
1503     { \l__zrefclever_setup_type_tl }
1504     {#1} { bool }
1505 }
1506 {
1507     \_zrefclever_opt_bool_gset_false:c

```

```

1508         {
1509             \__zrefclever_opt_varname_lang_type:eenn
1510             { \l__zrefclever_setup_language_tl }
1511             { \l__zrefclever_setup_type_tl }
1512             {#1} { bool }
1513         }
1514     }
1515 }
1516 } ,
1517 #1 .default:n = true ,
1518 no #1 .meta:n = { #1 = false } ,
1519 no #1 .value_forbidden:n = true ,
1520 }
1521 }

```

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.

```

1522 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1523 {
1524     \tl_const:cn
1525     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1526 }
1527 \keyval_parse:nnn
1528 { }
1529 { \__zrefclever_opt_tl_cset_fallback:nn }
1530 {
1531     tpairsep = {,~} ,
1532     tlistsep = {,~} ,
1533     tlastsep = {,~} ,
1534     notesep = {~} ,
1535     namesep = {\nobreakspace} ,
1536     pairsep = {,~} ,
1537     listsep = {,~} ,
1538     lastsep = {,~} ,
1539     rangeseq = {\textendash} ,
1540 }

```

4.8 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}
1541 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1542 {
1543     \tl_if_empty:nTF {#3}
1544     { \prop_remove:Nn #1 {#2} }

```

```

1545     { \prop_put:Nnn #1 {#2} {#3} }
1546   }

```

(End definition for `_zrefclever_prop_put_non_empty:Nnn`.)

ref option

`\l_zrefclever_ref_property_tl` stores the property to which the reference is being made. Note that one thing *must* be handled at this point: the existence of the property itself, as far as `zref` is concerned. This because typesetting relies on the check `\zref@ifrefcontainsprop`, which *presumes* the property is defined and silently expands the *true* branch if it is not (insightful comments by Ulrike Fischer at <https://github.com/ho-tex/zref/issues/13>). Therefore, before adding anything to `\l_zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door. We must also control for an empty value, since “empty” passes both `\zref@ifpropundefined` and `\zref@ifrefcontainsprop`.

```

1547 \tl_new:N \l_zrefclever_ref_property_tl
1548 \keys_define:nn { zref-clever/reference }
1549 {
1550   ref .code:n =
1551   {
1552     \tl_if_empty:nTF {#1}
1553     {
1554       \msg_warning:nnn { zref-clever }
1555       { zref-property-undefined } {#1}
1556       \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1557     }
1558     {
1559       \zref@ifpropundefined {#1}
1560       {
1561         \msg_warning:nnn { zref-clever }
1562         { zref-property-undefined } {#1}
1563         \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1564       }
1565       { \tl_set:Nn \l_zrefclever_ref_property_tl {#1} }
1566     }
1567   } ,
1568   ref .initial:n = default ,
1569   ref .value_required:n = true ,
1570   page .meta:n = { ref = page },
1571   page .value_forbidden:n = true ,
1572 }

```

typeset option

```

1573 \bool_new:N \l_zrefclever_typeset_ref_bool
1574 \bool_new:N \l_zrefclever_typeset_name_bool
1575 \keys_define:nn { zref-clever/reference }
1576 {
1577   typeset .choice: ,
1578   typeset / both .code:n =
1579   {
1580     \bool_set_true:N \l_zrefclever_typeset_ref_bool

```

```

1581     \bool_set_true:N \l__zrefclever_typeset_name_bool
1582   } ,
1583 typeset / ref .code:n =
1584   {
1585     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1586     \bool_set_false:N \l__zrefclever_typeset_name_bool
1587   } ,
1588 typeset / name .code:n =
1589   {
1590     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1591     \bool_set_true:N \l__zrefclever_typeset_name_bool
1592   } ,
1593 typeset .initial:n = both ,
1594 typeset .value_required:n = true ,
1595
1596 noname .meta:n = { typeset = ref } ,
1597 noname .value_forbidden:n = true ,
1598 noref .meta:n = { typeset = name } ,
1599 noref .value_forbidden:n = true ,
1600 }

```

sort option

```

1601 \bool_new:N \l__zrefclever_typeset_sort_bool
1602 \keys_define:nn { zref-clever/reference }
1603 {
1604   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1605   sort .initial:n = true ,
1606   sort .default:n = true ,
1607   nosort .meta:n = { sort = false } ,
1608   nosort .value_forbidden:n = true ,
1609 }

```

typesort option

\l__zrefclever_typesort_seq is stored reversed, since the sort priorities are computed in the negative range in \l__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.

```

1610 \seq_new:N \l__zrefclever_typesort_seq
1611 \keys_define:nn { zref-clever/reference }
1612 {
1613   typesort .code:n =
1614   {
1615     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1616     \seq_reverse:N \l__zrefclever_typesort_seq
1617   } ,
1618   typesort .initial:n =
1619   { part , chapter , section , paragraph } ,
1620   typesort .value_required:n = true ,
1621   notypesort .code:n =
1622   { \seq_clear:N \l__zrefclever_typesort_seq } ,
1623   notypesort .value_forbidden:n = true ,
1624 }

```

comp option

```
1625 \bool_new:N \l__zrefclever_typeset_compress_bool
1626 \keys_define:nn { zref-clever/reference }
1627 {
1628   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1629   comp .initial:n = true ,
1630   comp .default:n = true ,
1631   nocomp .meta:n = { comp = false },
1632   nocomp .value_forbidden:n = true ,
1633 }
```

endrange option

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

`endrangefunc` *must* receive three arguments and, more specifically, its signature *must* be `VVN`. For this reason, `endrangefunc` should be stored without the signature, which is added, and hard-coded, at the calling place. The first argument is *⟨begin range label⟩*, the second *⟨end range label⟩*, and the last *⟨tl var to set⟩*. Of course, *⟨tl var to set⟩* must be set to a proper value, and that's the main task of the function. `endrangefunc` must also handle the case where `\zref@ifrefcontainsprop` is false, since `__zrefclever_get_ref_endrange:nnN` cannot take care of that. For this purpose, it may set *⟨tl var to set⟩* to the special value `zc@missingproperty`, to signal a missing property for `__zrefclever_get_ref_endrange:nnN`.

An empty `endrangefunc` signals that no processing is to be made to the end range reference, that is, that it should be treated like any other one, as defined by the `ref` option. This may happen either because `endrange` was never set for the reference type, and empty is the value “returned” by `__zrefclever_get_rf_opt_tl:nnnN` for options not set, or because `endrange` was set to `ref` at some scope which happens to get precedence.

One thing I was divided about in this functionality was whether to (x-)expand the references before processing them, when such processing is required. At first sight, it makes sense to do so, since we are aiming at “removing common parts” as close as possible to the printed representation of the references (`cleveref` does expand them in `\crefstriprefix`). On the other hand, this brings some new challenges: if a fragile command gets there, we are in trouble; also, if a protected one gets there, though things won't break as badly, we may “strip” the macro and stay with different arguments, which will then end up in the input stream. I think `biblatex` is a good reference here, and it offers `\NumCheckSetup`, `\NumsCheckSetup`, and `\PagesCheckSetup` aimed at locally redefining some commands which may interfere with the processing. This is a good idea, thus we offer a similar hook for the same purpose: `endrange-setup`.

```
1634 \NewHook { zref-clever/endrange-setup }
1635 \keys_define:nn { zref-clever/reference }
1636 {
1637   endrange .code:n =
1638     {
1639       \str_case:nnF {#1}
1640         {
1641           { ref }

```

```

1642 {
1643   \_zrefclever_opt_tl_clear:c
1644   {
1645     \_zrefclever_opt_varname_general:nn
1646     { endrangefunc } { tl }
1647   }
1648   \_zrefclever_opt_tl_clear:c
1649   {
1650     \_zrefclever_opt_varname_general:nn
1651     { endrangeprop } { tl }
1652   }
1653 }
1654
1655 { stripprefix }
1656 {
1657   \_zrefclever_opt_tl_set:cn
1658   {
1659     \_zrefclever_opt_varname_general:nn
1660     { endrangefunc } { tl }
1661   }
1662   { __zrefclever_get_endrange_stripprefix }
1663   \_zrefclever_opt_tl_clear:c
1664   {
1665     \_zrefclever_opt_varname_general:nn
1666     { endrangeprop } { tl }
1667   }
1668 }
1669
1670 { pagecomp }
1671 {
1672   \_zrefclever_opt_tl_set:cn
1673   {
1674     \_zrefclever_opt_varname_general:nn
1675     { endrangefunc } { tl }
1676   }
1677   { __zrefclever_get_endrange_pagecomp }
1678   \_zrefclever_opt_tl_clear:c
1679   {
1680     \_zrefclever_opt_varname_general:nn
1681     { endrangeprop } { tl }
1682   }
1683 }
1684
1685 { pagecomp2 }
1686 {
1687   \_zrefclever_opt_tl_set:cn
1688   {
1689     \_zrefclever_opt_varname_general:nn
1690     { endrangefunc } { tl }
1691   }
1692   { __zrefclever_get_endrange_pagecomptwo }
1693   \_zrefclever_opt_tl_clear:c
1694   {
1695     \_zrefclever_opt_varname_general:nn

```

```

1696         { endrangeprop } { t1 }
1697     }
1698 }
1699
1700 { unset }
1701 {
1702     \__zrefclever_opt_t1_unset:c
1703     {
1704         \__zrefclever_opt_varname_general:nn
1705         { endrangefunc } { t1 }
1706     }
1707     \__zrefclever_opt_t1_unset:c
1708     {
1709         \__zrefclever_opt_varname_general:nn
1710         { endrangeprop } { t1 }
1711     }
1712 }
1713 }
1714 {
1715     \tl_if_empty:nTF {#1}
1716     {
1717         \msg_warning:nnn { zref-clever }
1718         { endrange-property-undefined } {#1}
1719     }
1720     {
1721         \zref@ifpropundefined {#1}
1722         {
1723             \msg_warning:nnn { zref-clever }
1724             { endrange-property-undefined } {#1}
1725         }
1726         {
1727             \__zrefclever_opt_t1_set:cn
1728             {
1729                 \__zrefclever_opt_varname_general:nn
1730                 { endrangefunc } { t1 }
1731             }
1732             { __zrefclever_get_endrange_property }
1733             \__zrefclever_opt_t1_set:cn
1734             {
1735                 \__zrefclever_opt_varname_general:nn
1736                 { endrangeprop } { t1 }
1737             }
1738             {#1}
1739         }
1740     }
1741 }
1742 },
1743 endrange .value_required:n = true ,
1744 }
1745 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1746 {
1747     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1748     {
1749         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }

```

```

1750     {
1751         \zrefclever_extract_default:Nnvn #3
1752         {#2} { l_zrefclever_ref_property_tl } { }
1753     }
1754     { \tl_set:Nn #3 { zc@missingproperty } }
1755 }
1756 {
1757     \zref@ifrefcontainsprop {#2} { \l_zrefclever_endrangeprop_tl }
1758     {

```

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.

```

1759     \bool_if:NTF \l_zrefclever_typeset_range_bool
1760     {
1761         \group_begin:
1762         \bool_set_false:N \l_tmpa_bool
1763         \exp_args:Nxx \tl_if_eq:nnT
1764         {
1765             \zrefclever_extract_unexp:nnn
1766             {#1} { externaldocument } { }
1767         }
1768         {
1769             \zrefclever_extract_unexp:nnn
1770             {#2} { externaldocument } { }
1771         }
1772     }
1773     \tl_if_eq:NnTF \l_zrefclever_ref_property_tl { page }
1774     {
1775         \exp_args:Nxx \tl_if_eq:nnT
1776         {
1777             \zrefclever_extract_unexp:nnn
1778             {#1} { zc@pgfmt } { }
1779         }
1780         {
1781             \zrefclever_extract_unexp:nnn
1782             {#2} { zc@pgfmt } { }
1783         }
1784         { \bool_set_true:N \l_tmpa_bool }
1785     }
1786     {
1787         \exp_args:Nxx \tl_if_eq:nnT
1788         {
1789             \zrefclever_extract_unexp:nnn
1790             {#1} { zc@counter } { }
1791         }
1792         {
1793             \zrefclever_extract_unexp:nnn
1794             {#2} { zc@counter } { }
1795         }
1796     }
1797     \exp_args:Nxx \tl_if_eq:nnT

```



```

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

```

1844 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1845 {
1846     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1847     {
1848         \group_begin:

```

```

1849 \UseHook { zref-clever/endrange-setup }
1850 \tl_set:Nx \l_tmpa_tl
1851 {
1852   \__zrefclever_extract:nnn
1853   {#1} { \l__zrefclever_ref_property_tl } { }
1854 }
1855 \tl_set:Nx \l_tmpb_tl
1856 {
1857   \__zrefclever_extract:nnn
1858   {#2} { \l__zrefclever_ref_property_tl } { }
1859 }
1860 \bool_set_false:N \l_tmpa_bool
1861 \bool_until_do:Nn \l_tmpa_bool
1862 {
1863   \exp_args:Nxx \tl_if_eq:nnTF
1864   { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1865   {
1866     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1867     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1868     \tl_if_empty:NT \l_tmpb_tl
1869     { \bool_set_true:N \l_tmpa_bool }
1870   }
1871   { \bool_set_true:N \l_tmpa_bool }
1872 }
1873 \exp_args:NNNV
1874 \group_end:
1875 \tl_set:Nn #3 \l_tmpb_tl
1876 }
1877 { \tl_set:Nn #3 { zc@missingproperty } }
1878 }
1879 \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>).

```

1880 \prg_new_protected_conditional:Npnn
1881 \__zrefclever_is_integer_rgx:n #1 { F , TF }
1882 {
1883   \regex_match:nnTF { \A\d+\Z } {#1}
1884   { \prg_return_true: }
1885   { \prg_return_false: }
1886 }
1887 \prg_generate_conditional_variant:Nnn
1888 \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End definition for `__zrefclever_is_integer_rgx:n`)

```

1889 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1890 {
1891   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1892   {
1893     \group_begin:
1894     \UseHook { zref-clever/endrange-setup }
1895     \tl_set:Nx \l_tmpa_tl
1896     {
1897       \__zrefclever_extract:nnn

```

```

1898         {#1} { \l__zrefclever_ref_property_tl } { }
1899     }
1900     \tl_set:Nx \l_tmpb_tl
1901     {
1902         \__zrefclever_extract:nnn
1903         {#2} { \l__zrefclever_ref_property_tl } { }
1904     }
1905     \bool_set_false:N \l_tmpa_bool
1906     \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1907     {
1908         \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1909         { \bool_set_true:N \l_tmpa_bool }
1910     }
1911     { \bool_set_true:N \l_tmpa_bool }
1912     \bool_until_do:Nn \l_tmpa_bool
1913     {
1914         \exp_args:Nxx \tl_if_eq:nnTF
1915         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1916         {
1917             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1918             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1919             \tl_if_empty:NT \l_tmpb_tl
1920             { \bool_set_true:N \l_tmpa_bool }
1921         }
1922         { \bool_set_true:N \l_tmpa_bool }
1923     }
1924     \exp_args:NNNV
1925     \group_end:
1926     \tl_set:Nn #3 \l_tmpb_tl
1927 }
1928 { \tl_set:Nn #3 { zc@missingproperty } }
1929 }
1930 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1931 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1932 {
1933     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1934     {
1935         \group_begin:
1936         \UseHook { zref-clever/endorange-setup }
1937         \tl_set:Nx \l_tmpa_tl
1938         {
1939             \__zrefclever_extract:nnn
1940             {#1} { \l__zrefclever_ref_property_tl } { }
1941         }
1942         \tl_set:Nx \l_tmpb_tl
1943         {
1944             \__zrefclever_extract:nnn
1945             {#2} { \l__zrefclever_ref_property_tl } { }
1946         }
1947         \bool_set_false:N \l_tmpa_bool
1948         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1949         {
1950             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1951             { \bool_set_true:N \l_tmpa_bool }

```

```

1952     }
1953     { \bool_set_true:N \l_tmpa_bool }
1954 \bool_until_do:Nn \l_tmpa_bool
1955     {
1956     \exp_args:Nxx \tl_if_eq:nnTF
1957     { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1958     {
1959     \bool_lazy_or:nnTF
1960     { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1961     { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1962     {
1963     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1964     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1965     }
1966     { \bool_set_true:N \l_tmpa_bool }
1967     }
1968     { \bool_set_true:N \l_tmpa_bool }
1969     }
1970 \exp_args:NNNV
1971 \group_end:
1972 \tl_set:Nn #3 \l_tmpb_tl
1973 }
1974 { \tl_set:Nn #3 { zc@missingproperty } }
1975 }
1976 \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`.

```

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

```

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`.

```

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

```

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`.

```
1991 \bool_new:N \l_zrefclever_noabbrev_first_bool
1992 \keys_define:nn { zref-clever/reference }
1993 {
1994   noabbrevfirst .bool_set:N = \l_zrefclever_noabbrev_first_bool ,
1995   noabbrevfirst .initial:n = false ,
1996   noabbrevfirst .default:n = true ,
1997 }
```

S option

```
1998 \keys_define:nn { zref-clever/reference }
1999 {
2000   S .meta:n =
2001     { capfirst = {#1} , noabbrevfirst = {#1} } ,
2002   S .default:n = true ,
2003 }
```

hyperref option

```
2004 \bool_new:N \l_zrefclever_hyperlink_bool
2005 \bool_new:N \l_zrefclever_hyperref_warn_bool
2006 \keys_define:nn { zref-clever/reference }
2007 {
2008   hyperref .choice: ,
2009   hyperref / auto .code:n =
2010     {
2011       \bool_set_true:N \l_zrefclever_hyperlink_bool
2012       \bool_set_false:N \l_zrefclever_hyperref_warn_bool
2013     } ,
2014   hyperref / true .code:n =
2015     {
2016       \bool_set_true:N \l_zrefclever_hyperlink_bool
2017       \bool_set_true:N \l_zrefclever_hyperref_warn_bool
2018     } ,
2019   hyperref / false .code:n =
2020     {
2021       \bool_set_false:N \l_zrefclever_hyperlink_bool
2022       \bool_set_false:N \l_zrefclever_hyperref_warn_bool
2023     } ,
2024   hyperref .initial:n = auto ,
2025   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.

```
2026   nohyperref .meta:n = { hyperref = false } ,
2027   nohyperref .value_forbidden:n = true ,
2028 }
2029 \AddToHook { begindocument }
```

```

2030 {
2031   \__zrefclever_if_package_loaded:nTF { hyperref }
2032   {
2033     \bool_if:NT \l__zrefclever_hyperlink_bool
2034     { \RequirePackage { zref-hyperref } }
2035   }
2036   {
2037     \bool_if:NT \l__zrefclever_hyperref_warn_bool
2038     { \msg_warning:nn { zref-clever } { missing-hyperref } }
2039     \bool_set_false:N \l__zrefclever_hyperlink_bool
2040   }
2041   \keys_define:nn { zref-clever/reference }
2042   {
2043     hyperref .code:n =
2044     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2045     nohyperref .code:n =
2046     { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2047   }
2048 }

```

nameinlink option

```

2049 \str_new:N \l__zrefclever_nameinlink_str
2050 \keys_define:nn { zref-clever/reference }
2051 {
2052   nameinlink .choice: ,
2053   nameinlink / true .code:n =
2054   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2055   nameinlink / false .code:n =
2056   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2057   nameinlink / single .code:n =
2058   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2059   nameinlink / tsingle .code:n =
2060   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2061   nameinlink .initial:n = tsingle ,
2062   nameinlink .default:n = true ,
2063 }

```

preposinlink option (deprecated)

```

2064 \keys_define:nn { zref-clever/reference }
2065 {
2066   preposinlink .code:n =
2067   {
2068     % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2069     \msg_warning:nmmm { zref-clever } { option-deprecated }
2070     { preposinlink } { rebounds }
2071   } ,
2072 }

```

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`.

```

2073 \AddToHook { begindocument }
2074 {
2075   \__zrefclever_if_package_loaded:nTF { babel }
2076   {
2077     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
2078     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2079   }
2080   {
2081     \__zrefclever_if_package_loaded:nTF { polyglossia }
2082     {
2083       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2084       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2085     }
2086     {
2087       \tl_set:Nn \l__zrefclever_current_language_tl { english }
2088       \tl_set:Nn \l__zrefclever_main_language_tl { english }
2089     }
2090   }
2091 }

2092 \keys_define:nn { zref-clever/reference }
2093 {
2094   lang .code:n =
2095   {
2096     \AddToHook { begindocument }
2097     {
2098       \str_case:nnF {#1}
2099       {
2100         { current }
2101         {
2102           \tl_set:Nn \l__zrefclever_ref_language_tl
2103             { \l__zrefclever_current_language_tl }
2104         }
2105
2106         { main }
2107         {
2108           \tl_set:Nn \l__zrefclever_ref_language_tl
2109             { \l__zrefclever_main_language_tl }

```

```

2110     }
2111   }
2112   {
2113     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2114     \__zrefclever_language_if_declared:nF {#1}
2115     {
2116       \msg_warning:nnn { zref-clever }
2117       { unknown-language-opt } {#1}
2118     }
2119   }
2120   \__zrefclever_provide_langfile:x
2121   { \l__zrefclever_ref_language_tl }
2122 }
2123 },
2124 lang .initial:n = current ,
2125 lang .value_required:n = true ,
2126 }
2127 \AddToHook { begindocument / before }
2128 {
2129   \AddToHook { begindocument }
2130   {

```

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.

```

2131   \keys_define:nn { zref-clever/reference }
2132   {
2133     lang .code:n =
2134     {
2135       \str_case:nnF {#1}
2136       {
2137         { current }
2138         {
2139           \tl_set:Nn \l__zrefclever_ref_language_tl
2140           { \l__zrefclever_current_language_tl }
2141         }
2142
2143         { main }
2144         {
2145           \tl_set:Nn \l__zrefclever_ref_language_tl
2146           { \l__zrefclever_main_language_tl }
2147         }
2148       }
2149     }
2150     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2151     \__zrefclever_language_if_declared:nF {#1}
2152     {
2153       \msg_warning:nnn { zref-clever }
2154       { unknown-language-opt } {#1}
2155     }
2156   }
2157 },
2158 }
2159 }

```



```
2160 }
```

d option

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

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

```
2161 \tl_new:N \l__zrefclever_ref_decl_case_tl
2162 \keys_define:nn { zref-clever/reference }
2163 {
2164   d .code:n =
2165     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2166 }
2167 \AddToHook { begindocument }
2168 {
2169   \keys_define:nn { zref-clever/reference }
2170 }
```

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

```
2171   d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2172   d .value_required:n = true ,
2173 }
2174 }
```

nudge & co. options

```
2175 \bool_new:N \l__zrefclever_nudge_enabled_bool
2176 \bool_new:N \l__zrefclever_nudge_multitype_bool
2177 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2178 \bool_new:N \l__zrefclever_nudge_singular_bool
2179 \bool_new:N \l__zrefclever_nudge_gender_bool
2180 \tl_new:N \l__zrefclever_ref_gender_tl
2181 \keys_define:nn { zref-clever/reference }
2182 {
2183   nudge .choice: ,
2184   nudge / true .code:n =
2185     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2186   nudge / false .code:n =
2187     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2188   nudge / ifdraft .code:n =
2189     {
2190       \ifdraft
2191         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2192         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2193     } ,
2194   nudge / iffina .code:n =
2195     {
2196       \ifoptionfinal
2197         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
```

```

2198         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2199     } ,
2200     nudge .initial:n = false ,
2201     nudge .default:n = true ,
2202     nonudge .meta:n = { nudge = false } ,
2203     nonudge .value_forbidden:n = true ,
2204     nudgeif .code:n =
2205     {
2206         \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2207         \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2208         \bool_set_false:N \l__zrefclever_nudge_gender_bool
2209         \clist_map_inline:nn {#1}
2210         {
2211             \str_case:nnF {##1}
2212             {
2213                 { multitype }
2214                 { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2215                 { comptosing }
2216                 { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2217                 { gender }
2218                 { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2219                 { all }
2220                 {
2221                     \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2222                     \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2223                     \bool_set_true:N \l__zrefclever_nudge_gender_bool
2224                 }
2225             }
2226         {
2227             \msg_warning:nnn { zref-clever }
2228             { nudgeif-unknown-value } {##1}
2229         }
2230     }
2231 } ,
2232 nudgeif .value_required:n = true ,
2233 nudgeif .initial:n = all ,
2234 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2235 sg .initial:n = false ,
2236 sg .default:n = true ,
2237 g .code:n =
2238 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2239 }
2240 \AddToHook { begindocument }
2241 {
2242     \keys_define:nn { zref-clever/reference }
2243     {

```

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

```

2244         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2245         g .value_required:n = true ,
2246     }
2247 }

```

font option

```
2248 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2249 \keys_define:nn { zref-clever/reference }
2250 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```
2251 \keys_define:nn { zref-clever/reference }
2252 {
2253   titleref .code:n =
2254   {
2255     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2256     \msg_warning:nxxx { zref-clever }{ option-deprecated } { titleref }
2257     { \iow_char:N\usepackage\iow_char:N{zref-titleref\iow_char:N} }
2258   } ,
2259 }
```

vario option

```
2260 \keys_define:nn { zref-clever/reference }
2261 {
2262   vario .code:n =
2263   {
2264     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2265     \msg_warning:nxxx { zref-clever }{ option-deprecated } { vario }
2266     { \iow_char:N\usepackage\iow_char:N{zref-vario\iow_char:N} }
2267   } ,
2268 }
```

note option

```
2269 \tl_new:N \l__zrefclever_zceref_note_tl
2270 \keys_define:nn { zref-clever/reference }
2271 {
2272   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
2273   note .value_required:n = true ,
2274 }
```

check option

Integration with zref-check.

```
2275 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2276 \bool_new:N \l__zrefclever_zceref_with_check_bool
2277 \keys_define:nn { zref-clever/reference }
2278 {
2279   check .code:n =
2280   { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2281 }
2282 \AddToHook { begindocument }
2283 {
2284   \__zrefclever_if_package_loaded:nTF { zref-check }
2285   {
2286     \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2287     {
2288       \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2289       \keys_define:nn { zref-clever/reference }
2290       {
```

```

2291         check .code:n =
2292         {
2293             \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2294             \keys_set:nn { zref-check / zcheck } {#1}
2295         } ,
2296         check .value_required:n = true ,
2297     }
2298 }
2299 {
2300     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2301     \keys_define:nn { zref-clever/reference }
2302     {
2303         check .code:n =
2304         {
2305             \msg_warning:nnn { zref-clever }
2306             { zref-check-too-old } { 2021-09-16-v0.2.1 }
2307         } ,
2308     }
2309 }
2310 }
2311 {
2312     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2313     \keys_define:nn { zref-clever/reference }
2314     {
2315         check .code:n =
2316         { \msg_warning:nnn { zref-clever } { missing-zref-check } } ,
2317     }
2318 }
2319 }

```

reftype option

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

```

2320 \tl_new:N \l__zrefclever_reftype_override_tl
2321 \keys_define:nn { zref-clever/label }
2322 {
2323     reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2324     reftype .default:n = {} ,
2325     reftype .initial:n = {} ,
2326 }

```

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`.

```

2327 \prop_new:N \l__zrefclever_counter_type_prop
2328 \keys_define:nn { zref-clever/label }
2329 {
2330     countertype .code:n =
2331     {

```

```

2332     \keyval_parse:nnn
2333     {
2334         \msg_warning:nnnn { zref-clever }
2335         { key-requires-value } { countertype }
2336     }
2337     {
2338         \__zrefclever_prop_put_non_empty:Nnn
2339         \l__zrefclever_counter_type_prop
2340     }
2341     {#1}
2342 } ,
2343 countertype .value_required:n = true ,
2344 countertype .initial:n =
2345 {
2346     subsection    = section ,
2347     subsubsection = section ,
2348     subparagraph  = paragraph ,
2349     enumi         = item ,
2350     enumii        = item ,
2351     enumiii       = item ,
2352     enumiv        = item ,
2353     mpfootnote   = footnote ,
2354 } ,
2355 }

```

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. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resetters_seq` with the `counterresetby` option.

```

2356 \seq_new:N \l__zrefclever_counter_resetters_seq
2357 \keys_define:nn { zref-clever/label }
2358 {

```

```

2359 counterresetters .code:n =
2360 {
2361   \clist_map_inline:nn {#1}
2362   {
2363     \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2364     {
2365       \seq_put_right:Nn
2366         \l__zrefclever_counter_resetters_seq {##1}
2367     }
2368   }
2369 },
2370 counterresetters .initial:n =
2371 {
2372   part ,
2373   chapter ,
2374   section ,
2375   subsection ,
2376   subsubsection ,
2377   paragraph ,
2378   subparagraph ,
2379 },
2380 counterresetters .value_required:n = true ,
2381 }

```

counterresetby option

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

```

2382 \prop_new:N \l__zrefclever_counter_resetby_prop
2383 \keys_define:nn { zref-clever/label }
2384 {
2385   counterresetby .code:n =
2386   {
2387     \keyval_parse:nnn
2388     {
2389       \msg_warning:nnn { zref-clever }
2390         { key-requires-value } { counterresetby }
2391     }
2392     {
2393       \__zrefclever_prop_put_non_empty:Nnn
2394         \l__zrefclever_counter_resetby_prop
2395       }
2396     {#1}
2397   } ,
2398   counterresetby .value_required:n = true ,
2399   counterresetby .initial:n =
2400   {

```

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.

```
2401     enumii = enumi   ,
2402     enumiii = enumii ,
2403     enumiv = enumiii ,
2404   } ,
2405 }
```

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.

```
2406 \tl_new:N \l__zrefclever_current_counter_tl
2407 \keys_define:nn { zref-clever/label }
2408 {
2409   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2410   currentcounter .default:n = \@currentcounter ,
2411   currentcounter .initial:n = \@currentcounter ,
2412 }
```

nocompat option

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

```

2443 }
2444 \AtEndOfPackage
2445 {
2446   \AddToHook { begindocument }
2447   {
2448     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2449     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2450   }
2451 }

```

`_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 `\l_zrefclever_nocompat_modules_seq` is not in `\l_zrefclever_nocompat_modules_seq`. The `begindocument` hook is needed so that we can have the option functional along the whole preamble, not just at package load time. This 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>}
2452 \cs_new_protected:Npn \_zrefclever_compat_module:nn #1#2
2453 {
2454   \AddToHook { begindocument }
2455   {
2456     \bool_if:NF \g__zrefclever_nocompat_bool
2457     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2458     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2459   }
2460 }

```

(End definition for `_zrefclever_compat_module:nn`.)

Reference options

This is a set of options related to reference typesetting which receive equal treatment and, hence, are handled in batch. Since we are dealing with options to be passed to `\zcref` or to `\zcsetup` or at load time, only “not necessarily type-specific” options are pertinent here.

```

2461 \seq_map_inline:Nn
2462   \g__zrefclever_rf_opts_tl_reference_seq
2463   {
2464     \keys_define:nn { zref-clever/reference }
2465     {
2466       #1 .default:o = \c_novalue_tl ,
2467       #1 .code:n =
2468       {
2469         \tl_if_novalue:nTF {##1}
2470         {
2471           \_zrefclever_opt_tl_unset:c
2472           { \_zrefclever_opt_varname_general:nn {#1} { tl } }
2473         }

```



```

2474         {
2475             \__zrefclever_opt_tl_set:cn
2476             { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2477             {##1}
2478         }
2479     } ,
2480 }
2481 }
2482 \keys_define:nn { zref-clever/reference }
2483 {
2484     refpre .code:n =
2485     {
2486         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2487         \msg_warning:nmmm { zref-clever }{ option-deprecated }
2488         { refpre } { refbounds }
2489     } ,
2490     refpos .code:n =
2491     {
2492         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2493         \msg_warning:nmmm { zref-clever }{ option-deprecated }
2494         { refpos } { refbounds }
2495     } ,
2496     preref .code:n =
2497     {
2498         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2499         \msg_warning:nmmm { zref-clever }{ option-deprecated }
2500         { preref } { refbounds }
2501     } ,
2502     postref .code:n =
2503     {
2504         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2505         \msg_warning:nmmm { zref-clever }{ option-deprecated }
2506         { postref } { refbounds }
2507     } ,
2508 }
2509 \seq_map_inline:Nn
2510 \g__zrefclever_rf_opts_seq_refbounds_seq
2511 {
2512     \keys_define:nn { zref-clever/reference }
2513     {
2514         #1 .default:o = \c_novalue_tl ,
2515         #1 .code:n =
2516         {
2517             \tl_if_novalue:nTF {##1}
2518             {
2519                 \__zrefclever_opt_seq_unset:c
2520                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2521             }
2522             {
2523                 \seq_clear:N \l_tmpa_seq
2524                 \__zrefclever_opt_seq_set_clist_split:Nn
2525                 \l_tmpa_seq {##1}
2526                 \bool_lazy_or:nnTF
2527                 { \tl_if_empty_p:n {##1} }

```

```

2528         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2529         {
2530         \__zrefclever_opt_seq_set_eq:cN
2531         { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2532         \l_tmpa_seq
2533         }
2534         {
2535         \msg_warning:nxxx { zref-clever }
2536         { rebounds-must-be-four }
2537         {#1} { \seq_count:N \l_tmpa_seq }
2538         }
2539     } ,
2540 } ,
2541 }
2542 }
2543 \seq_map_inline:Nn
2544 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2545 {
2546   \keys_define:nn { zref-clever/reference }
2547   {
2548     #1 .choice: ,
2549     #1 / true .code:n =
2550     {
2551       \__zrefclever_opt_bool_set_true:c
2552       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2553     } ,
2554     #1 / false .code:n =
2555     {
2556       \__zrefclever_opt_bool_set_false:c
2557       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2558     } ,
2559     #1 / unset .code:n =
2560     {
2561       \__zrefclever_opt_bool_unset:c
2562       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2563     } ,
2564     #1 .default:n = true ,
2565     no #1 .meta:n = { #1 = false } ,
2566     no #1 .value_forbidden:n = true ,
2567   }
2568 }

```

Package options

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

```

2569 \keys_define:nn { }
2570 {
2571   zref-clever/zcsetup .inherit:n =
2572   {

```

```

2573         zref-clever/label ,
2574         zref-clever/reference ,
2575     }
2576 }

```

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.

```

2577 \bool_lazy_and:nnT
2578 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2579 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2580 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 \zcsetup

\zcsetup Provide \zcsetup.

```

\zcsetup{<options>}

2581 \NewDocumentCommand \zcsetup { m }
2582 { \__zrefclever_zcsetup:n {#1} }

```

(End definition for \zcsetup.)

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

```

\__zrefclever_zcsetup:n{<options>}

2583 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2584 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2585 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End definition for __zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

\zcRefTypeSetup is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any language-specific setting, either done at \zcLanguageSetup or by the package’s language files. On the other hand, they have a lower precedence than non type-specific general options. The <options> should be given in the usual key=val format. The <type> does not need to pre-exist, the property list variable to store the properties for the type gets created if need be.

```

\zcRefTypeSetup \zcRefTypeSetup {<type>} {<options>}

2586 \NewDocumentCommand \zcRefTypeSetup { m m }
2587 {
2588     \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2589     \keys_set:nn { zref-clever/typesetup } {#2}
2590     \tl_clear:N \l__zrefclever_setup_type_tl
2591 }

```

(End definition for \zcRefTypeSetup.)

```
2592 \seq_map_inline:Nn
2593   \g__zrefclever_rf_opts_tl_not_type_specific_seq
2594   {
2595     \keys_define:nn { zref-clever/typesetup }
2596     {
2597       #1 .code:n =
2598       {
2599         \msg_warning:nnn { zref-clever }
2600         { option-not-type-specific } {#1}
2601       } ,
2602     }
2603   }
2604 \seq_map_inline:Nn
2605   \g__zrefclever_rf_opts_tl_typesetup_seq
2606   {
2607     \keys_define:nn { zref-clever/typesetup }
2608     {
2609       #1 .default:o = \c_novalue_tl ,
2610       #1 .code:n =
2611       {
2612         \tl_if_novalue:nTF {##1}
2613         {
2614           \__zrefclever_opt_tl_unset:c
2615           {
2616             \__zrefclever_opt_varname_type:enn
2617             { \l__zrefclever_setup_type_tl } {#1} { tl }
2618           }
2619         }
2620         {
2621           \__zrefclever_opt_tl_set:cn
2622           {
2623             \__zrefclever_opt_varname_type:enn
2624             { \l__zrefclever_setup_type_tl } {#1} { tl }
2625           }
2626           {##1}
2627         }
2628       } ,
2629     }
2630   }
2631 \keys_define:nn { zref-clever/typesetup }
2632 {
2633   endrange .code:n =
2634   {
2635     \str_case:nnF {#1}
2636     {
2637       { ref }
2638       {
2639         \__zrefclever_opt_tl_clear:c
2640         {
2641           \__zrefclever_opt_varname_type:enn
2642           { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2643         }
2644         \__zrefclever_opt_tl_clear:c
```

```

2645         {
2646             \__zrefclever_opt_varname_type:enn
2647             { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2648         }
2649     }
2650
2651 { stripprefix }
2652 {
2653     \__zrefclever_opt_t1_set:cn
2654     {
2655         \__zrefclever_opt_varname_type:enn
2656         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2657     }
2658     { __zrefclever_get_endrange_stripprefix }
2659     \__zrefclever_opt_t1_clear:c
2660     {
2661         \__zrefclever_opt_varname_type:enn
2662         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2663     }
2664 }
2665
2666 { pagecomp }
2667 {
2668     \__zrefclever_opt_t1_set:cn
2669     {
2670         \__zrefclever_opt_varname_type:enn
2671         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2672     }
2673     { __zrefclever_get_endrange_pagecomp }
2674     \__zrefclever_opt_t1_clear:c
2675     {
2676         \__zrefclever_opt_varname_type:enn
2677         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2678     }
2679 }
2680
2681 { pagecomp2 }
2682 {
2683     \__zrefclever_opt_t1_set:cn
2684     {
2685         \__zrefclever_opt_varname_type:enn
2686         { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2687     }
2688     { __zrefclever_get_endrange_pagecomptwo }
2689     \__zrefclever_opt_t1_clear:c
2690     {
2691         \__zrefclever_opt_varname_type:enn
2692         { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2693     }
2694 }
2695
2696 { unset }
2697 {
2698     \__zrefclever_opt_t1_unset:c

```

```

2699         {
2700             \__zrefclever_opt_varname_type:enn
2701             { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2702         }
2703     \__zrefclever_opt_tl_unset:c
2704     {
2705         \__zrefclever_opt_varname_type:enn
2706         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2707     }
2708 }
2709 }
2710 {
2711     \tl_if_empty:nTF {#1}
2712     {
2713         \msg_warning:nnn { zref-clever }
2714         { endrange-property-undefined } {#1}
2715     }
2716     {
2717         \zref@ifpropundefined {#1}
2718         {
2719             \msg_warning:nnn { zref-clever }
2720             { endrange-property-undefined } {#1}
2721         }
2722         {
2723             \__zrefclever_opt_tl_set:cn
2724             {
2725                 \__zrefclever_opt_varname_type:enn
2726                 { \l__zrefclever_setup_type_tl }
2727                 { endrangefunc } { tl }
2728             }
2729             { __zrefclever_get_endrange_property }
2730             \__zrefclever_opt_tl_set:cn
2731             {
2732                 \__zrefclever_opt_varname_type:enn
2733                 { \l__zrefclever_setup_type_tl }
2734                 { endrangeprop } { tl }
2735             }
2736             {#1}
2737         }
2738     }
2739 }
2740 },
2741 endrange .value_required:n = true ,
2742 }
2743 \keys_define:nn { zref-clever/typesetup }
2744 {
2745     refpre .code:n =
2746     {
2747         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2748         \msg_warning:nmm { zref-clever }{ option-deprecated }
2749         { refpre } { refbounds }
2750     } ,
2751     refpos .code:n =
2752     {

```

```

2753     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2754     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2755     { refpos } { rebounds }
2756   } ,
2757   preref .code:n =
2758   {
2759     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2760     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2761     { preref } { rebounds }
2762   } ,
2763   postref .code:n =
2764   {
2765     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2766     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2767     { postref } { rebounds }
2768   } ,
2769 }
2770 \seq_map_inline:Nn
2771 \g__zrefclever_rf_opts_seq_rebounds_seq
2772 {
2773   \keys_define:nm { zref-clever/typesetup }
2774   {
2775     #1 .default:o = \c_novalue_tl ,
2776     #1 .code:n =
2777     {
2778       \tl_if_novalue:nTF {##1}
2779       {
2780         \__zrefclever_opt_seq_unset:c
2781         {
2782           \__zrefclever_opt_varname_type:enn
2783           { \l__zrefclever_setup_type_tl } {#1} { seq }
2784         }
2785       }
2786     }
2787     \seq_clear:N \l_tmpa_seq
2788     \__zrefclever_opt_seq_set_clist_split:Nn
2789     \l_tmpa_seq {##1}
2790     \bool_lazy_or:nnTF
2791     { \tl_if_empty_p:n {##1} }
2792     { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2793     {
2794       \__zrefclever_opt_seq_set_eq:cN
2795       {
2796         \__zrefclever_opt_varname_type:enn
2797         { \l__zrefclever_setup_type_tl } {#1} { seq }
2798       }
2799       \l_tmpa_seq
2800     }
2801     {
2802       \msg_warning:nxxx { zref-clever }
2803       { rebounds-must-be-four }
2804       {#1} { \seq_count:N \l_tmpa_seq }
2805     }
2806   }

```

```

2807     } ,
2808   }
2809 }
2810 \seq_map_inline:Nn
2811 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2812 {
2813   \keys_define:nn { zref-clever/typesetup }
2814   {
2815     #1 .choice: ,
2816     #1 / true .code:n =
2817     {
2818       \__zrefclever_opt_bool_set_true:c
2819       {
2820         \__zrefclever_opt_varname_type:enn
2821         { \l__zrefclever_setup_type_tl }
2822         {#1} { bool }
2823       }
2824     } ,
2825     #1 / false .code:n =
2826     {
2827       \__zrefclever_opt_bool_set_false:c
2828       {
2829         \__zrefclever_opt_varname_type:enn
2830         { \l__zrefclever_setup_type_tl }
2831         {#1} { bool }
2832       }
2833     } ,
2834     #1 / unset .code:n =
2835     {
2836       \__zrefclever_opt_bool_unset:c
2837       {
2838         \__zrefclever_opt_varname_type:enn
2839         { \l__zrefclever_setup_type_tl }
2840         {#1} { bool }
2841       }
2842     } ,
2843     #1 .default:n = true ,
2844     no #1 .meta:n = { #1 = false } ,
2845     no #1 .value_forbidden:n = true ,
2846   }
2847 }

```

5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `\zcLanguageSetup` 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>}
```



```

2848 \NewDocumentCommand \zcLanguageSetup { m m }
2849 {
2850   \group_begin:
2851   \__zrefclever_language_if_declared:nTF {#1}
2852   {
2853     \tl_clear:N \l__zrefclever_setup_type_tl
2854     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2855     \__zrefclever_opt_seq_get:cNF
2856     {
2857       \__zrefclever_opt_varname_language:nnn
2858       {#1} { declension } { seq }
2859     }
2860     \l__zrefclever_lang_declension_seq
2861     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2862     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2863     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2864     {
2865       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2866       \l__zrefclever_lang_decl_case_tl
2867     }
2868     \__zrefclever_opt_seq_get:cNF
2869     {
2870       \__zrefclever_opt_varname_language:nnn
2871       {#1} { gender } { seq }
2872     }
2873     \l__zrefclever_lang_gender_seq
2874     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2875     \keys_set:nn { zref-clever/langsetup } {#2}
2876   }
2877   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2878   \group_end:
2879 }
2880 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2881 \keys_define:nn { zref-clever/langsetup }
2882 {
2883   type .code:n =
2884   {
2885     \tl_if_empty:nTF {#1}
2886     { \tl_clear:N \l__zrefclever_setup_type_tl }
2887     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2888   } ,
2889
2890   case .code:n =
2891   {
2892     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2893     {
2894       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2895       { \l__zrefclever_setup_language_tl } {#1}
2896     }
2897   }

```

```

2898     \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2899     { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2900     {
2901       \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2902       {#1} { \l__zrefclever_setup_language_tl }
2903       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2904       \l__zrefclever_lang_decl_case_tl
2905     }
2906   }
2907 } ,
2908 case .value_required:n = true ,
2909
2910 gender .value_required:n = true ,
2911 gender .code:n =
2912 {
2913   \seq_if_empty:NNTF \l__zrefclever_lang_gender_seq
2914   {
2915     \msg_warning:nxxxx { zref-clever } { language-no-gender }
2916     { \l__zrefclever_setup_language_tl } { gender } {#1}
2917   }
2918   {
2919     \tl_if_empty:NNTF \l__zrefclever_setup_type_tl
2920     {
2921       \msg_warning:nnn { zref-clever }
2922       { option-only-type-specific } { gender }
2923     }
2924     {
2925       \seq_clear:N \l_tmpa_seq
2926       \clist_map_inline:nn {#1}
2927       {
2928         \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2929         { \seq_put_right:Nn \l_tmpa_seq {##1} }
2930         {
2931           \msg_warning:nxxx { zref-clever }
2932           { gender-not-declared }
2933           { \l__zrefclever_setup_language_tl } {##1}
2934         }
2935       }
2936       \__zrefclever_opt_seq_gset_eq:cN
2937       {
2938         \__zrefclever_opt_varname_lang_type:eenn
2939         { \l__zrefclever_setup_language_tl }
2940         { \l__zrefclever_setup_type_tl }
2941         { gender }
2942         { seq }
2943       }
2944       \l_tmpa_seq
2945     }
2946   }
2947 } ,
2948 }
2949 \seq_map_inline:Nn
2950 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2951 {

```

```

2952 \keys_define:nn { zref-clever/langsetup }
2953 {
2954   #1 .value_required:n = true ,
2955   #1 .code:n =
2956   {
2957     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2958     {
2959       \__zrefclever_opt_tl_gset:cn
2960       {
2961         \__zrefclever_opt_varname_lang_default:enn
2962         { \l__zrefclever_setup_language_tl } {#1} { t1 }
2963       }
2964       {##1}
2965     }
2966     {
2967       \msg_warning:nnn { zref-clever }
2968       { option-not-type-specific } {#1}
2969     }
2970   } ,
2971 }
2972 }
2973 \seq_map_inline:Nn
2974 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2975 {
2976   \keys_define:nn { zref-clever/langsetup }
2977   {
2978     #1 .value_required:n = true ,
2979     #1 .code:n =
2980     {
2981       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2982       {
2983         \__zrefclever_opt_tl_gset:cn
2984         {
2985           \__zrefclever_opt_varname_lang_default:enn
2986           { \l__zrefclever_setup_language_tl } {#1} { t1 }
2987         }
2988         {##1}
2989       }
2990       {
2991         \__zrefclever_opt_tl_gset:cn
2992         {
2993           \__zrefclever_opt_varname_lang_type:eenn
2994           { \l__zrefclever_setup_language_tl }
2995           { \l__zrefclever_setup_type_tl }
2996           {#1} { t1 }
2997         }
2998         {##1}
2999       }
3000     } ,
3001 }
3002 }
3003 \keys_define:nn { zref-clever/langsetup }
3004 {
3005   endrange .value_required:n = true ,

```

```

3006   endrange .code:n =
3007     {
3008       \str_case:nnF {#1}
3009       {
3010         { ref }
3011         {
3012           \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3013           {
3014             \__zrefclever_opt_tl_gclear:c
3015             {
3016               \__zrefclever_opt_varname_lang_default:enn
3017               { \l__zrefclever_setup_language_tl }
3018               { endrangefunc } { tl }
3019             }
3020             \__zrefclever_opt_tl_gclear:c
3021             {
3022               \__zrefclever_opt_varname_lang_default:enn
3023               { \l__zrefclever_setup_language_tl }
3024               { endrangeprop } { tl }
3025             }
3026           }
3027         {
3028           \__zrefclever_opt_tl_gclear:c
3029           {
3030             \__zrefclever_opt_varname_lang_type:eenn
3031             { \l__zrefclever_setup_language_tl }
3032             { \l__zrefclever_setup_type_tl }
3033             { endrangefunc } { tl }
3034           }
3035           \__zrefclever_opt_tl_gclear:c
3036           {
3037             \__zrefclever_opt_varname_lang_type:eenn
3038             { \l__zrefclever_setup_language_tl }
3039             { \l__zrefclever_setup_type_tl }
3040             { endrangeprop } { tl }
3041           }
3042         }
3043       }
3044     }
3045   { stripprefix }
3046   {
3047     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3048     {
3049       \__zrefclever_opt_tl_gset:cn
3050       {
3051         \__zrefclever_opt_varname_lang_default:enn
3052         { \l__zrefclever_setup_language_tl }
3053         { endrangefunc } { tl }
3054       }
3055       { __zrefclever_get_endrange_stripprefix }
3056       \__zrefclever_opt_tl_gclear:c
3057       {
3058         \__zrefclever_opt_varname_lang_default:enn
3059         { \l__zrefclever_setup_language_tl }

```

```

3060         { endrangeprop } { t1 }
3061     }
3062 }
3063 {
3064     \__zrefclever_opt_tl_gset:cn
3065     {
3066         \__zrefclever_opt_varname_lang_type:eenn
3067         { \l__zrefclever_setup_language_tl }
3068         { \l__zrefclever_setup_type_tl }
3069         { endrangefunc } { t1 }
3070     }
3071     { __zrefclever_get_endrange_stripprefix }
3072     \__zrefclever_opt_tl_gclear:c
3073     {
3074         \__zrefclever_opt_varname_lang_type:eenn
3075         { \l__zrefclever_setup_language_tl }
3076         { \l__zrefclever_setup_type_tl }
3077         { endrangeprop } { t1 }
3078     }
3079 }
3080 }
3081
3082 { pagecomp }
3083 {
3084     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3085     {
3086         \__zrefclever_opt_tl_gset:cn
3087         {
3088             \__zrefclever_opt_varname_lang_default:enn
3089             { \l__zrefclever_setup_language_tl }
3090             { endrangefunc } { t1 }
3091         }
3092         { __zrefclever_get_endrange_pagecomp }
3093         \__zrefclever_opt_tl_gclear:c
3094         {
3095             \__zrefclever_opt_varname_lang_default:enn
3096             { \l__zrefclever_setup_language_tl }
3097             { endrangeprop } { t1 }
3098         }
3099     }
3100 }
3101 {
3102     \__zrefclever_opt_tl_gset:cn
3103     {
3104         \__zrefclever_opt_varname_lang_type:eenn
3105         { \l__zrefclever_setup_language_tl }
3106         { \l__zrefclever_setup_type_tl }
3107         { endrangefunc } { t1 }
3108     }
3109     { __zrefclever_get_endrange_pagecomp }
3110     \__zrefclever_opt_tl_gclear:c
3111     {
3112         \__zrefclever_opt_varname_lang_type:eenn
3113         { \l__zrefclever_setup_language_tl }
3114         { \l__zrefclever_setup_type_tl }

```

```

3114         { endrangeprop } { t1 }
3115     }
3116 }
3117 }
3118
3119 { pagecomp2 }
3120 {
3121   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3122   {
3123     \__zrefclever_opt_tl_gset:cn
3124     {
3125       \__zrefclever_opt_varname_lang_default:enn
3126       { \l__zrefclever_setup_language_tl }
3127       { endrangefunc } { t1 }
3128     }
3129     { __zrefclever_get_endrange_pagecomptwo }
3130     \__zrefclever_opt_tl_gclear:c
3131     {
3132       \__zrefclever_opt_varname_lang_default:enn
3133       { \l__zrefclever_setup_language_tl }
3134       { endrangeprop } { t1 }
3135     }
3136   }
3137   {
3138     \__zrefclever_opt_tl_gset:cn
3139     {
3140       \__zrefclever_opt_varname_lang_type:eenn
3141       { \l__zrefclever_setup_language_tl }
3142       { \l__zrefclever_setup_type_tl }
3143       { endrangefunc } { t1 }
3144     }
3145     { __zrefclever_get_endrange_pagecomptwo }
3146     \__zrefclever_opt_tl_gclear:c
3147     {
3148       \__zrefclever_opt_varname_lang_type:eenn
3149       { \l__zrefclever_setup_language_tl }
3150       { \l__zrefclever_setup_type_tl }
3151       { endrangeprop } { t1 }
3152     }
3153   }
3154 }
3155 }
3156 {
3157   \tl_if_empty:nTF {#1}
3158   {
3159     \msg_warning:nnn { zref-clever }
3160     { endrange-property-undefined } {#1}
3161   }
3162   {
3163     \zref@ifpropundefined {#1}
3164     {
3165       \msg_warning:nnn { zref-clever }
3166       { endrange-property-undefined } {#1}
3167     }

```

```

3168         {
3169             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3170             {
3171                 \__zrefclever_opt_tl_gset:cn
3172                 {
3173                     \__zrefclever_opt_varname_lang_default:enn
3174                     { \l__zrefclever_setup_language_tl }
3175                     { endrangefunc } { tl }
3176                 }
3177                 { __zrefclever_get_endrange_property }
3178                 \__zrefclever_opt_tl_gset:cn
3179                 {
3180                     \__zrefclever_opt_varname_lang_default:enn
3181                     { \l__zrefclever_setup_language_tl }
3182                     { endrangeprop } { tl }
3183                 }
3184                 {#1}
3185             }
3186         {
3187             \__zrefclever_opt_tl_gset:cn
3188             {
3189                 \__zrefclever_opt_varname_lang_type:eenn
3190                 { \l__zrefclever_setup_language_tl }
3191                 { \l__zrefclever_setup_type_tl }
3192                 { endrangefunc } { tl }
3193             }
3194             { __zrefclever_get_endrange_property }
3195             \__zrefclever_opt_tl_gset:cn
3196             {
3197                 \__zrefclever_opt_varname_lang_type:eenn
3198                 { \l__zrefclever_setup_language_tl }
3199                 { \l__zrefclever_setup_type_tl }
3200                 { endrangeprop } { tl }
3201             }
3202             {#1}
3203         }
3204     }
3205 }
3206 } ,
3207 }
3208 }
3209 \keys_define:nn { zref-clever/langsetup }
3210 {
3211     refpre .code:n =
3212     {
3213         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3214         \msg_warning:nmmm { zref-clever }{ option-deprecated }
3215         { refpre } { refbounds }
3216     } ,
3217     refpos .code:n =
3218     {
3219         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3220         \msg_warning:nmmm { zref-clever }{ option-deprecated }
3221         { refpos } { refbounds }

```

```

3222     } ,
3223     preref .code:n =
3224     {
3225         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3226         \msg_warning:n { zref-clever } { option-deprecated }
3227         { preref } { rebounds }
3228     } ,
3229     postref .code:n =
3230     {
3231         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3232         \msg_warning:n { zref-clever } { option-deprecated }
3233         { postref } { rebounds }
3234     } ,
3235 }
3236 \seq_map_inline:Nn
3237   \g__zrefclever_rf_opts_tl_type_names_seq
3238   {
3239     \keys_define:nn { zref-clever/langsetup }
3240     {
3241       #1 .value_required:n = true ,
3242       #1 .code:n =
3243       {
3244         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3245         {
3246           \msg_warning:n { zref-clever }
3247           { option-only-type-specific } {#1}
3248         }
3249         {
3250           \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3251           {
3252             \__zrefclever_opt_tl_gset:cn
3253             {
3254               \__zrefclever_opt_varname_lang_type:een
3255               { \l__zrefclever_setup_language_tl }
3256               { \l__zrefclever_setup_type_tl }
3257               {#1} { tl }
3258             }
3259             {##1}
3260           }
3261           {
3262             \__zrefclever_opt_tl_gset:cn
3263             {
3264               \__zrefclever_opt_varname_lang_type:een
3265               { \l__zrefclever_setup_language_tl }
3266               { \l__zrefclever_setup_type_tl }
3267               { \l__zrefclever_lang_decl_case_tl - #1 }
3268               { tl }
3269             }
3270             {##1}
3271           }
3272         }
3273       } ,
3274     }
3275 }

```



```

3276 \seq_map_inline:Nn
3277   \g__zrefclever_rf_opts_seq_refbounds_seq
3278   {
3279     \keys_define:nm { zref-clever/langsetup }
3280     {
3281       #1 .value_required:n = true ,
3282       #1 .code:n =
3283       {
3284         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3285         {
3286           \seq_gclear:N \g_tmpa_seq
3287           \__zrefclever_opt_seq_gset_clist_split:Nn
3288             \g_tmpa_seq {##1}
3289           \bool_lazy_or:nnTF
3290             { \tl_if_empty_p:n {##1} }
3291             {
3292               \int_compare_p:nNn
3293                 { \seq_count:N \g_tmpa_seq } = { 4 }
3294             }
3295             {
3296               \__zrefclever_opt_seq_gset_eq:cN
3297                 {
3298                   \__zrefclever_opt_varname_lang_default:enn
3299                     { \l__zrefclever_setup_language_tl }
3300                     {##1} { seq }
3301                 }
3302               \g_tmpa_seq
3303             }
3304             {
3305               \msg_warning:nxxx { zref-clever }
3306                 { refbounds-must-be-four }
3307                 {##1} { \seq_count:N \g_tmpa_seq }
3308             }
3309           }
3310         {
3311           \seq_gclear:N \g_tmpa_seq
3312           \__zrefclever_opt_seq_gset_clist_split:Nn
3313             \g_tmpa_seq {##1}
3314           \bool_lazy_or:nnTF
3315             { \tl_if_empty_p:n {##1} }
3316             {
3317               \int_compare_p:nNn
3318                 { \seq_count:N \g_tmpa_seq } = { 4 }
3319             }
3320             {
3321               \__zrefclever_opt_seq_gset_eq:cN
3322                 {
3323                   \__zrefclever_opt_varname_lang_type:eenn
3324                     { \l__zrefclever_setup_language_tl }
3325                     { \l__zrefclever_setup_type_tl } {##1} { seq }
3326                 }
3327               \g_tmpa_seq
3328             }
3329           }

```

```

3330         \msg_warning:nxxx { zref-clever }
3331         { refbounds-must-be-four }
3332         {#1} { \seq_count:N \g_tmpa_seq }
3333     }
3334 }
3335 } ,
3336 }
3337 }
3338 \seq_map_inline:Nn
3339 \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
3340 {
3341     \keys_define:nm { zref-clever/langsetup }
3342     {
3343         #1 .choice: ,
3344         #1 / true .code:n =
3345         {
3346             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3347             {
3348                 \__zrefclever_opt_bool_gset_true:c
3349                 {
3350                     \__zrefclever_opt_varname_lang_default:enn
3351                     { \l__zrefclever_setup_language_tl }
3352                     {#1} { bool }
3353                 }
3354             }
3355             {
3356                 \__zrefclever_opt_bool_gset_true:c
3357                 {
3358                     \__zrefclever_opt_varname_lang_type:eenn
3359                     { \l__zrefclever_setup_language_tl }
3360                     { \l__zrefclever_setup_type_tl }
3361                     {#1} { bool }
3362                 }
3363             }
3364         } ,
3365         #1 / false .code:n =
3366         {
3367             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3368             {
3369                 \__zrefclever_opt_bool_gset_false:c
3370                 {
3371                     \__zrefclever_opt_varname_lang_default:enn
3372                     { \l__zrefclever_setup_language_tl }
3373                     {#1} { bool }
3374                 }
3375             }
3376             {
3377                 \__zrefclever_opt_bool_gset_false:c
3378                 {
3379                     \__zrefclever_opt_varname_lang_type:eenn
3380                     { \l__zrefclever_setup_language_tl }
3381                     { \l__zrefclever_setup_type_tl }
3382                     {#1} { bool }
3383                 }

```

```

3384         }
3385     } ,
3386     #1 .default:n = true ,
3387     no #1 .meta:n = { #1 = false } ,
3388     no #1 .value_forbidden:n = true ,
3389 }
3390 }

```

6 User interface

6.1 `\zcref`

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

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

```

3391 \NewDocumentCommand \zcref { s O { } m }
3392 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for `\zcref`.)

`__zrefclever_zcref:nnnn` An intermediate internal function, which does the actual heavy lifting, and places `{\labels}` as first argument, so that it can be protected by `\zref@wrapper@babel` in `\zcref`.

```
\__zrefclever_zcref:nnnn {\labels} {(*)} {\options}
```

```

3393 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3394 {
3395     \group_begin:

```

Set options.

```
3396     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

3397     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3398     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

Ensure language file for reference language is loaded, if available. We cannot rely on `\keys_set:nn` for the task, since if the `lang` option is set for `current`, the actual language may have changed outside our control. `__zrefclever_provide_langfile:x` does nothing if the language file is already loaded.

```
3399     \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3400     \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```

3401     \bool_lazy_and:nnT
3402     { \l__zrefclever_zrefcheck_available_bool }
3403     { \l__zrefclever_zcref_with_check_bool }
3404     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```
3405     \bool_lazy_or:nnT
3406     { \l__zrefclever_typeset_sort_bool }
3407     { \l__zrefclever_typeset_range_bool }
3408     { \__zrefclever_sort_labels: }
```

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

```
3409     \group_begin:
3410     \l__zrefclever_ref_typeset_font_tl
3411     \__zrefclever_typeset_refs:
3412     \group_end:
```

Typeset note.

```
3413     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3414     {
3415         \__zrefclever_get_rf_opt_tl:nxxN { notsep }
3416         { \l__zrefclever_label_type_a_tl }
3417         { \l__zrefclever_ref_language_tl }
3418         \l_tmpa_tl
3419         \l_tmpa_tl
3420         \l__zrefclever_zcref_note_tl
3421     }
```

Integration with zref-check.

```
3422     \bool_lazy_and:nnT
3423     { \l__zrefclever_zrefcheck_available_bool }
3424     { \l__zrefclever_zcref_with_check_bool }
3425     {
3426         \zrefcheck_zcref_end_label_maybe:
3427         \zrefcheck_zcref_run_checks_on_labels:n
3428         { \l__zrefclever_zcref_labels_seq }
3429     }
```

Integration with mathtools.

```
3430     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3431     {
3432         \__zrefclever_mathtools_showonlyrefs:n
3433         { \l__zrefclever_zcref_labels_seq }
3434     }
3435     \group_end:
3436 }
```

(End definition for __zrefclever_zcref:nnnn.)

```
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
```

```
3437 \seq_new:N \l__zrefclever_zcref_labels_seq
3438 \bool_new:N \l__zrefclever_link_star_bool
```

(End definition for \l__zrefclever_zcref_labels_seq and \l__zrefclever_link_star_bool.)

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```
\zcpageref{*}[\langle options \rangle]{\langle labels \rangle}

3439 \NewDocumentCommand \zcpageref { s O { } m }
3440 {
3441   \group_begin:
3442   \IfBooleanT {#1}
3443     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3444   \zcref [#2, ref = page] {#3}
3445   \group_end:
3446 }
```

(End definition for \zcpageref.)

7 Sorting

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

```
\l__zrefclever_label_type_a_tl Auxiliary variables, for use in sorting, and some also in typesetting. Used to store refer-
\l__zrefclever_label_type_b_tl ence information – label properties – of the “current” (a) and “next” (b) labels.
\l__zrefclever_label_enclval_a_tl 3447 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_enclval_b_tl 3448 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_extdoc_a_tl 3449 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_extdoc_b_tl 3450 \tl_new:N \l__zrefclever_label_enclval_b_tl
3451 \tl_new:N \l__zrefclever_label_extdoc_a_tl
3452 \tl_new:N \l__zrefclever_label_extdoc_b_tl
```

(End definition for \l__zrefclever_label_type_a_tl and others.)

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

```
3453 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for \l__zrefclever_sort_decided_bool.)

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

```
3454 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
3455 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for \l__zrefclever_sort_prior_a_int and \l__zrefclever_sort_prior_b_int.)

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

```
3456 \seq_new:N \l__zrefclever_label_types_seq
```

(End definition for `\l__zrefclever_label_types_seq`.)

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

```
3457 \cs_new_protected:Npn \__zrefclever_sort_labels:
3458 {
```

Store label types sequence.

```
3459   \seq_clear:N \l__zrefclever_label_types_seq
3460   \tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }
3461   {
3462     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3463     \__zrefclever_label_type_put_new_right:n
3464   }
```

Sort.

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

```

3493         { \zrefclever_sort_page:nn {##1} {##2} }
3494         { \zrefclever_sort_default:nn {##1} {##2} }
3495     }
3496 }
3497 }
3498 }

```

(End definition for \zrefclever_sort_labels:.)

\zrefclever_label_type_put_new_right:n

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

```

\zrefclever_label_type_put_new_right:n {<label>}
3499 \cs_new_protected:Npn \zrefclever_label_type_put_new_right:n #1
3500 {
3501   \zrefclever_extract_default:Nnnn
3502   \l_zrefclever_label_type_a_tl {#1} {zc@type} { }
3503   \seq_if_in:NVF \l_zrefclever_label_types_seq
3504   \l_zrefclever_label_type_a_tl
3505   {
3506     \seq_put_right:NV \l_zrefclever_label_types_seq
3507     \l_zrefclever_label_type_a_tl
3508   }
3509 }

```

(End definition for \zrefclever_label_type_put_new_right:n.)

\zrefclever_sort_default:mn

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

```

\zrefclever_sort_default:mn {<label a>} {<label b>}
3510 \cs_new_protected:Npn \zrefclever_sort_default:mn #1#2
3511 {
3512   \zrefclever_extract_default:Nnnn
3513   \l_zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3514   \zrefclever_extract_default:Nnnn
3515   \l_zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3516
3517   \tl_if_eq:NNTF
3518   \l_zrefclever_label_type_a_tl
3519   \l_zrefclever_label_type_b_tl
3520   { \zrefclever_sort_default_same_type:mn {#1} {#2} }
3521   { \zrefclever_sort_default_different_types:mn {#1} {#2} }
3522 }

```

(End definition for _zrefclever_sort_default:nn.)

```
\_zrefclever_sort_default_same_type:nn      \_zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
3523 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
3524 {
3525   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3526   {#1} {zc@enclval} { }
3527   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3528   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3529   {#2} {zc@enclval} { }
3530   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3531   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3532   {#1} {externaldocument} { }
3533   \_zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3534   {#2} {externaldocument} { }
3535
3536   \bool_set_false:N \l__zrefclever_sort_decided_bool
3537
3538   % First we check if there's any "external document" difference (coming
3539   % from 'zref-xr') and, if so, sort based on that.
3540   \tl_if_eq:NNF
3541     \l__zrefclever_label_extdoc_a_tl
3542     \l__zrefclever_label_extdoc_b_tl
3543     {
3544       \bool_if:nTF
3545         {
3546           \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3547           ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3548         }
3549         {
3550           \bool_set_true:N \l__zrefclever_sort_decided_bool
3551           \sort_return_same:
3552         }
3553         {
3554           \bool_if:nTF
3555             {
3556               ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3557               \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3558             }
3559             {
3560               \bool_set_true:N \l__zrefclever_sort_decided_bool
3561               \sort_return_swapped:
3562             }
3563             {
3564               \bool_set_true:N \l__zrefclever_sort_decided_bool
3565               % Two different "external documents": last resort, sort by the
3566               % document name itself.
3567               \str_compare:eNeTF
3568                 { \l__zrefclever_label_extdoc_b_tl } <
3569                 { \l__zrefclever_label_extdoc_a_tl }
3570                 { \sort_return_swapped: }
3571                 { \sort_return_same: }
3572             }
3573         }
}
```



```

3574 }
3575
3576 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3577 {
3578   \bool_if:nTF
3579   {
3580     % Both are empty: neither label has any (further) "enclosing
3581     % counters" (left).
3582     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3583     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3584   }
3585   {
3586     \bool_set_true:N \l__zrefclever_sort_decided_bool
3587     \int_compare:nNnTF
3588     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3589     >
3590     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3591     { \sort_return_swapped: }
3592     { \sort_return_same:   }
3593   }
3594 }
3595 \bool_if:nTF
3596 {
3597   % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
3598   \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3599 }
3600 {
3601   \bool_set_true:N \l__zrefclever_sort_decided_bool
3602   \int_compare:nNnTF
3603   { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3604   >
3605   { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3606   { \sort_return_swapped: }
3607   { \sort_return_same:   }
3608 }
3609 {
3610   \bool_if:nTF
3611   {
3612     % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3613     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3614   }
3615   {
3616     \bool_set_true:N \l__zrefclever_sort_decided_bool
3617     \int_compare:nNnTF
3618     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3619     <
3620     { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3621     { \sort_return_same:   }
3622     { \sort_return_swapped: }
3623   }
3624 }
3625 % Neither is empty: we can compare the values of the
3626 % current enclosing counter in the loop, if they are
3627 % equal, we are still in the loop, if they are not, a

```

```

3628 % sorting decision can be made directly.
3629 \int_compare:nNnTF
3630 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3631 =
3632 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3633 {
3634 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3635 { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3636 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3637 { \tl_tail:N \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 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3645 { \sort_return_swapped: }
3646 { \sort_return_same: }
3647 }
3648 }
3649 }
3650 }
3651 }
3652 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

`__zrefclever_sort_default_different_types:nn`

```

\__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
3653 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3654 {

```

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”.

```

3655 \int_zero:N \l__zrefclever_sort_prior_a_int
3656 \int_zero:N \l__zrefclever_sort_prior_b_int
3657 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3658 {
3659 \tl_if_eq:nnTF {##2} {{othertypes}}
3660 {
3661 \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3662 { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3663 \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3664 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3665 }
3666 {
3667 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3668 { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3669 {
3670 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3671 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3672 }
3673 }
3674 }

```

Then do the actual sorting.

```

3675     \bool_if:nTF
3676     {
3677         \int_compare_p:nNn
3678         { \l__zrefclever_sort_prior_a_int } <
3679         { \l__zrefclever_sort_prior_b_int }
3680     }
3681     { \sort_return_same: }
3682     {
3683         \bool_if:nTF
3684         {
3685             \int_compare_p:nNn
3686             { \l__zrefclever_sort_prior_a_int } >
3687             { \l__zrefclever_sort_prior_b_int }
3688         }
3689         { \sort_return_swapped: }
3690         {
3691             % Sort priorities are equal: the type that occurs first in
3692             % 'labels', as given by the user, is kept (or brought) forward.
3693             \seq_map_inline:Nn \l__zrefclever_label_types_seq
3694             {
3695                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3696                 { \seq_map_break:n { \sort_return_same: } }
3697                 {
3698                     \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3699                     { \seq_map_break:n { \sort_return_swapped: } }
3700                 }
3701             }
3702         }
3703     }
3704 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

`__zrefclever_sort_page:nn` The sorting function for sorting of defined labels for references to “page”. This function is expected to be called within the sorting loop of `__zrefclever_sort_labels:` and receives the pair of labels being considered for a change of order or not. It should *always* “return” either `\sort_return_same:` or `\sort_return_swapped:`. Compared to the sorting of default labels, this is a piece of cake (thanks to `abspage`).

```

\__zrefclever_sort_page:nn {(label a)} {(label b)}
3705 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3706 {
3707     \int_compare:nNnTF
3708     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3709     >
3710     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3711     { \sort_return_swapped: }
3712     { \sort_return_same: }
3713 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

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

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

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

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

Range compression, in particular, relies heavily on counting to be able 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 arbitrary long “streak” finishes, we have to store the label which (potentially) begins a range (kept in `\l__zrefclever_range_beg_label_tl`). `\l__zrefclever_next_maybe_range_bool` signals when “next” is potentially a range with “current”, and `\l__zrefclever_next_is_same_bool` when their values are actually equal.

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

Variables

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

```
\l_zrefclever_typeset_last_bool 3714 \seq_new:N \l__zrefclever_typeset_labels_seq
\l_zrefclever_last_of_type_bool 3715 \bool_new:N \l__zrefclever_typeset_last_bool
3716 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

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

```
\l_zrefclever_label_count_int 3717 \int_new:N \l__zrefclever_type_count_int
\l__zrefclever_ref_count_int 3718 \int_new:N \l__zrefclever_label_count_int
3719 \int_new:N \l__zrefclever_ref_count_int
```

(End definition for `\l_zrefclever_type_count_int`, `\l__zrefclever_label_count_int`, and `\l__zrefclever_ref_count_int`.)

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

```
\l__zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl 3720 \tl_new:N \l__zrefclever_label_a_tl
\l_zrefclever_typeset_queue_curr_tl 3721 \tl_new:N \l__zrefclever_label_b_tl
\l_zrefclever_type_first_label_tl 3722 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_type_first_label_type_tl 3723 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
3724 \tl_new:N \l__zrefclever_type_first_label_tl
3725 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

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

```
\l_zrefclever_name_in_link_bool 3726 \tl_new:N \l__zrefclever_type_name_tl
\l_zrefclever_type_name_missing_bool 3727 \bool_new:N \l__zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl 3728 \bool_new:N \l__zrefclever_type_name_missing_bool
\l__zrefclever_name_format_fallback_tl
\l_zrefclever_type_name_gender_seq
```

```

3729 \tl_new:N \l__zrefclever_name_format_tl
3730 \tl_new:N \l__zrefclever_name_format_fallback_tl
3731 \seq_new:N \l__zrefclever_type_name_gender_seq

```

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

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

```

\l__zrefclever_range_count_int
\l__zrefclever_range_same_count_int
\l__zrefclever_range_beg_label_tl
\l__zrefclever_range_beg_is_first_bool
\l__zrefclever_range_end_ref_tl
\l__zrefclever_next_maybe_range_bool
\l__zrefclever_next_is_same_bool
3732 \int_new:N \l__zrefclever_range_count_int
3733 \int_new:N \l__zrefclever_range_same_count_int
3734 \tl_new:N \l__zrefclever_range_beg_label_tl
3735 \bool_new:N \l__zrefclever_range_beg_is_first_bool
3736 \tl_new:N \l__zrefclever_range_end_ref_tl
3737 \bool_new:N \l__zrefclever_next_maybe_range_bool
3738 \bool_new:N \l__zrefclever_next_is_same_bool

```

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

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

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangesep_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_tl
\l__zrefclever_endrangefunc_tl
\l__zrefclever_endrangeprop_tl
\l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool
\l__zrefclever_rangetopair_bool
3739 \tl_new:N \l__zrefclever_tpairsep_tl
3740 \tl_new:N \l__zrefclever_tlistsep_tl
3741 \tl_new:N \l__zrefclever_tlastsep_tl
3742 \tl_new:N \l__zrefclever_namesep_tl
3743 \tl_new:N \l__zrefclever_pairsep_tl
3744 \tl_new:N \l__zrefclever_listsep_tl
3745 \tl_new:N \l__zrefclever_lastsep_tl
3746 \tl_new:N \l__zrefclever_rangesep_tl
3747 \tl_new:N \l__zrefclever_namefont_tl
3748 \tl_new:N \l__zrefclever_reffont_tl
3749 \tl_new:N \l__zrefclever_endrangefunc_tl
3750 \tl_new:N \l__zrefclever_endrangeprop_tl
3751 \bool_new:N \l__zrefclever_cap_bool
3752 \bool_new:N \l__zrefclever_abbrev_bool
3753 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

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

```

\l__zrefclever_refbounds_first_seq
\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
3754 \seq_new:N \l__zrefclever_refbounds_first_seq
3755 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3756 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3757 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3758 \seq_new:N \l__zrefclever_refbounds_mid_seq
3759 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3760 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3761 \seq_new:N \l__zrefclever_refbounds_last_seq
3762 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3763 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3764 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3765 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

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

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

```
3766 \bool_new:N \l__zrefclever_verbose_testing_bool
```

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```
3767 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3768 {
3769   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3770   \l__zrefclever_zcref_labels_seq
3771   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3772   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3773   \tl_clear:N \l__zrefclever_type_first_label_tl
3774   \tl_clear:N \l__zrefclever_type_first_label_type_tl
3775   \tl_clear:N \l__zrefclever_range_beg_label_tl
3776   \tl_clear:N \l__zrefclever_range_end_ref_tl
3777   \int_zero:N \l__zrefclever_label_count_int
3778   \int_zero:N \l__zrefclever_type_count_int
3779   \int_zero:N \l__zrefclever_ref_count_int
3780   \int_zero:N \l__zrefclever_range_count_int
3781   \int_zero:N \l__zrefclever_range_same_count_int
3782   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3783   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3784
3785   % Get type block options (not type-specific).
3786   \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3787   { \l__zrefclever_label_type_a_tl }
3788   { \l__zrefclever_ref_language_tl }
3789   \l__zrefclever_tpairsep_tl
3790   \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3791   { \l__zrefclever_label_type_a_tl }
3792   { \l__zrefclever_ref_language_tl }
3793   \l__zrefclever_tlistsep_tl
3794   \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3795   { \l__zrefclever_label_type_a_tl }
3796   { \l__zrefclever_ref_language_tl }
3797   \l__zrefclever_tlastsep_tl
3798
3799   % Process label stack.
3800   \bool_set_false:N \l__zrefclever_typeset_last_bool
3801   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3802   {
3803     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3804     \l__zrefclever_label_a_tl
3805     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3806     {
3807       \tl_clear:N \l__zrefclever_label_b_tl
3808       \bool_set_true:N \l__zrefclever_typeset_last_bool
3809     }
3810     {
```

```

3811         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3812         \l__zrefclever_label_b_tl
3813     }
3814
3815 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3816 {
3817     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3818     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3819 }
3820 {
3821     \__zrefclever_extract_default:NVnn
3822     \l__zrefclever_label_type_a_tl
3823     \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3824     \__zrefclever_extract_default:NVnn
3825     \l__zrefclever_label_type_b_tl
3826     \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3827 }
3828
3829 % First, we establish whether the "current label" (i.e. 'a') is the
3830 % last one of its type. This can happen because the "next label"
3831 % (i.e. 'b') is of a different type (or different definition status),
3832 % or because we are at the end of the list.
3833 \bool_if:NTF \l__zrefclever_typeset_last_bool
3834 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3835 {
3836     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3837     {
3838         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3839         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3840         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3841     }
3842     {
3843         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3844         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3845         {
3846             % Neither is undefined, we must check the types.
3847             \tl_if_eq:NNTF
3848             \l__zrefclever_label_type_a_tl
3849             \l__zrefclever_label_type_b_tl
3850             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3851             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3852         }
3853     }
3854 }
3855
3856 % Handle warnings in case of reference or type undefined.
3857 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3858 \zref@refused { \l__zrefclever_label_a_tl }
3859 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3860 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3861 {}
3862 {
3863     \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3864     {

```



```

3865         \msg_warning:nxx { zref-clever } { missing-type }
3866         { \l__zrefclever_label_a_tl }
3867     }
3868 \zref@ifrefcontainsprop
3869 { \l__zrefclever_label_a_tl }
3870 { \l__zrefclever_ref_property_tl }
3871 { }
3872 {
3873     \msg_warning:nxxx { zref-clever } { missing-property }
3874     { \l__zrefclever_ref_property_tl }
3875     { \l__zrefclever_label_a_tl }
3876 }
3877 }
3878
3879 % Get possibly type-specific separators, rebounds, font and other
3880 % options, once per type.
3881 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3882 {
3883     \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3884     { \l__zrefclever_label_type_a_tl }
3885     { \l__zrefclever_ref_language_tl }
3886     \l__zrefclever_namesep_tl
3887     \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3888     { \l__zrefclever_label_type_a_tl }
3889     { \l__zrefclever_ref_language_tl }
3890     \l__zrefclever_pairsep_tl
3891     \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3892     { \l__zrefclever_label_type_a_tl }
3893     { \l__zrefclever_ref_language_tl }
3894     \l__zrefclever_listsep_tl
3895     \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3896     { \l__zrefclever_label_type_a_tl }
3897     { \l__zrefclever_ref_language_tl }
3898     \l__zrefclever_lastsep_tl
3899     \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3900     { \l__zrefclever_label_type_a_tl }
3901     { \l__zrefclever_ref_language_tl }
3902     \l__zrefclever_rangesep_tl
3903     \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3904     { \l__zrefclever_label_type_a_tl }
3905     { \l__zrefclever_ref_language_tl }
3906     \l__zrefclever_namefont_tl
3907     \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3908     { \l__zrefclever_label_type_a_tl }
3909     { \l__zrefclever_ref_language_tl }
3910     \l__zrefclever_reffont_tl
3911     \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3912     { \l__zrefclever_label_type_a_tl }
3913     { \l__zrefclever_ref_language_tl }
3914     \l__zrefclever_endrangefunc_tl
3915     \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3916     { \l__zrefclever_label_type_a_tl }
3917     { \l__zrefclever_ref_language_tl }
3918     \l__zrefclever_endrangeprop_tl

```

```

3919 \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3920 { \l__zrefclever_label_type_a_tl }
3921 { \l__zrefclever_ref_language_tl }
3922 \l__zrefclever_cap_bool
3923 \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3924 { \l__zrefclever_label_type_a_tl }
3925 { \l__zrefclever_ref_language_tl }
3926 \l__zrefclever_abbrev_bool
3927 \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3928 { \l__zrefclever_label_type_a_tl }
3929 { \l__zrefclever_ref_language_tl }
3930 \l__zrefclever_rangetopair_bool
3931 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3932 { \l__zrefclever_label_type_a_tl }
3933 { \l__zrefclever_ref_language_tl }
3934 \l__zrefclever_refbounds_first_seq
3935 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }
3936 { \l__zrefclever_label_type_a_tl }
3937 { \l__zrefclever_ref_language_tl }
3938 \l__zrefclever_refbounds_first_sg_seq
3939 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3940 { \l__zrefclever_label_type_a_tl }
3941 { \l__zrefclever_ref_language_tl }
3942 \l__zrefclever_refbounds_first_pb_seq
3943 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3944 { \l__zrefclever_label_type_a_tl }
3945 { \l__zrefclever_ref_language_tl }
3946 \l__zrefclever_refbounds_first_rb_seq
3947 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3948 { \l__zrefclever_label_type_a_tl }
3949 { \l__zrefclever_ref_language_tl }
3950 \l__zrefclever_refbounds_mid_seq
3951 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3952 { \l__zrefclever_label_type_a_tl }
3953 { \l__zrefclever_ref_language_tl }
3954 \l__zrefclever_refbounds_mid_rb_seq
3955 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3956 { \l__zrefclever_label_type_a_tl }
3957 { \l__zrefclever_ref_language_tl }
3958 \l__zrefclever_refbounds_mid_re_seq
3959 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3960 { \l__zrefclever_label_type_a_tl }
3961 { \l__zrefclever_ref_language_tl }
3962 \l__zrefclever_refbounds_last_seq
3963 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3964 { \l__zrefclever_label_type_a_tl }
3965 { \l__zrefclever_ref_language_tl }
3966 \l__zrefclever_refbounds_last_pe_seq
3967 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3968 { \l__zrefclever_label_type_a_tl }
3969 { \l__zrefclever_ref_language_tl }
3970 \l__zrefclever_refbounds_last_re_seq
3971 }
3972

```

```

3973 % Here we send this to a couple of auxiliary functions.
3974 \bool_if:NTF \l__zrefclever_last_of_type_bool
3975 % There exists no next label of the same type as the current.
3976 { \__zrefclever_typeset_refs_last_of_type: }
3977 % There exists a next label of the same type as the current.
3978 { \__zrefclever_typeset_refs_not_last_of_type: }
3979 }
3980 }

```

(End definition for `__zrefclever_typeset_refs:`.)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is *not* the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the “next” label and find something of a different “type” (loose here, maybe different definition status, maybe end of stack). So, though this is not very strict, `__zrefclever_typeset_refs_last_of_type:` is more of a “wrapping up” function, and it is indeed the one which does the actual typesetting, while `__zrefclever_typeset_refs_not_last_of_type:` is more of an “accumulation” function.

`__zrefclever_typeset_refs_last_of_type:` Handles typesetting when the current label is the last of its type.

```

3981 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3982 {
3983 % Process the current label to the current queue.
3984 \int_case:nnF { \l__zrefclever_label_count_int }
3985 {
3986 % It is the last label of its type, but also the first one, and that's
3987 % what matters here: just store it.
3988 % Test: 'zc-typeset01.lvt': "Last of type: single"
3989 { 0 }
3990 {
3991 \tl_set:NV \l__zrefclever_type_first_label_tl
3992 \l__zrefclever_label_a_tl
3993 \tl_set:NV \l__zrefclever_type_first_label_type_tl
3994 \l__zrefclever_label_type_a_tl
3995 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3996 \l__zrefclever_refbounds_first_sg_seq
3997 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3998 }
3999
4000 % The last is the second: we have a pair (if not repeated).
4001 % Test: 'zc-typeset01.lvt': "Last of type: pair"
4002 { 1 }
4003 {
4004 \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4005 {
4006 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4007 \l__zrefclever_refbounds_first_sg_seq
4008 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4009 }
4010 {
4011 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4012 {
4013 \exp_not:V \l__zrefclever_pairsep_tl

```

```

4014         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4015         \l__zrefclever_refbounds_last_pe_seq
4016     }
4017     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4018     \l__zrefclever_refbounds_first_pb_seq
4019     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4020 }
4021 }
4022 }
4023 % Last is third or more of its type: without repetition, we'd have the
4024 % last element on a list, but control for possible repetition.
4025 {
4026   \int_case:nnF { \l__zrefclever_range_count_int }
4027   {
4028     % There was no range going on.
4029     % Test: 'zc-typeset01.lvt': "Last of type: not range"
4030     { 0 }
4031     {
4032       \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4033       {
4034         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4035         {
4036           \exp_not:V \l__zrefclever_pairsep_tl
4037           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4038           \l__zrefclever_refbounds_last_pe_seq
4039         }
4040       }
4041       {
4042         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4043         {
4044           \exp_not:V \l__zrefclever_lastsep_tl
4045           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4046           \l__zrefclever_refbounds_last_seq
4047         }
4048       }
4049     }
4050     % Last in the range is also the second in it.
4051     % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4052     { 1 }
4053     {
4054       \int_compare:nNnTF
4055       { \l__zrefclever_range_same_count_int } = { 1 }
4056       {
4057         % We know 'range_beg_is_first_bool' is false, since this is
4058         % the second element in the range, but the third or more in
4059         % the type list.
4060         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4061         {
4062           \exp_not:V \l__zrefclever_pairsep_tl
4063           \__zrefclever_get_ref:VN
4064           \l__zrefclever_range_beg_label_tl
4065           \l__zrefclever_refbounds_last_pe_seq
4066         }
4067         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq

```

```

4068         \l__zrefclever_refbounds_first_pb_seq
4069     \bool_set_true:N
4070         \l__zrefclever_type_first_refbounds_set_bool
4071     }
4072     {
4073     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4074     {
4075         \exp_not:V \l__zrefclever_listsep_tl
4076         \__zrefclever_get_ref:VN
4077         \l__zrefclever_range_beg_label_tl
4078         \l__zrefclever_refbounds_mid_seq
4079         \exp_not:V \l__zrefclever_lastsep_tl
4080         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4081         \l__zrefclever_refbounds_last_seq
4082     }
4083     }
4084 }
4085 }
4086 % Last in the range is third or more in it.
4087 {
4088     \int_case:nmF
4089     {
4090         \l__zrefclever_range_count_int -
4091         \l__zrefclever_range_same_count_int
4092     }
4093     {
4094     % Repetition, not a range.
4095     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4096     { 0 }
4097     {
4098     % If 'range_beg_is_first_bool' is true, it means it was also
4099     % the first of the type, and hence its typesetting was
4100     % already handled, and we just have to set refbounds.
4101     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4102     {
4103         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4104         \l__zrefclever_refbounds_first_sg_seq
4105         \bool_set_true:N
4106         \l__zrefclever_type_first_refbounds_set_bool
4107     }
4108     {
4109     \int_compare:nNnTF
4110     { \l__zrefclever_ref_count_int } < { 2 }
4111     {
4112         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4113         {
4114             \exp_not:V \l__zrefclever_pairsep_tl
4115             \__zrefclever_get_ref:VN
4116             \l__zrefclever_range_beg_label_tl
4117             \l__zrefclever_refbounds_last_pe_seq
4118         }
4119     }
4120     {
4121         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

4122         {
4123             \exp_not:V \l__zrefclever_lastsep_tl
4124             \__zrefclever_get_ref:VN
4125             \l__zrefclever_range_beg_label_tl
4126             \l__zrefclever_refbounds_last_seq
4127         }
4128     }
4129 }
4130 }
4131 % A 'range', but with no skipped value, treat as pair if range
4132 % started with first of type, otherwise as list.
4133 % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4134 { 1 }
4135 {
4136     % Ditto.
4137     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4138     {
4139         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4140         \l__zrefclever_refbounds_first_pb_seq
4141         \bool_set_true:N
4142         \l__zrefclever_type_first_refbounds_set_bool
4143         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4144         {
4145             \exp_not:V \l__zrefclever_pairsep_tl
4146             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4147             \l__zrefclever_refbounds_last_pe_seq
4148         }
4149     }
4150     {
4151         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4152         {
4153             \exp_not:V \l__zrefclever_listsep_tl
4154             \__zrefclever_get_ref:VN
4155             \l__zrefclever_range_beg_label_tl
4156             \l__zrefclever_refbounds_mid_seq
4157         }
4158         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4159         {
4160             \exp_not:V \l__zrefclever_lastsep_tl
4161             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4162             \l__zrefclever_refbounds_last_seq
4163         }
4164     }
4165 }
4166 }
4167 {
4168     % An actual range.
4169     % Test: 'zc-typeset01.lvt': "Last of type: range"
4170     % Ditto.
4171     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4172     {
4173         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4174         \l__zrefclever_refbounds_first_rb_seq
4175         \bool_set_true:N

```

```

4176         \l__zrefclever_type_first_refbounds_set_bool
4177     }
4178     {
4179     \int_compare:nNnTF
4180     { \l__zrefclever_ref_count_int } < { 2 }
4181     {
4182         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4183         {
4184             \exp_not:V \l__zrefclever_pairsep_tl
4185             \__zrefclever_get_ref:VN
4186             \l__zrefclever_range_beg_label_tl
4187             \l__zrefclever_refbounds_mid_rb_seq
4188         }
4189         \seq_set_eq:NN
4190         \l__zrefclever_type_first_refbounds_seq
4191         \l__zrefclever_refbounds_first_pb_seq
4192         \bool_set_true:N
4193         \l__zrefclever_type_first_refbounds_set_bool
4194     }
4195     {
4196         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4197         {
4198             \exp_not:V \l__zrefclever_lastsep_tl
4199             \__zrefclever_get_ref:VN
4200             \l__zrefclever_range_beg_label_tl
4201             \l__zrefclever_refbounds_mid_rb_seq
4202         }
4203     }
4204 }
4205 \bool_lazy_and:nnTF
4206 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4207 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4208 {
4209     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4210     \l__zrefclever_range_beg_label_tl
4211     \l__zrefclever_label_a_tl
4212     \l__zrefclever_range_end_ref_tl
4213     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4214     {
4215         \exp_not:V \l__zrefclever_rangeseq_tl
4216         \__zrefclever_get_ref_endrange:VVN
4217         \l__zrefclever_label_a_tl
4218         \l__zrefclever_range_end_ref_tl
4219         \l__zrefclever_refbounds_last_re_seq
4220     }
4221 }
4222 {
4223     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4224     {
4225         \exp_not:V \l__zrefclever_rangeseq_tl
4226         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4227         \l__zrefclever_refbounds_last_re_seq
4228     }
4229 }

```

```

4230     }
4231   }
4232 }
4233
4234 % Handle "range" option. The idea is simple: if the queue is not empty,
4235 % we replace it with the end of the range (or pair). We can still
4236 % retrieve the end of the range from 'label_a' since we know to be
4237 % processing the last label of its type at this point.
4238 \bool_if:NT \l__zrefclever_typeset_range_bool
4239 {
4240   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4241   {
4242     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4243     { }
4244     {
4245       \msg_warning:nxx { zref-clever } { single-element-range }
4246       { \l__zrefclever_type_first_label_type_tl }
4247     }
4248   }
4249   {
4250     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4251     \bool_if:NT \l__zrefclever_rangetopair_bool
4252     {
4253       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4254       { }
4255       {
4256         \__zrefclever_labels_in_sequence:nn
4257         { \l__zrefclever_type_first_label_tl }
4258         { \l__zrefclever_label_a_tl }
4259       }
4260     }
4261     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4262     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4263     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4264     {
4265       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4266       {
4267         \exp_not:V \l__zrefclever_pairsep_tl
4268         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4269         \l__zrefclever_refbounds_last_pe_seq
4270       }
4271       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4272       \l__zrefclever_refbounds_first_pb_seq
4273       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4274     }
4275     {
4276       \bool_lazy_and:nnTF
4277       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4278       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VWN } }
4279       {
4280         % We must get 'type_first_label_tl' instead of
4281         % 'range_beg_label_tl' here, since it is not necessary
4282         % that the first of type was actually starting a range for
4283         % the 'range' option to be used.

```



```

4284         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4285         \l__zrefclever_type_first_label_tl
4286         \l__zrefclever_label_a_tl
4287         \l__zrefclever_range_end_ref_tl
4288         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4289         {
4290             \exp_not:V \l__zrefclever_rangeseq_tl
4291             \__zrefclever_get_ref_endrange:VVN
4292             \l__zrefclever_label_a_tl
4293             \l__zrefclever_range_end_ref_tl
4294             \l__zrefclever_refbounds_last_re_seq
4295         }
4296     }
4297     {
4298         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4299         {
4300             \exp_not:V \l__zrefclever_rangeseq_tl
4301             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4302             \l__zrefclever_refbounds_last_re_seq
4303         }
4304     }
4305     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4306     \l__zrefclever_refbounds_first_rb_seq
4307     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4308 }
4309 }
4310 }
4311
4312 % If none of the special cases for the first of type refbounds have been
4313 % set, do it.
4314 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4315 {
4316     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4317     \l__zrefclever_refbounds_first_seq
4318 }
4319
4320 % Now that the type block is finished, we can add the name and the first
4321 % ref to the queue. Also, if "typeset" option is not "both", handle it
4322 % here as well.
4323 \__zrefclever_type_name_setup:
4324 \bool_if:nTF
4325 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4326 {
4327     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4328     { \__zrefclever_get_ref_first: }
4329 }
4330 {
4331     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4332     {
4333         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4334         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4335         {
4336             \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4337             \l__zrefclever_type_first_refbounds_seq

```

```

4338     }
4339   }
4340   {
4341     \bool_if:NTF \l__zrefclever_typeset_name_bool
4342     {
4343       % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4344       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4345       {
4346         \bool_if:NTF \l__zrefclever_name_in_link_bool
4347         {
4348           \exp_not:N \group_begin:
4349           \exp_not:V \l__zrefclever_namefont_tl
4350           \__zrefclever_hyperlink:nmn
4351           {
4352             \__zrefclever_extract_url_unexp:V
4353             \l__zrefclever_type_first_label_tl
4354           }
4355           {
4356             \__zrefclever_extract_unexp:Vnn
4357             \l__zrefclever_type_first_label_tl
4358             { anchor } { }
4359           }
4360           { \exp_not:V \l__zrefclever_type_name_tl }
4361           \exp_not:N \group_end:
4362         }
4363         {
4364           \exp_not:N \group_begin:
4365           \exp_not:V \l__zrefclever_namefont_tl
4366           \exp_not:V \l__zrefclever_type_name_tl
4367           \exp_not:N \group_end:
4368         }
4369       }
4370     }
4371     {
4372       % Logically, this case would correspond to "typeset=none", but
4373       % it should not occur, given that the options are set up to
4374       % typeset either "ref" or "name". Still, leave here a
4375       % sensible fallback, equal to the behavior of "both".
4376       % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4377       \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4378       { \__zrefclever_get_ref_first: }
4379     }
4380   }
4381 }
4382
4383 % Typeset the previous type block, if there is one.
4384 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4385 {
4386   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4387   { \l__zrefclever_tlistsep_tl }
4388   \l__zrefclever_typeset_queue_prev_tl
4389 }
4390
4391 % Extra log for testing.

```

```

4392 \bool_if:NT \l__zrefclever_verbose_testing_bool
4393   { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4394
4395 % Wrap up loop, or prepare for next iteration.
4396 \bool_if:NTF \l__zrefclever_typeset_last_bool
4397 {
4398   % We are finishing, typeset the current queue.
4399   \int_case:nnF { \l__zrefclever_type_count_int }
4400   {
4401     % Single type.
4402     % Test: 'zc-typeset01.lvt': "Last of type: single type"
4403     { 0 }
4404     { \l__zrefclever_typeset_queue_curr_tl }
4405     % Pair of types.
4406     % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4407     { 1 }
4408     {
4409       \l__zrefclever_tpairsep_tl
4410       \l__zrefclever_typeset_queue_curr_tl
4411     }
4412   }
4413   {
4414     % Last in list of types.
4415     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4416     \l__zrefclever_tlastsep_tl
4417     \l__zrefclever_typeset_queue_curr_tl
4418   }
4419   % And nudge in case of multitype reference.
4420   \bool_lazy_all:nT
4421   {
4422     { \l__zrefclever_nudge_enabled_bool }
4423     { \l__zrefclever_nudge_multitype_bool }
4424     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4425   }
4426   { \msg_warning:nn { zref-clever } { nudge-multitype } }
4427 }
4428 {
4429 % There are further labels, set variables for next iteration.
4430 \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4431   \l__zrefclever_typeset_queue_curr_tl
4432 \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4433 \tl_clear:N \l__zrefclever_type_first_label_tl
4434 \tl_clear:N \l__zrefclever_type_first_label_type_tl
4435 \tl_clear:N \l__zrefclever_range_beg_label_tl
4436 \tl_clear:N \l__zrefclever_range_end_ref_tl
4437 \int_zero:N \l__zrefclever_label_count_int
4438 \int_zero:N \l__zrefclever_ref_count_int
4439 \int_incr:N \l__zrefclever_type_count_int
4440 \int_zero:N \l__zrefclever_range_count_int
4441 \int_zero:N \l__zrefclever_range_same_count_int
4442 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4443 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4444 }
4445 }

```

(End definition for `_zrefclever_typeset_refs_last_of_type:`)

`_zrefclever_typeset_refs_not_last_of_type:`

Handles typesetting when the current label is not the last of its type.

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

```

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

```

```

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

```

```

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

```

```

4659         \l__zrefclever_range_beg_label_tl
4660         \l__zrefclever_label_a_tl
4661         \l__zrefclever_range_end_ref_tl
4662         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4663         {
4664             \exp_not:V \l__zrefclever_rangesep_tl
4665             \__zrefclever_get_ref_endrange:VVN
4666             \l__zrefclever_label_a_tl
4667             \l__zrefclever_range_end_ref_tl
4668             \l__zrefclever_refbounds_mid_re_seq
4669         }
4670     }
4671     {
4672         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4673         {
4674             \exp_not:V \l__zrefclever_rangesep_tl
4675             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4676             \l__zrefclever_refbounds_mid_re_seq
4677         }
4678     }
4679 }
4680 }
4681 % We just closed a range, reset 'range_beg_is_first' in case a
4682 % second range for the same type occurs, in which case its
4683 % 'range_beg' will no longer be 'first'.
4684 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4685 % Reset counters.
4686 \int_zero:N \l__zrefclever_range_count_int
4687 \int_zero:N \l__zrefclever_range_same_count_int
4688 }
4689 }
4690 % Step label counter for next iteration.
4691 \int_incr:N \l__zrefclever_label_count_int
4692 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`.)

Auxiliary functions

`__zrefclever_get_ref:nN` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:nN` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:nN` and `__zrefclever_get_ref_first:` get called, as they must, in the context of x type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after

that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

`_zrefclever_ref_default:` Default values for undefined references and undefined type names, respectively. We are ultimately using `\zref@default`, but calls to it should be made through these internal functions, according to the case. As a bonus, we don’t need to protect them with `\exp_not:N`, as `\zref@default` would require, since we already define them protected.

```
4693 \cs_new_protected:Npn \_zrefclever_ref_default:
4694   { \zref@default }
4695 \cs_new_protected:Npn \_zrefclever_name_default:
4696   { \zref@default }
```

(End definition for `_zrefclever_ref_default:` and `_zrefclever_name_default:.`)

`_zrefclever_get_ref:nN` Handles a complete reference block to be accumulated in the “queue”, including 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>}

4697 \cs_new:Npn \_zrefclever_get_ref:nN #1#2
4698   {
4699     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4700     {
4701       \bool_if:nTF
4702         {
4703           \l__zrefclever_hyperlink_bool &&
4704           ! \l__zrefclever_link_star_bool
4705         }
4706         {
4707           \seq_item:Nn #2 { 1 }
4708           \_zrefclever_hyperlink:nnn
4709             { \_zrefclever_extract_url_unexp:n {#1} }
4710             { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4711             {
4712               \seq_item:Nn #2 { 2 }
4713               \exp_not:N \group_begin:
4714               \exp_not:V \l__zrefclever_reffont_tl
4715               \_zrefclever_extract_unexp:nvn {#1}
4716                 { \l__zrefclever_ref_property_tl } { }
4717               \exp_not:N \group_end:
4718               \seq_item:Nn #2 { 3 }
4719             }
4720           \seq_item:Nn #2 { 4 }
4721         }
4722         {
4723           \seq_item:Nn #2 { 1 }
4724           \seq_item:Nn #2 { 2 }
4725           \exp_not:N \group_begin:

```

```

4726         \exp_not:V \l__zrefclever_reffont_tl
4727         \__zrefclever_extract_unexp:nvn {#1}
4728         { l__zrefclever_ref_property_tl } { }
4729         \exp_not:N \group_end:
4730         \seq_item:Nn #2 { 3 }
4731         \seq_item:Nn #2 { 4 }
4732     }
4733 }
4734 { \__zrefclever_ref_default: }
4735 }
4736 \cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }

```

(End definition for __zrefclever_get_ref:nN.)

```

\__zrefclever_get_ref_endrange:nnN
4737 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3
4738 {
4739   \str_if_eq:nnTF {#2} { zc@missingproperty }
4740   { \__zrefclever_ref_default: }
4741   {
4742     \bool_if:nTF
4743     {
4744       \l__zrefclever_hyperlink_bool &&
4745       ! \l__zrefclever_link_star_bool
4746     }
4747     {
4748       \seq_item:Nn #3 { 1 }
4749       \__zrefclever_hyperlink:nnn
4750       { \__zrefclever_extract_url_unexp:n {#1} }
4751       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4752       {
4753         \seq_item:Nn #3 { 2 }
4754         \exp_not:N \group_begin:
4755         \exp_not:V \l__zrefclever_reffont_tl
4756         \exp_not:n {#2}
4757         \exp_not:N \group_end:
4758         \seq_item:Nn #3 { 3 }
4759       }
4760       \seq_item:Nn #3 { 4 }
4761     }
4762     {
4763       \seq_item:Nn #3 { 1 }
4764       \seq_item:Nn #3 { 2 }
4765       \exp_not:N \group_begin:
4766       \exp_not:V \l__zrefclever_reffont_tl
4767       \exp_not:n {#2}
4768       \exp_not:N \group_end:
4769       \seq_item:Nn #3 { 3 }
4770       \seq_item:Nn #3 { 4 }
4771     }
4772   }
4773 }
4774 \cs_generate_variant:Nn \__zrefclever_get_ref_endrange:nnN { VVN }

```

(End definition for __zrefclever_get_ref_endrange:nnN.)

`_zrefclever_get_ref_first`: Handles a complete reference block for the first label of its type to be accumulated in the “queue”, including “pre” and “pos” elements, hyperlinking, and the reference type “name”. It does not receive arguments, but relies on being called in the appropriate place in `_zrefclever_typeset_refs_last_of_type`: where a number of variables are expected to be appropriately set for it to consume. Prominently among those is `\l_zrefclever_type_first_label_tl`, but it also expected to be called right after `_zrefclever_type_name_setup`: which sets `\l_zrefclever_type_name_tl` and `\l_zrefclever_name_in_link_bool` which it uses.

```

4775 \cs_new:Npn \_zrefclever_get_ref_first:
4776 {
4777   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
4778   { \_zrefclever_ref_default: }
4779   {
4780     \bool_if:NTF \l\_zrefclever_name_in_link_bool
4781     {
4782       \zref@ifrefcontainsprop
4783       { \l\_zrefclever_type_first_label_tl }
4784       { \l\_zrefclever_ref_property_tl }
4785       {
4786         \_zrefclever_hyperlink:nnn
4787         {
4788           \_zrefclever_extract_url_unexp:V
4789           \l\_zrefclever_type_first_label_tl
4790         }
4791         {
4792           \_zrefclever_extract_unexp:Vnn
4793           \l\_zrefclever_type_first_label_tl { anchor } { }
4794         }
4795         {
4796           \exp_not:N \group_begin:
4797           \exp_not:V \l\_zrefclever_namefont_tl
4798           \exp_not:V \l\_zrefclever_type_name_tl
4799           \exp_not:N \group_end:
4800           \exp_not:V \l\_zrefclever_namesep_tl
4801           \seq_item:Nn \l\_zrefclever_type_first_refbounds_seq { 1 }
4802           \seq_item:Nn \l\_zrefclever_type_first_refbounds_seq { 2 }
4803           \exp_not:N \group_begin:
4804           \exp_not:V \l\_zrefclever_reffont_tl
4805           \_zrefclever_extract_unexp:Vvn
4806           \l\_zrefclever_type_first_label_tl
4807           { \l\_zrefclever_ref_property_tl } { }
4808           \exp_not:N \group_end:
4809           \seq_item:Nn \l\_zrefclever_type_first_refbounds_seq { 3 }
4810         }
4811         \seq_item:Nn \l\_zrefclever_type_first_refbounds_seq { 4 }
4812       }
4813     }
4814     {
4815       \exp_not:N \group_begin:
4816       \exp_not:V \l\_zrefclever_namefont_tl
4817       \exp_not:V \l\_zrefclever_type_name_tl
4818       \exp_not:N \group_end:
4819       \exp_not:V \l\_zrefclever_namesep_tl
4820       \_zrefclever_ref_default:

```

```

4820     }
4821   }
4822   {
4823     \bool_if:nTF \l__zrefclever_type_name_missing_bool
4824     {
4825       \__zrefclever_name_default:
4826       \exp_not:V \l__zrefclever_namesep_tl
4827     }
4828     {
4829       \exp_not:N \group_begin:
4830       \exp_not:V \l__zrefclever_namefont_tl
4831       \exp_not:V \l__zrefclever_type_name_tl
4832       \exp_not:N \group_end:
4833       \tl_if_empty:NF \l__zrefclever_type_name_tl
4834         { \exp_not:V \l__zrefclever_namesep_tl }
4835     }
4836   \zref@ifrefcontainsprop
4837   { \l__zrefclever_type_first_label_tl }
4838   { \l__zrefclever_ref_property_tl }
4839   {
4840     \bool_if:nTF
4841     {
4842       \l__zrefclever_hyperlink_bool &&
4843       ! \l__zrefclever_link_star_bool
4844     }
4845     {
4846       \seq_item:Nn
4847         \l__zrefclever_type_first_refbounds_seq { 1 }
4848       \__zrefclever_hyperlink:nnn
4849       {
4850         \__zrefclever_extract_url_unexp:V
4851         \l__zrefclever_type_first_label_tl
4852       }
4853       {
4854         \__zrefclever_extract_unexp:Vnn
4855         \l__zrefclever_type_first_label_tl { anchor } { }
4856       }
4857     }
4858     {
4859       \seq_item:Nn
4860         \l__zrefclever_type_first_refbounds_seq { 2 }
4861       \exp_not:N \group_begin:
4862       \exp_not:V \l__zrefclever_reffont_tl
4863       \__zrefclever_extract_unexp:Vnn
4864         \l__zrefclever_type_first_label_tl
4865         { \l__zrefclever_ref_property_tl } { }
4866       \exp_not:N \group_end:
4867       \seq_item:Nn
4868         \l__zrefclever_type_first_refbounds_seq { 3 }
4869     }
4870     \seq_item:Nn
4871       \l__zrefclever_type_first_refbounds_seq { 4 }
4872   }
4873   {
4874     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }

```

```

4874         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4875         \exp_not:N \group_begin:
4876         \exp_not:V \l__zrefclever_reffont_tl
4877         \__zrefclever_extract_unexp:Vvn
4878         \l__zrefclever_type_first_label_tl
4879         { \l__zrefclever_ref_property_tl } { }
4880         \exp_not:N \group_end:
4881         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4882         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4883     }
4884 }
4885 { \__zrefclever_ref_default: }
4886 }
4887 }
4888 }

```

(End definition for `__zrefclever_get_ref_first:`.)

`__zrefclever_type_name_setup:` Auxiliary function to `__zrefclever_typeset_refs_last_of_type:`. It is responsible for setting the type name variable `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool`. If a type name can't be found, `\l__zrefclever_type_name_tl` is cleared. The function takes no arguments, but is expected to be called in `__zrefclever_typeset_refs_last_of_type:` right before `__zrefclever_get_ref_first:`, which is the main consumer of the variables it sets, though not the only one (and hence this cannot be moved into `__zrefclever_get_ref_first:` itself). It also expects a number of relevant variables to have been appropriately set, and which it uses, prominently `\l__zrefclever_type_first_label_type_tl`, but also the queue itself in `\l__zrefclever_typeset_queue_curr_tl`, which should be "ready except for the first label", and the type counter `\l__zrefclever_type_count_int`.

```

4889 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4890 {
4891     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4892     {
4893         \tl_clear:N \l__zrefclever_type_name_tl
4894         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4895     }
4896     {
4897         \tl_if_eq:NnTF
4898         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4899         {
4900             \tl_clear:N \l__zrefclever_type_name_tl
4901             \bool_set_true:N \l__zrefclever_type_name_missing_bool
4902         }
4903         {
4904             % Determine whether we should use capitalization, abbreviation,
4905             % and plural.
4906             \bool_lazy_or:nnTF
4907             { \l__zrefclever_cap_bool }
4908             {
4909                 \l__zrefclever_capfirst_bool &&
4910                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4911             }
4912             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }

```

```

4913     { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4914 % If the queue is empty, we have a singular, otherwise, plural.
4915 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4916   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4917   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4918 \bool_lazy_and:nnTF
4919   { \l__zrefclever_abbrev_bool }
4920   {
4921     ! \int_compare_p:nNn
4922       { \l__zrefclever_type_count_int } = { 0 } ||
4923     ! \l__zrefclever_noabbrev_first_bool
4924   }
4925   {
4926     \tl_set:NV \l__zrefclever_name_format_fallback_tl
4927       \l__zrefclever_name_format_tl
4928     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4929   }
4930   { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4931
4932 % Handle number and gender nudges.
4933 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4934   {
4935     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4936     {
4937       \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4938       {
4939         \msg_warning:nnx { zref-clever }
4940           { nudge-plural-when-sg }
4941           { \l__zrefclever_type_first_label_type_tl }
4942       }
4943     }
4944     {
4945       \bool_lazy_all:nT
4946       {
4947         { \l__zrefclever_nudge_comptosing_bool }
4948         { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4949       }
4950       {
4951         \int_compare_p:nNn
4952           { \l__zrefclever_label_count_int } > { 0 }
4953       }
4954     }
4955     {
4956       \msg_warning:nnx { zref-clever }
4957         { nudge-comptosing }
4958         { \l__zrefclever_type_first_label_type_tl }
4959     }
4960 \bool_lazy_and:nnT
4961   { \l__zrefclever_nudge_gender_bool }
4962   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4963   {
4964     \__zrefclever_get_rf_opt_seq:nxxN { gender }
4965     { \l__zrefclever_type_first_label_type_tl }
4966     { \l__zrefclever_ref_language_tl }

```

```

4967         \l__zrefclever_type_name_gender_seq
4968     \seq_if_in:NVF
4969         \l__zrefclever_type_name_gender_seq
4970         \l__zrefclever_ref_gender_tl
4971     {
4972         \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4973     {
4974         \msg_warning:nxxxx { zref-clever }
4975         { nudge-gender-not-declared-for-type }
4976         { \l__zrefclever_ref_gender_tl }
4977         { \l__zrefclever_type_first_label_type_tl }
4978         { \l__zrefclever_ref_language_tl }
4979     }
4980     {
4981         \msg_warning:nxxxxx { zref-clever }
4982         { nudge-gender-mismatch }
4983         { \l__zrefclever_type_first_label_type_tl }
4984         { \l__zrefclever_ref_gender_tl }
4985         {
4986             \seq_use:Nn
4987             \l__zrefclever_type_name_gender_seq { ,~ }
4988         }
4989         { \l__zrefclever_ref_language_tl }
4990     }
4991 }
4992 }
4993 }
4994
4995 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4996 {
4997     \__zrefclever_opt_tl_get:cNF
4998     {
4999         \__zrefclever_opt_varname_type:een
5000         { \l__zrefclever_type_first_label_type_tl }
5001         { \l__zrefclever_name_format_tl }
5002         { tl }
5003     }
5004     \l__zrefclever_type_name_tl
5005     {
5006         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5007         {
5008             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5009             \tl_put_left:NV \l__zrefclever_name_format_tl
5010             \l__zrefclever_ref_decl_case_tl
5011         }
5012         \__zrefclever_opt_tl_get:cNF
5013         {
5014             \__zrefclever_opt_varname_lang_type:eeen
5015             { \l__zrefclever_ref_language_tl }
5016             { \l__zrefclever_type_first_label_type_tl }
5017             { \l__zrefclever_name_format_tl }
5018             { tl }
5019         }
5020     }
5021     \l__zrefclever_type_name_tl

```

```

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

```



```

5075         { \l__zrefclever_type_first_label_type_tl }
5076         { \l__zrefclever_name_format_fallback_tl }
5077         { tl }
5078     }
5079     \l__zrefclever_type_name_tl
5080     {
5081         \tl_clear:N \l__zrefclever_type_name_tl
5082         \bool_set_true:N
5083             \l__zrefclever_type_name_missing_bool
5084         \msg_warning:nxxx { zref-clever }
5085             { missing-name }
5086         { \l__zrefclever_name_format_tl }
5087         { \l__zrefclever_type_first_label_type_tl }
5088     }
5089 }
5090 }
5091 }
5092 }
5093 }
5094 }
5095
5096 % Signal whether the type name is to be included in the hyperlink or not.
5097 \bool_lazy_any:nTF
5098 {
5099     { ! \l__zrefclever_hyperlink_bool }
5100     { \l__zrefclever_link_star_bool }
5101     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5102     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5103 }
5104 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5105 {
5106     \bool_lazy_any:nTF
5107     {
5108         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5109         {
5110             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5111             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5112         }
5113         {
5114             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5115             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5116             \l__zrefclever_typeset_last_bool &&
5117             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5118         }
5119     }
5120     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5121     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5122 }
5123 }

```

(End definition for `__zrefclever_type_name_setup:`.)

`__zrefclever_hyperlink:nmn` This avoids using the internal `\hyper@link`, using only public `hyperref` commands (see <https://github.com/latex3/hyperref/issues/229#issuecomment-1093870142>, thanks Ulrike Fisher).

```

    \__zrefclever_hyperlink:nnn {<url/file>} {<anchor>} {<text>}
5124 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5125 {
5126   \tl_if_empty:nTF {#1}
5127     { \hyperlink {#2} {#3} }
5128     { \hyper@linkfile {#3} {#1} {#2} }
5129 }

```

(End definition for __zrefclever_hyperlink:nnn.)

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

```

5130 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5131 {
5132   \zref@ifpropundefined { urluse }
5133     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5134     {
5135       \zref@ifrefcontainsprop {#1} { urluse }
5136         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5137         { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5138     }
5139 }
5140 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for __zrefclever_extract_url_unexp:n.)

__zrefclever_labels_in_sequence:nn Auxiliary function to __zrefclever_typeset_refs_not_last_of_type:. Sets \l__zrefclever_next_maybe_range_bool to true if <label b> comes in immediate sequence from <label a>. And sets both \l__zrefclever_next_maybe_range_bool and \l__zrefclever_next_is_same_bool to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside __zrefclever_typeset_refs_not_last_of_type:, so this function is expected to be called at its beginning, if compression is enabled.

```

    \__zrefclever_labels_in_sequence:nn {<label a>} {<label b>}
5141 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5142 {
5143   \exp_args:Nxx \tl_if_eq:nnT
5144     { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5145     { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5146     {
5147       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5148         {
5149           \exp_args:Nxx \tl_if_eq:nnT
5150             { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5151             { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5152             {
5153               \int_compare:nNnTF
5154                 { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5155                 =
5156                 { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }

```

```

5157         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5158         {
5159             \int_compare:nNnT
5160             { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5161             =
5162             { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5163             {
5164                 \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5165                 \bool_set_true:N \l__zrefclever_next_is_same_bool
5166             }
5167         }
5168     }
5169 }
5170 {
5171     \exp_args:Nxx \tl_if_eq:nnT
5172     { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5173     { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5174     {
5175         \exp_args:Nxx \tl_if_eq:nnT
5176         { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5177         { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5178         {
5179             \int_compare:nNnTF
5180             { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5181             =
5182             { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5183             { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5184             {
5185                 \int_compare:nNnT
5186                 { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5187                 =
5188                 { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5189                 {

```

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.

```

5190         \exp_args:Nxx \tl_if_eq:nnT
5191         {
5192             \__zrefclever_extract_unexp:nvn {#1}
5193             { l__zrefclever_ref_property_tl } { }
5194         }
5195         {
5196             \__zrefclever_extract_unexp:nvn {#2}
5197             { l__zrefclever_ref_property_tl } { }
5198         }
5199         {
5200             \bool_set_true:N
5201             \l__zrefclever_next_maybe_range_bool
5202             \bool_set_true:N
5203             \l__zrefclever_next_is_same_bool
5204         }

```

```

5205     }
5206   }
5207 }
5208 }
5209 }
5210 }
5211 }

```

(End definition for `_zrefclever_labels_in_sequence:nn`.)

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

```

\_zrefclever_get_rf_opt_tl:nnnN
    \_zrefclever_get_rf_opt_tl:nnnN {<option>}
      {<ref type>} {<language>} {<tl variable>}
5212 \cs_new_protected:Npn \_zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5213   {
5214     % First attempt: general options.
5215     \_zrefclever_opt_tl_get:cNF
5216     { \_zrefclever_opt_varname_general:nn {#1} { tl } }
5217     #4
5218     {
5219       % If not found, try type specific options.
5220       \_zrefclever_opt_tl_get:cNF
5221       { \_zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5222       #4
5223       {
5224         % If not found, try type- and language-specific.
5225         \_zrefclever_opt_tl_get:cNF
5226         { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5227         #4
5228         {
5229           % If not found, try language-specific default.
5230           \_zrefclever_opt_tl_get:cNF
5231           { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5232           #4
5233           {
5234             % If not found, try fallback.
5235             \_zrefclever_opt_tl_get:cNF
5236             { \_zrefclever_opt_varname_fallback:nn {#1} { tl } }
5237             #4
5238             { \tl_clear:N #4 }
5239           }
5240         }
5241       }
5242     }
5243   }
5244 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_tl:nnnN { nxxN }

```

(End definition for `_zrefclever_get_rf_opt_tl:nnnN`.)

```

\_zrefclever_get_rf_opt_seq:nnnN
    \_zrefclever_get_rf_opt_seq:nnnN {<option>}
      {<ref type>} {<language>} {<seq variable>}

```

```

5245 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5246 {
5247   % First attempt: general options.
5248   \__zrefclever_opt_seq_get:cNF
5249   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5250   #4
5251   {
5252     % If not found, try type specific options.
5253     \__zrefclever_opt_seq_get:cNF
5254     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5255     #4
5256     {
5257       % If not found, try type- and language-specific.
5258       \__zrefclever_opt_seq_get:cNF
5259       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5260       #4
5261       {
5262         % If not found, try language-specific default.
5263         \__zrefclever_opt_seq_get:cNF
5264         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5265         #4
5266         {
5267           % If not found, try fallback.
5268           \__zrefclever_opt_seq_get:cNF
5269           { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5270           #4
5271           { \seq_clear:N #4 }
5272         }
5273       }
5274     }
5275   }
5276 }
5277 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { nxxN }

```

(End definition for __zrefclever_get_rf_opt_seq:nnnN.)

__zrefclever_get_rf_opt_bool:nnnnN

```

\__zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
  {<ref type>} {<language>} {<bool variable>}
5278 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5279 {
5280   % First attempt: general options.
5281   \__zrefclever_opt_bool_get:cNF
5282   { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5283   #5
5284   {
5285     % If not found, try type specific options.
5286     \__zrefclever_opt_bool_get:cNF
5287     { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5288     #5
5289     {
5290       % If not found, try type- and language-specific.
5291       \__zrefclever_opt_bool_get:cNF
5292       { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5293       #5

```

```

5294     {
5295     % If not found, try language-specific default.
5296     \__zrefclever_opt_bool_get:cNF
5297     { \__zrefclever_opt_varname_lang_default:nm {#4} {#1} { bool } }
5298     #5
5299     {
5300     % If not found, try fallback.
5301     \__zrefclever_opt_bool_get:cNF
5302     { \__zrefclever_opt_varname_fallback:nm {#1} { bool } }
5303     #5
5304     { \use:c { bool_set_ #2 :N } #5 }
5305     }
5306     }
5307   }
5308 }
5309 }
5310 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nmmnN { nxxxN }

```

(End definition for `__zrefclever_get_rf_opt_bool:nmmnN`.)

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.

```

5311 \__zrefclever_compat_module:nm { appendix }
5312 {
5313   \AddToHook { cmd / appendix / before }
5314   {
5315     \__zrefclever_zcsetup:n
5316     {
5317       countertype =
5318       {

```

```

5319         chapter      = appendix ,
5320         section      = appendix ,
5321         subsection   = appendix ,
5322         subsubsection = appendix ,
5323         paragraph    = appendix ,
5324         subparagraph = appendix ,
5325     }
5326 }
5327 }
5328 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmdhooks` (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.

```

5329 \__zrefclever_compat_module:nn { appendices }
5330 {
5331   \__zrefclever_if_package_loaded:nT { appendix }
5332   {
5333     \newcounter { zc@appendix }
5334     \newcounter { zc@save@appendix }
5335     \setcounter { zc@appendix } { 0 }
5336     \setcounter { zc@save@appendix } { 0 }
5337     \cs_if_exist:cTF { chapter }
5338     {
5339       \__zrefclever_zcsetup:n
5340       { counterresetby = { chapter = zc@appendix } }
5341     }
5342     {
5343       \cs_if_exist:cT { section }
5344       {
5345         \__zrefclever_zcsetup:n
5346         { counterresetby = { section = zc@appendix } }
5347       }
5348     }
5349     \AddToHook { env / appendices / begin }
5350     {
5351       \stepcounter { zc@save@appendix }
5352       \setcounter { zc@appendix } { \value { zc@save@appendix } }
5353       \__zrefclever_zcsetup:n
5354       {
5355         countertype =
5356         {
5357           chapter      = appendix ,

```

```

5358         section      = appendix ,
5359         subsection   = appendix ,
5360         subsubsection = appendix ,
5361         paragraph    = appendix ,
5362         subparagraph  = appendix ,
5363     }
5364 }
5365 }
5366 \AddToHook { env / appendices / end }
5367 { \setcounter { zc@appendix } { 0 } }
5368 \AddToHook { cmd / appendix / before }
5369 {
5370     \stepcounter { zc@save@appendix }
5371     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5372 }
5373 \AddToHook { env / subappendices / begin }
5374 {
5375     \__zrefclever_zcsetup:n
5376     {
5377         countertype =
5378         {
5379             section      = appendix ,
5380             subsection   = appendix ,
5381             subsubsection = appendix ,
5382             paragraph    = appendix ,
5383             subparagraph  = appendix ,
5384         } ,
5385     }
5386 }
5387 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5388 }
5389 }

```

9.3 memoir

The memoir document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. Some of them are implemented in ways which make difficult the use of zref, particularly `\zlabel`, short of redefining the whole stuff ourselves. Hopefully, these features are specialized enough to make zref-clever useful enough with memoir without much friction, but unless some support is added upstream, it is difficult not to be a little intrusive here.

1. Caption functionality which receives $\langle label \rangle$ as optional argument, namely:

- (a) The `sidecaption` and `sidecontcaption` environments. These environments *store* the label in an internal macro, `\m@mscaplabel`, at the begin environment code (more precisely in `\@@sidecaption`), but both the call to `\refstepcounter` and the expansion of `\m@mscaplabel` take place at `\endsidecaption`. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the $\langle label \rangle$ argument to start with. I can see two ways to deal with these environments, none of which I really like. First, map through `\m@mscaplabel` until `\label` is found, then grab the next token

which is the $\langle label \rangle$. This can be used to set a $\backslash zlabel$ either with a kernel environment hook, or with $\backslash @mem@scap@afterhook$ (the former requires running $\backslash refstepcounter$ on our own, since the $env/.../end$ hook comes before this is done by $\backslash endsidecaption$). Second, locally redefine $\backslash label$ to set both labels inside the environments.

- (b) The bilingual caption commands: $\backslash bitwonumcaption$, $\backslash bionenumcaption$, and $\backslash bicaption$. These commands do not support setting the label in their arguments (the labels do get set, but they end up included in the `title` property of the label too). So we do the same for them as for $\backslash sidecaption$, just taking care of grouping, since we can't count on the convenience of the environment hook (luckily for us, they are scoped themselves, so we can add an extra group there).
- 2. The $\backslash subcaptionref$ command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside the $\langle subtitle \rangle$ argument of the subcaptioning commands, namely: $\backslash subcaption$, $\backslash contsubcaption$, $\backslash subbottom$, $\backslash contsubbottom$, $\backslash subtop$. This functionality is implemented by `memoir` by setting a *second label* with prefix $\backslash sub@{label}$, and storing there just that part of interest. With $\backslash zref$ this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find a place to hook into to add the property to the `main` list, since `memoir` does not really consider the possibility of some other command setting labels. $\backslash @memsubcaption$ is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.
- 3. `memoir`'s $\backslash footnote$, $\backslash verbfootnote$, $\backslash sidefootnote$ and $\backslash pagenote$, just as the regular $\backslash footnote$ until recently in the kernel, do not set $\backslash @currentcounter$ alongside $\backslash @currentlabel$, proper referencing to them requires setting the type for it.
- 4. Note that `memoir`'s appendix features “emulates” the `appendix` package, hence the corresponding compatibility module is loaded for `memoir` even if that package is not itself loaded. The same is true for the $\backslash appendix$ command module, since it is also defined.

```

5390 \__zrefclever_compat_module:nn { memoir }
5391   {
5392     \__zrefclever_if_class_loaded:nT { memoir }
5393     {

```

Add subfigure and subtable support out of the box. Technically, this is not “default” behavior for `memoir`, users have to enable it with $\backslash newsfloat$, but let this be smooth. Still, this does not cover any other floats created with $\backslash newfloat$. Also include setup for `verse`.

```

5394     \__zrefclever_zcsetup:n
5395     {
5396       countertype =
5397       {
5398         subfigure = figure ,
5399         subtable  = table  ,
5400         poemline  = line   ,
5401       } ,
5402       counterresetby =

```

```

5403     {
5404         subfigure = figure ,
5405         subtable = table ,
5406     } ,
5407 }

```

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

```

5408 \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5409 {
5410     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5411     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5412     {
5413         \__zrefclever_memoir_orig_label:n {##1}
5414         \zlabel{##1}
5415     }
5416     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
5417 }
5418 \AddToHook { env / sidecaption / begin }
5419 { \__zrefclever_memoir_both_labels: }
5420 \AddToHook { env / sidecontcaption / begin }
5421 { \__zrefclever_memoir_both_labels: }
5422 \AddToHook{ cmd / bitwonumcaption / before }
5423 { \group_begin: \__zrefclever_memoir_both_labels: }
5424 \AddToHook{ cmd / bitwonumcaption / after }
5425 { \group_end: }
5426 \AddToHook{ cmd / bionenumcaption / before }
5427 { \group_begin: \__zrefclever_memoir_both_labels: }
5428 \AddToHook{ cmd / bionenumcaption / after }
5429 { \group_end: }
5430 \AddToHook{ cmd / bicaption / before }
5431 { \group_begin: \__zrefclever_memoir_both_labels: }
5432 \AddToHook{ cmd / bicaption / after }
5433 { \group_end: }

```

Support for subcaption reference.

```

5434 \zref@newprop { subcaption }
5435 { \cs_if_exist_use:c { @@thesub \@capttype } }
5436 \AddToHook{ cmd / @memsubcaption / before }
5437 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5438 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5439 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5440 \AddToHook { env / minipage / begin }
5441 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5442 \AddToHook { cmd / @makefntext / before }
5443 {
5444     \__zrefclever_zcsetup:x
5445     { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5446 }
5447 \AddToHook { cmd / @makesidefntext / before }
5448 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5449 \__zrefclever_zcsetup:n
5450 {

```

```

5451         countertype =
5452         {
5453             sidefootnote = footnote ,
5454             pagenote = endnote ,
5455         } ,
5456     }
5457     \AddToHook { file / \jobname.ent / before }
5458     { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5459     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5460 }
5461 }

```

9.4 amsmath

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

```

5462 \__zrefclever_compat_module:nn { amsmath }
5463 {
5464     \__zrefclever_if_package_loaded:nT { amsmath }
5465     {

```

First, we define a function for label setting inside `amsmath` math environments, we want it to set both `\zlabel` and `\label`. We may “get a ride”, but not steal the place altogether. This makes for potentially redundant labels, but seems a good compromise. We *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that’s precisely the case inside the `multiline` environment (and, damn!, I took a beating of this detail...). See also <https://github.com/ho-tex/zref/issues/4> and <https://github.com/latex3/latex2e/issues/272>.

```

5466         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5467         {
5468             \__zrefclever_orig_ltxlabel:n {#1}
5469             \zref@wrapper@babel \zref@label {#1}
5470         }

```

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside `amsmath`’s math environments. And, after that, redefine it to be `__zrefclever_ltxlabel:n` instead. We must handle `hyperref` here, which comes very late in the preamble, and which loads `nameref` at `begindocument` (though this has changed recently 2022-05-16, see <https://github.com/latex3/hyperref/commit/a011ba9308a1b047dc151796de557da0bb22abaa>), which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. Other classes packages also redefine `\ltx@label`, which may cause some trouble. A `grep` on `texmf-dist` returns hits for: `thm-restate.sty`, `smartref.sty`, `jmlrbook.cls`, `cleveref.sty`, `cryptocode.sty`, `nameref.sty`, `easyeqn.sty`, `empheq.sty`, `ntheorem.sty`, `nccmath.sty`, `nwejm.cls`, `nwejmart.cls`, `aguplus.sty`, `aguplus.cls`, `agupp.sty`, `amsmath.hyp`, `amsmath.sty` (surprise!), `amsmath.4ht`, `nameref.4ht`, `frenchle.sty`, `french.sty`, plus corresponding documentations and different versions of the same packages. That’s not too many, but not “just a few” either. The critical ones are explicitly handled here (`amsmath` itself, and `nameref`). A number of those I’m really not acquainted with. For `cleveref`, in particular, this procedure is not compatible with it. If we happen to come later than it and override its definition, this may be a substantial problem for `cleveref`, since it will

find the label, but it won't contain the data it is expecting. However, this should normally not occur, if the user has followed the documented recommendation for `cleveref` to load it last, or at least very late, and besides I don't see much of an use case for using both `cleveref` and `zref-clever` together. I have documented in the user manual that this module may cause potential issues, and how to work around them. And I have made an upstream feature request for a hook, so that this could be made more cleanly at <https://github.com/latex3/hyperref/issues/212>.

```

5471     \__zrefclever_if_package_loaded:nTF { hyperref }
5472     {
5473         \AddToHook { package / nameref / after }
5474         {
5475             \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5476             \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5477         }
5478     }
5479     {
5480         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5481         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5482     }

```

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).

```

5483     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5484     \AddToHook { env / subequations / begin }
5485     {
5486         \__zrefclever_zcsetup:x
5487         {
5488             counterresetby =
5489             {
5490                 parentequation =
5491                 \__zrefclever_counter_reset_by:n { equation } ,
5492                 equation = parentequation ,
5493             } ,
5494             currentcounter = parentequation ,
5495             countertype = { parentequation = equation } ,
5496         }
5497         \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5498     }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout and does set `\@currentcounter` for `\tags`. But we still have to manually reset `currentcounter` to default because, since we had to manually set `currentcounter` to `parentequation` in `subequations`, we also have to manually set it to `equation` in environments which may be used within it. The `xxalignat` environment is not included, because it is “starred”

by default (i.e. unnumbered), and does not display or accepts labels or tags anyway. The `-ed` (`gathered`, `aligned`, and `alignedat`) and `cases` environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like `array`, `matrix` and variations. Finally, `split` too can only be used as part of another environment. We also arrange, at this point, for the provision of the `subeq` property, for the convenience of referring to them directly or to build terse ranges with the `endrange` option.

```

5499     \zref@newprop { subeq } { \alph { equation } }
5500     \clist_map_inline:nn
5501     {
5502         equation ,
5503         equation* ,
5504         align ,
5505         align* ,
5506         alignat ,
5507         alignat* ,
5508         flalign ,
5509         flalign* ,
5510         xalignat ,
5511         xalignat* ,
5512         gather ,
5513         gather* ,
5514         multiline ,
5515         multiline* ,
5516     }
5517     {
5518         \AddToHook { env / #1 / begin }
5519         {
5520             \__zrefclever_zcsetup:n { currentcounter = equation }
5521             \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5522                 { \zref@localaddprop \ZREF@mainlist { subeq } }
5523         }
5524     }
5525     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5526 }
5527 }

```

9.5 mathtools

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

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

```

5528 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5529 \__zrefclever_compat_module:nn { mathtools }
5530 {
5531     \__zrefclever_if_package_loaded:nT { mathtools }

```

```

5532 {
5533   \MH_if_boolean:nT { show_only_refs }
5534   {
5535     \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5536     \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5537     {
5538       \@bsphack
5539       \seq_map_inline:Nn #1
5540       {
5541         \exp_args:Nx \tl_if_eq:nnTF
5542         { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5543         { equation }
5544         {
5545           \protected@write \@auxout { }
5546           { \string \MT@newlabel {##1} }
5547         }
5548         {
5549           \exp_args:Nx \tl_if_eq:nnT
5550           { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5551           { parentequation }
5552           {
5553             \protected@write \@auxout { }
5554             { \string \MT@newlabel {##1} }
5555           }
5556         }
5557       }
5558     }
5559   }
5560   \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5561 }
5562 }
5563 }

```

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. However, if it happens not to work, there was no easy alternative handle I could find. In particular, it does not seem viable to leverage the `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

```

5564 \__zrefclever_compat_module:nn { breqn }
5565 {
5566   \__zrefclever_if_package_loaded:nT { breqn }
5567   {

```

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>).

```

5568   \bool_new:N \l__zrefclever_breqn_dgroup_bool

```

```

5569 \AddToHook { env / dgroup / begin }
5570 {
5571   \__zrefclever_zcsetup:x
5572   {
5573     counterresetby =
5574     {
5575       parentequation =
5576       \__zrefclever_counter_reset_by:n { equation } ,
5577       equation = parentequation ,
5578     } ,
5579     currentcounter = parentequation ,
5580     countertype = { parentequation = equation } ,
5581   }
5582   \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5583 }
5584 \zref@ifpropundefined { subeq }
5585 { \zref@newprop { subeq } { \alph { equation } } }
5586 { }
5587 \clist_map_inline:nn
5588 {
5589   dmath ,
5590   dseries ,
5591   darray ,
5592 }
5593 {
5594   \AddToHook { env / #1 / begin }
5595   {
5596     \__zrefclever_zcsetup:n { currentcounter = equation }
5597     \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5598     { \zref@localaddprop \ZREF@mainlist { subeq } }
5599   }
5600 }
5601 \msg_info:nnn { zref-clever } { compat-package } { breqn }
5602 }
5603 }

```

9.7 listings

```

5604 \__zrefclever_compat_module:nn { listings }
5605 {
5606   \__zrefclever_if_package_loaded:nT { listings }
5607   {
5608     \__zrefclever_zcsetup:n
5609     {
5610       countertype =
5611       {
5612         lstlisting = listing ,
5613         lstnumber = line ,
5614       } ,
5615       counterresetby = { lstnumber = lstlisting } ,
5616     }

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment. The *only* place to set this label is the `PreInit` hook. This hook, comes right

after `\lst@MakeCaption` in `\lst@Init`, which runs `\refstepcounter` on `lstlisting`, so we must come after it. Also `listings` itself sets `\@currentlabel` to `\thelstnumber` in the `Init` hook, which comes right after the `PreInit` one in `\lst@Init`. Since, if we add to `Init` here, we go to the end of it, we'd be seeing the wrong `\@currentlabel` at that point.

```
5617 \lst@AddToHook { PreInit }
5618 { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }
```

Set `currentcounter` to `lstnumber` in the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` here. Note that `listings` *does use* `\refstepcounter` on `lstnumber`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Indeed, the fact that `listings` manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```
5619 \lst@AddToHook { Init }
5620 { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5621 \msg_info:nnn { zref-clever } { compat-package } { listings }
5622 }
5623 }
```

9.8 `enumitem`

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

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

```
5624 \__zrefclever_compat_module:nn { enumitem }
5625 {
5626   \__zrefclever_if_package_loaded:nT { enumitem }
5627   {
5628     \int_set:Nn \l_tmpa_int { 5 }
5629     \bool_while_do:nn
5630     {
5631       \cs_if_exist_p:c
5632       { c@ enum \int_to_roman:n { \l_tmpa_int } }
5633     }
5634     {
5635       \__zrefclever_zcsetup:x
5636       {
5637         counterresetby =
5638         {
5639           enum \int_to_roman:n { \l_tmpa_int } =
```



```

5640         enum \int_to_roman:n { \l_tmpa_int - 1 }
5641     } ,
5642     countertype =
5643         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5644     }
5645     \int_incr:N \l_tmpa_int
5646 }
5647 \int_compare:nNnT { \l_tmpa_int } > { 5 }
5648 { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5649 }
5650 }

```

9.9 subcaption

```

5651 \__zrefclever_compat_module:nn { subcaption }
5652 {
5653     \__zrefclever_if_package_loaded:nT { subcaption }
5654     {
5655         \__zrefclever_zcsetup:n
5656         {
5657             countertype =
5658             {
5659                 subfigure = figure ,
5660                 subtable = table ,
5661             } ,
5662             counterresetby =
5663             {
5664                 subfigure = figure ,
5665                 subtable = table ,
5666             } ,
5667         }
5668     }
5669 }

```

Support for subref reference.

```

5668     \zref@newprop { subref }
5669     { \cs_if_exist_use:c { thesub \@capttype } }
5670     \tl_put_right:Nn \caption@subtypehook
5671     { \zref@localaddprop \ZREF@mainlist { subref } }
5672 }
5673 }

```

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.

```

5674 \__zrefclever_compat_module:nn { subfig }
5675 {
5676     \__zrefclever_if_package_loaded:nT { subfig }
5677     {
5678         \__zrefclever_zcsetup:n
5679         {
5680             countertype =
5681             {
5682                 subfigure = figure ,
5683                 subtable = table ,
5684             } ,
5685         }
5686     }
5687 }

```

```

5685         counterresetby =
5686         {
5687             subfigure = figure ,
5688             subtable = table ,
5689         } ,
5690     }
5691 }
5692 }
5693 </package>

```

10 Language files

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

10.1 Localization guidelines

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

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

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

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

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

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

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

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

Completeness and abbreviated forms: Ideally, the language file should be as complete as possible. “Complete” meaning it contains: i) the defaults for all basic separators, `namesep`, `pairsep`, `listsep`, `lastsep`, `tpairsep`, `tlistsep`, `tlastsep`, `notesep`, and `rangesep`; ii) the non-abbreviated forms of names for all the supported reference types, according to the language definitions, that is, usually for `Name-sg`, `name-sg`,

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

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

Input encoding of language files: When `zref-clever` was released, the \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” `rebounds` options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that's the point where we know from `babel` or `polyglossia` the `\languagename`. But we also don't want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

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

10.2 English

English language file has been initially provided by the author.

5694 `*package`

```

5695 \zcDeclareLanguage { english }
5696 \zcDeclareLanguageAlias { american } { english }
5697 \zcDeclareLanguageAlias { australian } { english }
5698 \zcDeclareLanguageAlias { british } { english }
5699 \zcDeclareLanguageAlias { canadian } { english }
5700 \zcDeclareLanguageAlias { newzealand } { english }
5701 \zcDeclareLanguageAlias { UKenglish } { english }
5702 \zcDeclareLanguageAlias { USenglish } { english }
5703 \end{package}

5704 (*lang-english)

5705 namesep = {\nobreakspace} ,
5706 pairsep = {\and\nobreakspace} ,
5707 listsep = {,~} ,
5708 lastsep = {\and\nobreakspace} ,
5709 tpairsep = {\and\nobreakspace} ,
5710 tlistsep = {,~} ,
5711 tlastsep = {,~\and\nobreakspace} ,
5712 notesep = {~} ,
5713 rangesep = {\to\nobreakspace} ,
5714
5715 type = book ,
5716 Name-sg = Book ,
5717 name-sg = book ,
5718 Name-pl = Books ,
5719 name-pl = books ,
5720
5721 type = part ,
5722 Name-sg = Part ,
5723 name-sg = part ,
5724 Name-pl = Parts ,
5725 name-pl = parts ,
5726
5727 type = chapter ,
5728 Name-sg = Chapter ,
5729 name-sg = chapter ,
5730 Name-pl = Chapters ,
5731 name-pl = chapters ,
5732
5733 type = section ,
5734 Name-sg = Section ,
5735 name-sg = section ,
5736 Name-pl = Sections ,
5737 name-pl = sections ,
5738
5739 type = paragraph ,
5740 Name-sg = Paragraph ,
5741 name-sg = paragraph ,
5742 Name-pl = Paragraphs ,
5743 name-pl = paragraphs ,
5744 Name-sg-ab = Par. ,
5745 name-sg-ab = par. ,
5746 Name-pl-ab = Par. ,
5747 name-pl-ab = par. ,

```

```

5748
5749 type = appendix ,
5750   Name-sg = Appendix ,
5751   name-sg = appendix ,
5752   Name-pl = Appendices ,
5753   name-pl = appendices ,
5754
5755 type = page ,
5756   Name-sg = Page ,
5757   name-sg = page ,
5758   Name-pl = Pages ,
5759   name-pl = pages ,
5760   rangeseq = {\textendash} ,
5761   rangetopair = false ,
5762
5763 type = line ,
5764   Name-sg = Line ,
5765   name-sg = line ,
5766   Name-pl = Lines ,
5767   name-pl = lines ,
5768
5769 type = figure ,
5770   Name-sg = Figure ,
5771   name-sg = figure ,
5772   Name-pl = Figures ,
5773   name-pl = figures ,
5774   Name-sg-ab = Fig. ,
5775   name-sg-ab = fig. ,
5776   Name-pl-ab = Figs. ,
5777   name-pl-ab = figs. ,
5778
5779 type = table ,
5780   Name-sg = Table ,
5781   name-sg = table ,
5782   Name-pl = Tables ,
5783   name-pl = tables ,
5784
5785 type = item ,
5786   Name-sg = Item ,
5787   name-sg = item ,
5788   Name-pl = Items ,
5789   name-pl = items ,
5790
5791 type = footnote ,
5792   Name-sg = Footnote ,
5793   name-sg = footnote ,
5794   Name-pl = Footnotes ,
5795   name-pl = footnotes ,
5796
5797 type = endnote ,
5798   Name-sg = Note ,
5799   name-sg = note ,
5800   Name-pl = Notes ,
5801   name-pl = notes ,

```

```

5802
5803 type = note ,
5804   Name-sg = Note ,
5805   name-sg = note ,
5806   Name-pl = Notes ,
5807   name-pl = notes ,
5808
5809 type = equation ,
5810   Name-sg = Equation ,
5811   name-sg = equation ,
5812   Name-pl = Equations ,
5813   name-pl = equations ,
5814   Name-sg-ab = Eq. ,
5815   name-sg-ab = eq. ,
5816   Name-pl-ab = Eqs. ,
5817   name-pl-ab = eqs. ,
5818   refbounds-first-sg = {,(,),} ,
5819   refbounds = {(,,)} ,
5820
5821 type = theorem ,
5822   Name-sg = Theorem ,
5823   name-sg = theorem ,
5824   Name-pl = Theorems ,
5825   name-pl = theorems ,
5826
5827 type = lemma ,
5828   Name-sg = Lemma ,
5829   name-sg = lemma ,
5830   Name-pl = Lemmas ,
5831   name-pl = lemmas ,
5832
5833 type = corollary ,
5834   Name-sg = Corollary ,
5835   name-sg = corollary ,
5836   Name-pl = Corollaries ,
5837   name-pl = corollaries ,
5838
5839 type = proposition ,
5840   Name-sg = Proposition ,
5841   name-sg = proposition ,
5842   Name-pl = Propositions ,
5843   name-pl = propositions ,
5844
5845 type = definition ,
5846   Name-sg = Definition ,
5847   name-sg = definition ,
5848   Name-pl = Definitions ,
5849   name-pl = definitions ,
5850
5851 type = proof ,
5852   Name-sg = Proof ,
5853   name-sg = proof ,
5854   Name-pl = Proofs ,
5855   name-pl = proofs ,

```

```

5856
5857 type = result ,
5858   Name-sg = Result ,
5859   name-sg = result ,
5860   Name-pl = Results ,
5861   name-pl = results ,
5862
5863 type = remark ,
5864   Name-sg = Remark ,
5865   name-sg = remark ,
5866   Name-pl = Remarks ,
5867   name-pl = remarks ,
5868
5869 type = example ,
5870   Name-sg = Example ,
5871   name-sg = example ,
5872   Name-pl = Examples ,
5873   name-pl = examples ,
5874
5875 type = algorithm ,
5876   Name-sg = Algorithm ,
5877   name-sg = algorithm ,
5878   Name-pl = Algorithms ,
5879   name-pl = algorithms ,
5880
5881 type = listing ,
5882   Name-sg = Listing ,
5883   name-sg = listing ,
5884   Name-pl = Listings ,
5885   name-pl = listings ,
5886
5887 type = exercise ,
5888   Name-sg = Exercise ,
5889   name-sg = exercise ,
5890   Name-pl = Exercises ,
5891   name-pl = exercises ,
5892
5893 type = solution ,
5894   Name-sg = Solution ,
5895   name-sg = solution ,
5896   Name-pl = Solutions ,
5897   name-pl = solutions ,
5898 </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`.

```

5899 (*package)
5900 \zcDeclareLanguage
5901 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5902 { german }

```



```

5903 \zcDeclareLanguageAlias { ngerman      } { german }
5904 \zcDeclareLanguageAlias { austrian     } { german }
5905 \zcDeclareLanguageAlias { naustrian     } { german }
5906 \zcDeclareLanguageAlias { swissgerman  } { german }
5907 \zcDeclareLanguageAlias { nswissgerman } { german }
5908 </package>

5909 <*lang-german>

5910 namesep = {\nobreakspace} ,
5911 pairsep  = {\~und\nobreakspace} ,
5912 listsep  = {,~} ,
5913 lastsep  = {\~und\nobreakspace} ,
5914 tpairsep = {\~und\nobreakspace} ,
5915 tlistsep = {,~} ,
5916 tlastsep = {\~und\nobreakspace} ,
5917 notesep  = {~} ,
5918 rangeseq = {\~bis\nobreakspace} ,
5919
5920 type = book ,
5921   gender = n ,
5922   case = N ,
5923     Name-sg = Buch ,
5924     Name-pl = Bücher ,
5925   case = A ,
5926     Name-sg = Buch ,
5927     Name-pl = Bücher ,
5928   case = D ,
5929     Name-sg = Buch ,
5930     Name-pl = Büchern ,
5931   case = G ,
5932     Name-sg = Buches ,
5933     Name-pl = Bücher ,
5934
5935 type = part ,
5936   gender = m ,
5937   case = N ,
5938     Name-sg = Teil ,
5939     Name-pl = Teile ,
5940   case = A ,
5941     Name-sg = Teil ,
5942     Name-pl = Teile ,
5943   case = D ,
5944     Name-sg = Teil ,
5945     Name-pl = Teilen ,
5946   case = G ,
5947     Name-sg = Teiles ,
5948     Name-pl = Teile ,
5949
5950 type = chapter ,
5951   gender = n ,
5952   case = N ,
5953     Name-sg = Kapitel ,
5954     Name-pl = Kapitel ,
5955   case = A ,

```

```

5956     Name-sg = Kapitel ,
5957     Name-pl = Kapitel ,
5958     case = D ,
5959     Name-sg = Kapitel ,
5960     Name-pl = Kapiteln ,
5961     case = G ,
5962     Name-sg = Kapitels ,
5963     Name-pl = Kapitel ,
5964
5965 type = section ,
5966     gender = m ,
5967     case = N ,
5968     Name-sg = Abschnitt ,
5969     Name-pl = Abschnitte ,
5970     case = A ,
5971     Name-sg = Abschnitt ,
5972     Name-pl = Abschnitte ,
5973     case = D ,
5974     Name-sg = Abschnitt ,
5975     Name-pl = Abschnitten ,
5976     case = G ,
5977     Name-sg = Abschnitts ,
5978     Name-pl = Abschnitte ,
5979
5980 type = paragraph ,
5981     gender = m ,
5982     case = N ,
5983     Name-sg = Absatz ,
5984     Name-pl = Absätze ,
5985     case = A ,
5986     Name-sg = Absatz ,
5987     Name-pl = Absätze ,
5988     case = D ,
5989     Name-sg = Absatz ,
5990     Name-pl = Absätzen ,
5991     case = G ,
5992     Name-sg = Absatzes ,
5993     Name-pl = Absätze ,
5994
5995 type = appendix ,
5996     gender = m ,
5997     case = N ,
5998     Name-sg = Anhang ,
5999     Name-pl = Anhänge ,
6000     case = A ,
6001     Name-sg = Anhang ,
6002     Name-pl = Anhänge ,
6003     case = D ,
6004     Name-sg = Anhang ,
6005     Name-pl = Anhängen ,
6006     case = G ,
6007     Name-sg = Anhangs ,
6008     Name-pl = Anhänge ,
6009

```

```

6010 type = page ,
6011   gender = f ,
6012   case = N ,
6013     Name-sg = Seite ,
6014     Name-pl = Seiten ,
6015   case = A ,
6016     Name-sg = Seite ,
6017     Name-pl = Seiten ,
6018   case = D ,
6019     Name-sg = Seite ,
6020     Name-pl = Seiten ,
6021   case = G ,
6022     Name-sg = Seite ,
6023     Name-pl = Seiten ,
6024   rangsep = {\textendash} ,
6025   rangetopair = false ,
6026
6027 type = line ,
6028   gender = f ,
6029   case = N ,
6030     Name-sg = Zeile ,
6031     Name-pl = Zeilen ,
6032   case = A ,
6033     Name-sg = Zeile ,
6034     Name-pl = Zeilen ,
6035   case = D ,
6036     Name-sg = Zeile ,
6037     Name-pl = Zeilen ,
6038   case = G ,
6039     Name-sg = Zeile ,
6040     Name-pl = Zeilen ,
6041
6042 type = figure ,
6043   gender = f ,
6044   case = N ,
6045     Name-sg = Abbildung ,
6046     Name-pl = Abbildungen ,
6047     Name-sg-ab = Abb. ,
6048     Name-pl-ab = Abb. ,
6049   case = A ,
6050     Name-sg = Abbildung ,
6051     Name-pl = Abbildungen ,
6052     Name-sg-ab = Abb. ,
6053     Name-pl-ab = Abb. ,
6054   case = D ,
6055     Name-sg = Abbildung ,
6056     Name-pl = Abbildungen ,
6057     Name-sg-ab = Abb. ,
6058     Name-pl-ab = Abb. ,
6059   case = G ,
6060     Name-sg = Abbildung ,
6061     Name-pl = Abbildungen ,
6062     Name-sg-ab = Abb. ,
6063     Name-pl-ab = Abb. ,

```

```

6064
6065 type = table ,
6066     gender = f ,
6067     case = N ,
6068         Name-sg = Tabelle ,
6069         Name-pl = Tabellen ,
6070     case = A ,
6071         Name-sg = Tabelle ,
6072         Name-pl = Tabellen ,
6073     case = D ,
6074         Name-sg = Tabelle ,
6075         Name-pl = Tabellen ,
6076     case = G ,
6077         Name-sg = Tabelle ,
6078         Name-pl = Tabellen ,
6079
6080 type = item ,
6081     gender = m ,
6082     case = N ,
6083         Name-sg = Punkt ,
6084         Name-pl = Punkte ,
6085     case = A ,
6086         Name-sg = Punkt ,
6087         Name-pl = Punkte ,
6088     case = D ,
6089         Name-sg = Punkt ,
6090         Name-pl = Punkten ,
6091     case = G ,
6092         Name-sg = Punktes ,
6093         Name-pl = Punkte ,
6094
6095 type = footnote ,
6096     gender = f ,
6097     case = N ,
6098         Name-sg = Fußnote ,
6099         Name-pl = Fußnoten ,
6100     case = A ,
6101         Name-sg = Fußnote ,
6102         Name-pl = Fußnoten ,
6103     case = D ,
6104         Name-sg = Fußnote ,
6105         Name-pl = Fußnoten ,
6106     case = G ,
6107         Name-sg = Fußnote ,
6108         Name-pl = Fußnoten ,
6109
6110 type = endnote ,
6111     gender = f ,
6112     case = N ,
6113         Name-sg = Endnote ,
6114         Name-pl = Endnoten ,
6115     case = A ,
6116         Name-sg = Endnote ,
6117         Name-pl = Endnoten ,

```

```

6118 case = D ,
6119     Name-sg = Endnote ,
6120     Name-pl = Endnoten ,
6121 case = G ,
6122     Name-sg = Endnote ,
6123     Name-pl = Endnoten ,
6124
6125 type = note ,
6126     gender = f ,
6127     case = N ,
6128         Name-sg = Anmerkung ,
6129         Name-pl = Anmerkungen ,
6130 case = A ,
6131     Name-sg = Anmerkung ,
6132     Name-pl = Anmerkungen ,
6133 case = D ,
6134     Name-sg = Anmerkung ,
6135     Name-pl = Anmerkungen ,
6136 case = G ,
6137     Name-sg = Anmerkung ,
6138     Name-pl = Anmerkungen ,
6139
6140 type = equation ,
6141     gender = f ,
6142     case = N ,
6143         Name-sg = Gleichung ,
6144         Name-pl = Gleichungen ,
6145 case = A ,
6146     Name-sg = Gleichung ,
6147     Name-pl = Gleichungen ,
6148 case = D ,
6149     Name-sg = Gleichung ,
6150     Name-pl = Gleichungen ,
6151 case = G ,
6152     Name-sg = Gleichung ,
6153     Name-pl = Gleichungen ,
6154     refbounds-first-sg = {,(,)},
6155     refbounds = {(,,)} ,
6156
6157 type = theorem ,
6158     gender = n ,
6159     case = N ,
6160         Name-sg = Theorem ,
6161         Name-pl = Theoreme ,
6162 case = A ,
6163     Name-sg = Theorem ,
6164     Name-pl = Theoreme ,
6165 case = D ,
6166     Name-sg = Theorem ,
6167     Name-pl = Theoremen ,
6168 case = G ,
6169     Name-sg = Theorems ,
6170     Name-pl = Theoreme ,
6171

```

```

6172 type = lemma ,
6173   gender = n ,
6174   case = N ,
6175     Name-sg = Lemma ,
6176     Name-pl = Lemmata ,
6177   case = A ,
6178     Name-sg = Lemma ,
6179     Name-pl = Lemmata ,
6180   case = D ,
6181     Name-sg = Lemma ,
6182     Name-pl = Lemmata ,
6183   case = G ,
6184     Name-sg = Lemmas ,
6185     Name-pl = Lemmata ,
6186
6187 type = corollary ,
6188   gender = n ,
6189   case = N ,
6190     Name-sg = Korollar ,
6191     Name-pl = Korollare ,
6192   case = A ,
6193     Name-sg = Korollar ,
6194     Name-pl = Korollare ,
6195   case = D ,
6196     Name-sg = Korollar ,
6197     Name-pl = Korollaren ,
6198   case = G ,
6199     Name-sg = Korollars ,
6200     Name-pl = Korollare ,
6201
6202 type = proposition ,
6203   gender = m ,
6204   case = N ,
6205     Name-sg = Satz ,
6206     Name-pl = Sätze ,
6207   case = A ,
6208     Name-sg = Satz ,
6209     Name-pl = Sätze ,
6210   case = D ,
6211     Name-sg = Satz ,
6212     Name-pl = Sätzen ,
6213   case = G ,
6214     Name-sg = Satzes ,
6215     Name-pl = Sätze ,
6216
6217 type = definition ,
6218   gender = f ,
6219   case = N ,
6220     Name-sg = Definition ,
6221     Name-pl = Definitionen ,
6222   case = A ,
6223     Name-sg = Definition ,
6224     Name-pl = Definitionen ,
6225   case = D ,

```

```

6226     Name-sg = Definition ,
6227     Name-pl = Definitionen ,
6228     case = G ,
6229     Name-sg = Definition ,
6230     Name-pl = Definitionen ,
6231
6232 type = proof ,
6233     gender = m ,
6234     case = N ,
6235     Name-sg = Beweis ,
6236     Name-pl = Beweise ,
6237     case = A ,
6238     Name-sg = Beweis ,
6239     Name-pl = Beweise ,
6240     case = D ,
6241     Name-sg = Beweis ,
6242     Name-pl = Beweisen ,
6243     case = G ,
6244     Name-sg = Beweises ,
6245     Name-pl = Beweise ,
6246
6247 type = result ,
6248     gender = n ,
6249     case = N ,
6250     Name-sg = Ergebnis ,
6251     Name-pl = Ergebnisse ,
6252     case = A ,
6253     Name-sg = Ergebnis ,
6254     Name-pl = Ergebnisse ,
6255     case = D ,
6256     Name-sg = Ergebnis ,
6257     Name-pl = Ergebnissen ,
6258     case = G ,
6259     Name-sg = Ergebnisses ,
6260     Name-pl = Ergebnisse ,
6261
6262 type = remark ,
6263     gender = f ,
6264     case = N ,
6265     Name-sg = Bemerkung ,
6266     Name-pl = Bemerkungen ,
6267     case = A ,
6268     Name-sg = Bemerkung ,
6269     Name-pl = Bemerkungen ,
6270     case = D ,
6271     Name-sg = Bemerkung ,
6272     Name-pl = Bemerkungen ,
6273     case = G ,
6274     Name-sg = Bemerkung ,
6275     Name-pl = Bemerkungen ,
6276
6277 type = example ,
6278     gender = n ,
6279     case = N ,

```

```

6280     Name-sg = Beispiel ,
6281     Name-pl = Beispiele ,
6282     case = A ,
6283     Name-sg = Beispiel ,
6284     Name-pl = Beispiele ,
6285     case = D ,
6286     Name-sg = Beispiel ,
6287     Name-pl = Beispielen ,
6288     case = G ,
6289     Name-sg = Beispiels ,
6290     Name-pl = Beispiele ,
6291
6292     type = algorithm ,
6293     gender = m ,
6294     case = N ,
6295     Name-sg = Algorithmus ,
6296     Name-pl = Algorithmen ,
6297     case = A ,
6298     Name-sg = Algorithmus ,
6299     Name-pl = Algorithmen ,
6300     case = D ,
6301     Name-sg = Algorithmus ,
6302     Name-pl = Algorithmen ,
6303     case = G ,
6304     Name-sg = Algorithmus ,
6305     Name-pl = Algorithmen ,
6306
6307     type = listing ,
6308     gender = n ,
6309     case = N ,
6310     Name-sg = Listing ,
6311     Name-pl = Listings ,
6312     case = A ,
6313     Name-sg = Listing ,
6314     Name-pl = Listings ,
6315     case = D ,
6316     Name-sg = Listing ,
6317     Name-pl = Listings ,
6318     case = G ,
6319     Name-sg = Listings ,
6320     Name-pl = Listings ,
6321
6322     type = exercise ,
6323     gender = f ,
6324     case = N ,
6325     Name-sg = Übungsaufgabe ,
6326     Name-pl = Übungsaufgaben ,
6327     case = A ,
6328     Name-sg = Übungsaufgabe ,
6329     Name-pl = Übungsaufgaben ,
6330     case = D ,
6331     Name-sg = Übungsaufgabe ,
6332     Name-pl = Übungsaufgaben ,
6333     case = G ,

```



```

6334     Name-sg = Übungsaufgabe ,
6335     Name-pl = Übungsaufgaben ,
6336
6337 type = solution ,
6338     gender = f ,
6339     case = N ,
6340     Name-sg = Lösung ,
6341     Name-pl = Lösungen ,
6342     case = A ,
6343     Name-sg = Lösung ,
6344     Name-pl = Lösungen ,
6345     case = D ,
6346     Name-sg = Lösung ,
6347     Name-pl = Lösungen ,
6348     case = G ,
6349     Name-sg = Lösung ,
6350     Name-pl = Lösungen ,
6351 </lang-german>

```

10.4 French

French language file has been initially provided by the author, and has been improved thanks to Denis Bitouzé and François Lagarde (at issue [#1](#)) and participants of the Groupe francophone des Utilisateurs de T_EX (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`.

```

6352 <*package>
6353 \zcDeclareLanguage [ gender = { f , m } ] { french }
6354 \zcDeclareLanguageAlias { acadian } { french }
6355 </package>
6356 <*lang-french>
6357 namesep = {\nobreakspace} ,
6358 pairsep = {\~et\nobreakspace} ,
6359 listsep = { ,~ } ,
6360 lastsep = {\~et\nobreakspace} ,
6361 tpairsep = {\~et\nobreakspace} ,
6362 tlistsep = { ,~ } ,
6363 tlastsep = {\~et\nobreakspace} ,
6364 notesep = {~} ,
6365 rangesep = {\~à\nobreakspace} ,
6366
6367 type = book ,
6368     gender = m ,
6369     Name-sg = Livre ,
6370     name-sg = livre ,
6371     Name-pl = Livres ,
6372     name-pl = livres ,
6373
6374 type = part ,
6375     gender = f ,

```

```

6376 Name-sg = Partie ,
6377 name-sg = partie ,
6378 Name-pl = Parties ,
6379 name-pl = parties ,
6380
6381 type = chapter ,
6382 gender = m ,
6383 Name-sg = Chapitre ,
6384 name-sg = chapitre ,
6385 Name-pl = Chapitres ,
6386 name-pl = chapitres ,
6387
6388 type = section ,
6389 gender = f ,
6390 Name-sg = Section ,
6391 name-sg = section ,
6392 Name-pl = Sections ,
6393 name-pl = sections ,
6394
6395 type = paragraph ,
6396 gender = m ,
6397 Name-sg = Paragraphe ,
6398 name-sg = paragraphe ,
6399 Name-pl = Paragraphes ,
6400 name-pl = paragraphes ,
6401
6402 type = appendix ,
6403 gender = f ,
6404 Name-sg = Annexe ,
6405 name-sg = annexe ,
6406 Name-pl = Annexes ,
6407 name-pl = annexes ,
6408
6409 type = page ,
6410 gender = f ,
6411 Name-sg = Page ,
6412 name-sg = page ,
6413 Name-pl = Pages ,
6414 name-pl = pages ,
6415 rangsep = {-} ,
6416 rangetopair = false ,
6417
6418 type = line ,
6419 gender = f ,
6420 Name-sg = Ligne ,
6421 name-sg = ligne ,
6422 Name-pl = Lignes ,
6423 name-pl = lignes ,
6424
6425 type = figure ,
6426 gender = f ,
6427 Name-sg = Figure ,
6428 name-sg = figure ,
6429 Name-pl = Figures ,

```

```

6430 name-pl = figures ,
6431
6432 type = table ,
6433 gender = f ,
6434 Name-sg = Table ,
6435 name-sg = table ,
6436 Name-pl = Tables ,
6437 name-pl = tables ,
6438
6439 type = item ,
6440 gender = m ,
6441 Name-sg = Point ,
6442 name-sg = point ,
6443 Name-pl = Points ,
6444 name-pl = points ,
6445
6446 type = footnote ,
6447 gender = f ,
6448 Name-sg = Note ,
6449 name-sg = note ,
6450 Name-pl = Notes ,
6451 name-pl = notes ,
6452
6453 type = endnote ,
6454 gender = f ,
6455 Name-sg = Note ,
6456 name-sg = note ,
6457 Name-pl = Notes ,
6458 name-pl = notes ,
6459
6460 type = note ,
6461 gender = f ,
6462 Name-sg = Note ,
6463 name-sg = note ,
6464 Name-pl = Notes ,
6465 name-pl = notes ,
6466
6467 type = equation ,
6468 gender = f ,
6469 Name-sg = Équation ,
6470 name-sg = équation ,
6471 Name-pl = Équations ,
6472 name-pl = équations ,
6473 refbounds-first-sg = {,(,)}, ,
6474 refbounds = {(,,)} ,
6475
6476 type = theorem ,
6477 gender = m ,
6478 Name-sg = Théorème ,
6479 name-sg = théorème ,
6480 Name-pl = Théorèmes ,
6481 name-pl = théorèmes ,
6482
6483 type = lemma ,

```

```

6484   gender = m ,
6485   Name-sg = Lemme ,
6486   name-sg = lemme ,
6487   Name-pl = Lemmes ,
6488   name-pl = lemmes ,
6489
6490   type = corollary ,
6491   gender = m ,
6492   Name-sg = Corollaire ,
6493   name-sg = corollaire ,
6494   Name-pl = Corollaires ,
6495   name-pl = corollaires ,
6496
6497   type = proposition ,
6498   gender = f ,
6499   Name-sg = Proposition ,
6500   name-sg = proposition ,
6501   Name-pl = Propositions ,
6502   name-pl = propositions ,
6503
6504   type = definition ,
6505   gender = f ,
6506   Name-sg = Définition ,
6507   name-sg = définition ,
6508   Name-pl = Définitions ,
6509   name-pl = définitions ,
6510
6511   type = proof ,
6512   gender = f ,
6513   Name-sg = Démonstration ,
6514   name-sg = démonstration ,
6515   Name-pl = Démonstrations ,
6516   name-pl = démonstrations ,
6517
6518   type = result ,
6519   gender = m ,
6520   Name-sg = Résultat ,
6521   name-sg = résultat ,
6522   Name-pl = Résultats ,
6523   name-pl = résultats ,
6524
6525   type = remark ,
6526   gender = f ,
6527   Name-sg = Remarque ,
6528   name-sg = remarque ,
6529   Name-pl = Remarques ,
6530   name-pl = remarques ,
6531
6532   type = example ,
6533   gender = m ,
6534   Name-sg = Exemple ,
6535   name-sg = exemple ,
6536   Name-pl = Exemples ,
6537   name-pl = exemples ,

```

```

6538
6539 type = algorithm ,
6540   gender = m ,
6541   Name-sg = Algorithme ,
6542   name-sg = algorithme ,
6543   Name-pl = Algorithmes ,
6544   name-pl = algorithmes ,
6545
6546 type = listing ,
6547   gender = m ,
6548   Name-sg = Listing ,
6549   name-sg = listing ,
6550   Name-pl = Listings ,
6551   name-pl = listings ,
6552
6553 type = exercise ,
6554   gender = m ,
6555   Name-sg = Exercice ,
6556   name-sg = exercice ,
6557   Name-pl = Exercices ,
6558   name-pl = exercices ,
6559
6560 type = solution ,
6561   gender = f ,
6562   Name-sg = Solution ,
6563   name-sg = solution ,
6564   Name-pl = Solutions ,
6565   name-pl = solutions ,
6566 </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.

```

6567 <*package>
6568 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6569 \zcDeclareLanguageAlias { brazilian } { portuguese }
6570 \zcDeclareLanguageAlias { brazil } { portuguese }
6571 \zcDeclareLanguageAlias { portuges } { portuguese }
6572 </package>
6573 <*lang-portuguese>
6574 namesep = {\nobreakspace} ,
6575 pairsep = {\nobreakspace} ,
6576 listsep = {,~} ,
6577 lastsep = {\nobreakspace} ,
6578 tpairsep = {\nobreakspace} ,
6579 tlistsep = {,~} ,
6580 tlastsep = {\nobreakspace} ,
6581 notesep = {~} ,
6582 rangesep = {\nobreakspace} ,
6583

```

```

6584 type = book ,
6585     gender = m ,
6586     Name-sg = Livro ,
6587     name-sg = livro ,
6588     Name-pl = Livros ,
6589     name-pl = livros ,
6590
6591 type = part ,
6592     gender = f ,
6593     Name-sg = Parte ,
6594     name-sg = parte ,
6595     Name-pl = Partes ,
6596     name-pl = partes ,
6597
6598 type = chapter ,
6599     gender = m ,
6600     Name-sg = Capítulo ,
6601     name-sg = capítulo ,
6602     Name-pl = Capítulos ,
6603     name-pl = capítulos ,
6604
6605 type = section ,
6606     gender = f ,
6607     Name-sg = Seção ,
6608     name-sg = seção ,
6609     Name-pl = Seções ,
6610     name-pl = seções ,
6611
6612 type = paragraph ,
6613     gender = m ,
6614     Name-sg = Parágrafo ,
6615     name-sg = parágrafo ,
6616     Name-pl = Parágrafos ,
6617     name-pl = parágrafos ,
6618     Name-sg-ab = Par. ,
6619     name-sg-ab = par. ,
6620     Name-pl-ab = Par. ,
6621     name-pl-ab = par. ,
6622
6623 type = appendix ,
6624     gender = m ,
6625     Name-sg = Apêndice ,
6626     name-sg = apêndice ,
6627     Name-pl = Apêndices ,
6628     name-pl = apêndices ,
6629
6630 type = page ,
6631     gender = f ,
6632     Name-sg = Página ,
6633     name-sg = página ,
6634     Name-pl = Páginas ,
6635     name-pl = páginas ,
6636     rangesep = {\textendash} ,
6637     rangetopair = false ,

```

```

6638
6639 type = line ,
6640     gender = f ,
6641     Name-sg = Linha ,
6642     name-sg = linha ,
6643     Name-pl = Linhas ,
6644     name-pl = linhas ,
6645
6646 type = figure ,
6647     gender = f ,
6648     Name-sg = Figura ,
6649     name-sg = figura ,
6650     Name-pl = Figuras ,
6651     name-pl = figuras ,
6652     Name-sg-ab = Fig. ,
6653     name-sg-ab = fig. ,
6654     Name-pl-ab = Figs. ,
6655     name-pl-ab = figs. ,
6656
6657 type = table ,
6658     gender = f ,
6659     Name-sg = Tabela ,
6660     name-sg = tabela ,
6661     Name-pl = Tabelas ,
6662     name-pl = tabelas ,
6663
6664 type = item ,
6665     gender = m ,
6666     Name-sg = Item ,
6667     name-sg = item ,
6668     Name-pl = Itens ,
6669     name-pl = itens ,
6670
6671 type = footnote ,
6672     gender = f ,
6673     Name-sg = Nota ,
6674     name-sg = nota ,
6675     Name-pl = Notas ,
6676     name-pl = notas ,
6677
6678 type = endnote ,
6679     gender = f ,
6680     Name-sg = Nota ,
6681     name-sg = nota ,
6682     Name-pl = Notas ,
6683     name-pl = notas ,
6684
6685 type = note ,
6686     gender = f ,
6687     Name-sg = Nota ,
6688     name-sg = nota ,
6689     Name-pl = Notas ,
6690     name-pl = notas ,
6691

```

```

6692 type = equation ,
6693   gender = f ,
6694   Name-sg = Equação ,
6695   name-sg = equação ,
6696   Name-pl = Equações ,
6697   name-pl = equações ,
6698   Name-sg-ab = Eq. ,
6699   name-sg-ab = eq. ,
6700   Name-pl-ab = Eqs. ,
6701   name-pl-ab = eqs. ,
6702   rebounds-first-sg = {,(,)}, ,
6703   rebounds = {(,,)} ,
6704
6705 type = theorem ,
6706   gender = m ,
6707   Name-sg = Teorema ,
6708   name-sg = teorema ,
6709   Name-pl = Teoremas ,
6710   name-pl = teoremas ,
6711
6712 type = lemma ,
6713   gender = m ,
6714   Name-sg = Lema ,
6715   name-sg = lema ,
6716   Name-pl = Lemas ,
6717   name-pl = lemas ,
6718
6719 type = corollary ,
6720   gender = m ,
6721   Name-sg = Corolário ,
6722   name-sg = corolário ,
6723   Name-pl = Corolários ,
6724   name-pl = corolários ,
6725
6726 type = proposition ,
6727   gender = f ,
6728   Name-sg = Proposição ,
6729   name-sg = proposição ,
6730   Name-pl = Proposições ,
6731   name-pl = proposições ,
6732
6733 type = definition ,
6734   gender = f ,
6735   Name-sg = Definição ,
6736   name-sg = definição ,
6737   Name-pl = Definições ,
6738   name-pl = definições ,
6739
6740 type = proof ,
6741   gender = f ,
6742   Name-sg = Demonstração ,
6743   name-sg = demonstração ,
6744   Name-pl = Demonstrações ,
6745   name-pl = demonstrações ,

```



```

6746
6747 type = result ,
6748     gender = m ,
6749     Name-sg = Resultado ,
6750     name-sg = resultado ,
6751     Name-pl = Resultados ,
6752     name-pl = resultados ,
6753
6754 type = remark ,
6755     gender = f ,
6756     Name-sg = Observação ,
6757     name-sg = observação ,
6758     Name-pl = Observações ,
6759     name-pl = observações ,
6760
6761 type = example ,
6762     gender = m ,
6763     Name-sg = Exemplo ,
6764     name-sg = exemplo ,
6765     Name-pl = Exemplos ,
6766     name-pl = exemplos ,
6767
6768 type = algorithm ,
6769     gender = m ,
6770     Name-sg = Algoritmo ,
6771     name-sg = algoritmo ,
6772     Name-pl = Algoritmos ,
6773     name-pl = algoritmos ,
6774
6775 type = listing ,
6776     gender = f ,
6777     Name-sg = Listagem ,
6778     name-sg = listagem ,
6779     Name-pl = Listagens ,
6780     name-pl = listagens ,
6781
6782 type = exercise ,
6783     gender = m ,
6784     Name-sg = Exercício ,
6785     name-sg = exercício ,
6786     Name-pl = Exercícios ,
6787     name-pl = exercícios ,
6788
6789 type = solution ,
6790     gender = f ,
6791     Name-sg = Solução ,
6792     name-sg = solução ,
6793     Name-pl = Soluções ,
6794     name-pl = soluções ,
6795 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```
6796 \langle *package \rangle
6797 \zcdDeclareLanguage [ gender = { f , m } ] { spanish }
6798 \rangle /package \rangle
6799 \langle *lang-spanish \rangle
6800 namesep = {\nobreakspace} ,
6801 pairsep = {\~y\nobreakspace} ,
6802 listsep = { , ~ } ,
6803 lastsep = {\~y\nobreakspace} ,
6804 tpairsep = {\~y\nobreakspace} ,
6805 tlistsep = { , ~ } ,
6806 tlastsep = {\~y\nobreakspace} ,
6807 notesep = { ~ } ,
6808 rangesep = {\~a\nobreakspace} ,
6809
6810 type = book ,
6811   gender = m ,
6812   Name-sg = Libro ,
6813   name-sg = libro ,
6814   Name-pl = Libros ,
6815   name-pl = libros ,
6816
6817 type = part ,
6818   gender = f ,
6819   Name-sg = Parte ,
6820   name-sg = parte ,
6821   Name-pl = Partes ,
6822   name-pl = partes ,
6823
6824 type = chapter ,
6825   gender = m ,
6826   Name-sg = Capítulo ,
6827   name-sg = capítulo ,
6828   Name-pl = Capítulos ,
6829   name-pl = capítulos ,
6830
6831 type = section ,
6832   gender = f ,
6833   Name-sg = Sección ,
6834   name-sg = sección ,
6835   Name-pl = Secciones ,
6836   name-pl = secciones ,
6837
6838 type = paragraph ,
6839   gender = m ,
6840   Name-sg = Párrafo ,
6841   name-sg = párrafo ,
6842   Name-pl = Párrafos ,
6843   name-pl = párrafos ,
6844
6845 type = appendix ,
```

```

6846   gender = m ,
6847   Name-sg = Apéndice ,
6848   name-sg = apéndice ,
6849   Name-pl = Apéndices ,
6850   name-pl = apéndices ,
6851
6852   type = page ,
6853   gender = f ,
6854   Name-sg = Página ,
6855   name-sg = página ,
6856   Name-pl = Páginas ,
6857   name-pl = páginas ,
6858   rangesep = {\textendash} ,
6859   rangetopair = false ,
6860
6861   type = line ,
6862   gender = f ,
6863   Name-sg = Línea ,
6864   name-sg = línea ,
6865   Name-pl = Líneas ,
6866   name-pl = líneas ,
6867
6868   type = figure ,
6869   gender = f ,
6870   Name-sg = Figura ,
6871   name-sg = figura ,
6872   Name-pl = Figuras ,
6873   name-pl = figuras ,
6874
6875   type = table ,
6876   gender = m ,
6877   Name-sg = Cuadro ,
6878   name-sg = cuadro ,
6879   Name-pl = Cuadros ,
6880   name-pl = cuadros ,
6881
6882   type = item ,
6883   gender = m ,
6884   Name-sg = Punto ,
6885   name-sg = punto ,
6886   Name-pl = Puntos ,
6887   name-pl = puntos ,
6888
6889   type = footnote ,
6890   gender = f ,
6891   Name-sg = Nota ,
6892   name-sg = nota ,
6893   Name-pl = Notas ,
6894   name-pl = notas ,
6895
6896   type = endnote ,
6897   gender = f ,
6898   Name-sg = Nota ,
6899   name-sg = nota ,

```

```

6900 Name-pl = Notas ,
6901 name-pl = notas ,
6902
6903 type = note ,
6904 gender = f ,
6905 Name-sg = Nota ,
6906 name-sg = nota ,
6907 Name-pl = Notas ,
6908 name-pl = notas ,
6909
6910 type = equation ,
6911 gender = f ,
6912 Name-sg = Ecuación ,
6913 name-sg = ecuación ,
6914 Name-pl = Ecuaciones ,
6915 name-pl = ecuaciones ,
6916 refbounds-first-sg = {,(,)},
6917 refbounds = {(,,)} ,
6918
6919 type = theorem ,
6920 gender = m ,
6921 Name-sg = Teorema ,
6922 name-sg = teorema ,
6923 Name-pl = Teoremas ,
6924 name-pl = teoremas ,
6925
6926 type = lemma ,
6927 gender = m ,
6928 Name-sg = Lema ,
6929 name-sg = lema ,
6930 Name-pl = Lemas ,
6931 name-pl = lemas ,
6932
6933 type = corollary ,
6934 gender = m ,
6935 Name-sg = Corolario ,
6936 name-sg = corolario ,
6937 Name-pl = Corolarios ,
6938 name-pl = corolarios ,
6939
6940 type = proposition ,
6941 gender = f ,
6942 Name-sg = Proposición ,
6943 name-sg = proposición ,
6944 Name-pl = Proposiciones ,
6945 name-pl = proposiciones ,
6946
6947 type = definition ,
6948 gender = f ,
6949 Name-sg = Definición ,
6950 name-sg = definición ,
6951 Name-pl = Definiciones ,
6952 name-pl = definiciones ,
6953

```

```

6954 type = proof ,
6955     gender = f ,
6956     Name-sg = Demostración ,
6957     name-sg = demostración ,
6958     Name-pl = Demostraciones ,
6959     name-pl = demostraciones ,
6960
6961 type = result ,
6962     gender = m ,
6963     Name-sg = Resultado ,
6964     name-sg = resultado ,
6965     Name-pl = Resultados ,
6966     name-pl = resultados ,
6967
6968 type = remark ,
6969     gender = f ,
6970     Name-sg = Observación ,
6971     name-sg = observación ,
6972     Name-pl = Observaciones ,
6973     name-pl = observaciones ,
6974
6975 type = example ,
6976     gender = m ,
6977     Name-sg = Ejemplo ,
6978     name-sg = ejemplo ,
6979     Name-pl = Ejemplos ,
6980     name-pl = ejemplos ,
6981
6982 type = algorithm ,
6983     gender = m ,
6984     Name-sg = Algoritmo ,
6985     name-sg = algoritmo ,
6986     Name-pl = Algoritmos ,
6987     name-pl = algoritmos ,
6988
6989 type = listing ,
6990     gender = m ,
6991     Name-sg = Listado ,
6992     name-sg = listado ,
6993     Name-pl = Listados ,
6994     name-pl = listados ,
6995
6996 type = exercise ,
6997     gender = m ,
6998     Name-sg = Ejercicio ,
6999     name-sg = ejercicio ,
7000     Name-pl = Ejercicios ,
7001     name-pl = ejercicios ,
7002
7003 type = solution ,
7004     gender = f ,
7005     Name-sg = Solución ,
7006     name-sg = solución ,
7007     Name-pl = Soluciones ,

```

```

7008 name-pl = soluciones ,
7009 </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.

```

7010 <*package>
7011 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7012 </package>
7013 <*lang-dutch>
7014 namesep = {\nobreakspace} ,
7015 pairsep = {\sim\nobreakspace} ,
7016 listsep = { ,\sim } ,
7017 lastsep = {\sim\nobreakspace} ,
7018 tpairsep = {\sim\nobreakspace} ,
7019 tlistsep = { ,\sim } ,
7020 tlastsep = { ,\sim\nobreakspace} ,
7021 notesep = {\sim} ,
7022 rangsep = {\sim/m\nobreakspace} ,
7023
7024 type = book ,
7025 gender = n ,
7026 Name-sg = Boek ,
7027 name-sg = boek ,
7028 Name-pl = Boeken ,
7029 name-pl = boeken ,
7030
7031 type = part ,
7032 gender = n ,
7033 Name-sg = Deel ,
7034 name-sg = deel ,
7035 Name-pl = Delen ,
7036 name-pl = delen ,
7037
7038 type = chapter ,
7039 gender = n ,
7040 Name-sg = Hoofdstuk ,
7041 name-sg = hoofdstuk ,
7042 Name-pl = Hoofdstukken ,
7043 name-pl = hoofdstukken ,
7044
7045 type = section ,
7046 gender = m ,
7047 Name-sg = Paragraaf ,
7048 name-sg = paragraaf ,
7049 Name-pl = Paragrafen ,
7050 name-pl = paragrafen ,
7051
7052 type = paragraph ,
7053 gender = f ,
7054 Name-sg = Alinea ,

```

```

7055 name-sg = alinea ,
7056 Name-pl = Alinea's ,
7057 name-pl = alineas ,
7058

```

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.

```

7059 type = appendix ,
7060 gender = { f , m } ,
7061 Name-sg = Blage ,
7062 name-sg = blage ,
7063 Name-pl = Blagen ,
7064 name-pl = blagen ,
7065
7066 type = page ,
7067 gender = { f , m } ,
7068 Name-sg = Pagina ,
7069 name-sg = pagina ,
7070 Name-pl = Pagina's ,
7071 name-pl = pagina's ,
7072 rangesep = {\textendash} ,
7073 rangetopair = false ,
7074
7075 type = line ,
7076 gender = m ,
7077 Name-sg = Regel ,
7078 name-sg = regel ,
7079 Name-pl = Regels ,
7080 name-pl = regels ,
7081
7082 type = figure ,
7083 gender = { n , f , m } ,
7084 Name-sg = Figuur ,
7085 name-sg = figuur ,
7086 Name-pl = Figuren ,
7087 name-pl = figuren ,
7088
7089 type = table ,
7090 gender = { f , m } ,
7091 Name-sg = Tabel ,
7092 name-sg = tabel ,
7093 Name-pl = Tabellen ,
7094 name-pl = tabellen ,
7095
7096 type = item ,
7097 gender = n ,
7098 Name-sg = Punt ,
7099 name-sg = punt ,
7100 Name-pl = Punten ,
7101 name-pl = punten ,
7102
7103 type = footnote ,
7104 gender = { f , m } ,

```

```

7105 Name-sg = Voetnoot ,
7106 name-sg = voetnoot ,
7107 Name-pl = Voetnoten ,
7108 name-pl = voetnoten ,
7109
7110 type = endnote ,
7111 gender = { f , m } ,
7112 Name-sg = Eindnoot ,
7113 name-sg = eindnoot ,
7114 Name-pl = Eindnoten ,
7115 name-pl = eindnoten ,
7116
7117 type = note ,
7118 gender = f ,
7119 Name-sg = Opmerking ,
7120 name-sg = opmerking ,
7121 Name-pl = Opmerkingen ,
7122 name-pl = opmerkingen ,
7123
7124 type = equation ,
7125 gender = f ,
7126 Name-sg = Vergelking ,
7127 name-sg = vergelking ,
7128 Name-pl = Vergelkingen ,
7129 name-pl = vergelkingen ,
7130 Name-sg-ab = Vgl. ,
7131 name-sg-ab = vgl. ,
7132 Name-pl-ab = Vgl.'s ,
7133 name-pl-ab = vgl.'s ,
7134 refbounds-first-sg = { ,(,) } ,
7135 refbounds = { (,,) } ,
7136
7137 type = theorem ,
7138 gender = f ,
7139 Name-sg = Stelling ,
7140 name-sg = stelling ,
7141 Name-pl = Stellingen ,
7142 name-pl = stellingen ,
7143

```

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”.

```

7144 type = lemma ,
7145 gender = n ,
7146 Name-sg = Lemma ,
7147 name-sg = lemma ,
7148 Name-pl = Lemma's ,
7149 name-pl = lemma's ,
7150
7151 type = corollary ,
7152 gender = n ,
7153 Name-sg = Gevolg ,
7154 name-sg = gevolg ,

```



```

7155 Name-pl = Gevolgen ,
7156 name-pl = gevolgen ,
7157
7158 type = proposition ,
7159 gender = f ,
7160 Name-sg = Propositie ,
7161 name-sg = propositie ,
7162 Name-pl = Proposities ,
7163 name-pl = proposities ,
7164
7165 type = definition ,
7166 gender = f ,
7167 Name-sg = Definitie ,
7168 name-sg = definitie ,
7169 Name-pl = Definities ,
7170 name-pl = definities ,
7171
7172 type = proof ,
7173 gender = n ,
7174 Name-sg = Bews ,
7175 name-sg = bews ,
7176 Name-pl = Bewzen ,
7177 name-pl = bewzen ,
7178
7179 type = result ,
7180 gender = n ,
7181 Name-sg = Resultaat ,
7182 name-sg = resultaat ,
7183 Name-pl = Resultaten ,
7184 name-pl = resultaten ,
7185
7186 type = remark ,
7187 gender = f ,
7188 Name-sg = Opmerking ,
7189 name-sg = opmerking ,
7190 Name-pl = Opmerkingen ,
7191 name-pl = opmerkingen ,
7192
7193 type = example ,
7194 gender = n ,
7195 Name-sg = Voorbeeld ,
7196 name-sg = voorbeeld ,
7197 Name-pl = Voorbeelden ,
7198 name-pl = voorbeelden ,
7199

```

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".

```

7200 type = algorithm ,
7201 gender = { n , f , m } ,
7202 Name-sg = Algoritme ,
7203 name-sg = algoritme ,
7204 Name-pl = Algoritmen ,
7205 name-pl = algoritmen ,

```

7206

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

```
7207 type = listing ,
7208   gender = m ,
7209   Name-sg = Listing ,
7210   name-sg = listing ,
7211   Name-pl = Listings ,
7212   name-pl = listings ,
7213
7214 type = exercise ,
7215   gender = { f , m } ,
7216   Name-sg = Opgave ,
7217   name-sg = opgave ,
7218   Name-pl = Opgaven ,
7219   name-pl = opgaven ,
7220
7221 type = solution ,
7222   gender = f ,
7223   Name-sg = Oplossing ,
7224   name-sg = oplossing ,
7225   Name-pl = Oplossingen ,
7226   name-pl = oplossingen ,
7227 </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->

```
7228 <*package>
7229 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7230 </package>
7231 <*lang-italian>
7232 namesep = {\nobreakspace} ,
7233 pairsep = {\nobreakspace} ,
7234 listsep = {,~} ,
7235 lastsep = {\nobreakspace} ,
7236 tpairsep = {\nobreakspace} ,
7237 tlistsep = {,~} ,
7238 tlastsep = {,~\nobreakspace} ,
7239 notesep = {~} ,
7240 rangesep = {\nobreakspace} ,
7241 +refbounds-rb = {da\nobreakspace,,} ,
7242
7243 type = book ,
7244   gender = m ,
7245   Name-sg = Libro ,
7246   name-sg = libro ,
7247   Name-pl = Libri ,
7248   name-pl = libri ,
```

```

7249
7250 type = part ,
7251     gender = f ,
7252     Name-sg = Parte ,
7253     name-sg = parte ,
7254     Name-pl = Parti ,
7255     name-pl = parti ,
7256
7257 type = chapter ,
7258     gender = m ,
7259     Name-sg = Capitolo ,
7260     name-sg = capitolo ,
7261     Name-pl = Capitoli ,
7262     name-pl = capitoli ,
7263
7264 type = section ,
7265     gender = m ,
7266     Name-sg = Paragrafo ,
7267     name-sg = paragrafo ,
7268     Name-pl = Paragrafi ,
7269     name-pl = paragrafi ,
7270
7271 type = paragraph ,
7272     gender = m ,
7273     Name-sg = Capoverso ,
7274     name-sg = capoverso ,
7275     Name-pl = Capoversi ,
7276     name-pl = capoversi ,
7277
7278 type = appendix ,
7279     gender = f ,
7280     Name-sg = Appendice ,
7281     name-sg = appendice ,
7282     Name-pl = Appendici ,
7283     name-pl = appendici ,
7284
7285 type = page ,
7286     gender = f ,
7287     Name-sg = Pagina ,
7288     name-sg = pagina ,
7289     Name-pl = Pagine ,
7290     name-pl = pagine ,
7291     Name-sg-ab = Pag. ,
7292     name-sg-ab = pag. ,
7293     Name-pl-ab = Pag. ,
7294     name-pl-ab = pag. ,
7295     rangesep = {\textendash} ,
7296     rangetopair = false ,
7297     +refbounds-rb = {,,} ,
7298
7299 type = line ,
7300     gender = f ,
7301     Name-sg = Riga ,
7302     name-sg = riga ,

```

```

7303 Name-pl = Righe ,
7304 name-pl = righe ,
7305
7306 type = figure ,
7307 gender = f ,
7308 Name-sg = Figura ,
7309 name-sg = figura ,
7310 Name-pl = Figure ,
7311 name-pl = figure ,
7312 Name-sg-ab = Fig. ,
7313 name-sg-ab = fig. ,
7314 Name-pl-ab = Fig. ,
7315 name-pl-ab = fig. ,
7316
7317 type = table ,
7318 gender = f ,
7319 Name-sg = Tabella ,
7320 name-sg = tabella ,
7321 Name-pl = Tabelle ,
7322 name-pl = tabelle ,
7323 Name-sg-ab = Tab. ,
7324 name-sg-ab = tab. ,
7325 Name-pl-ab = Tab. ,
7326 name-pl-ab = tab. ,
7327
7328 type = item ,
7329 gender = m ,
7330 Name-sg = Punto ,
7331 name-sg = punto ,
7332 Name-pl = Punti ,
7333 name-pl = punti ,
7334
7335 type = footnote ,
7336 gender = f ,
7337 Name-sg = Nota ,
7338 name-sg = nota ,
7339 Name-pl = Note ,
7340 name-pl = note ,
7341
7342 type = endnote ,
7343 gender = f ,
7344 Name-sg = Nota ,
7345 name-sg = nota ,
7346 Name-pl = Note ,
7347 name-pl = note ,
7348
7349 type = note ,
7350 gender = f ,
7351 Name-sg = Nota ,
7352 name-sg = nota ,
7353 Name-pl = Note ,
7354 name-pl = note ,
7355
7356 type = equation ,

```

```

7357 gender = f ,
7358 Name-sg = Equazione ,
7359 name-sg = equazione ,
7360 Name-pl = Equazioni ,
7361 name-pl = equazioni ,
7362 Name-sg-ab = Eq. ,
7363 name-sg-ab = eq. ,
7364 Name-pl-ab = Eq. ,
7365 name-pl-ab = eq. ,
7366 +refbounds-rb = {da\nobreakspace(,,)} ,
7367 refbounds-first-sg = {,(,)}, ,
7368 refbounds = {(,,)} ,
7369
7370 type = theorem ,
7371 gender = m ,
7372 Name-sg = Teorema ,
7373 name-sg = teorema ,
7374 Name-pl = Teoremi ,
7375 name-pl = teoremi ,
7376
7377 type = lemma ,
7378 gender = m ,
7379 Name-sg = Lemma ,
7380 name-sg = lemma ,
7381 Name-pl = Lemmi ,
7382 name-pl = lemmi ,
7383
7384 type = corollary ,
7385 gender = m ,
7386 Name-sg = Corollario ,
7387 name-sg = corollario ,
7388 Name-pl = Corollari ,
7389 name-pl = corollari ,
7390
7391 type = proposition ,
7392 gender = f ,
7393 Name-sg = Proposizione ,
7394 name-sg = proposizione ,
7395 Name-pl = Proposizioni ,
7396 name-pl = proposizioni ,
7397
7398 type = definition ,
7399 gender = f ,
7400 Name-sg = Definizione ,
7401 name-sg = definizione ,
7402 Name-pl = Definizioni ,
7403 name-pl = definizioni ,
7404
7405 type = proof ,
7406 gender = f ,
7407 Name-sg = Dimostrazione ,
7408 name-sg = dimostrazione ,
7409 Name-pl = Dimostrazioni ,
7410 name-pl = dimostrazioni ,

```

```

7411
7412 type = result ,
7413     gender = m ,
7414     Name-sg = Risultato ,
7415     name-sg = risultato ,
7416     Name-pl = Risultati ,
7417     name-pl = risultati ,
7418
7419 type = remark ,
7420     gender = f ,
7421     Name-sg = Osservazione ,
7422     name-sg = osservazione ,
7423     Name-pl = Osservazioni ,
7424     name-pl = osservazioni ,
7425
7426 type = example ,
7427     gender = m ,
7428     Name-sg = Esempio ,
7429     name-sg = esempio ,
7430     Name-pl = Esempi ,
7431     name-pl = esempi ,
7432
7433 type = algorithm ,
7434     gender = m ,
7435     Name-sg = Algoritmo ,
7436     name-sg = algoritmo ,
7437     Name-pl = Algoritmi ,
7438     name-pl = algoritmi ,
7439
7440 type = listing ,
7441     gender = m ,
7442     Name-sg = Listato ,
7443     name-sg = listato ,
7444     Name-pl = Listati ,
7445     name-pl = listati ,
7446
7447 type = exercise ,
7448     gender = m ,
7449     Name-sg = Esercizio ,
7450     name-sg = esercizio ,
7451     Name-pl = Esercizi ,
7452     name-pl = esercizi ,
7453
7454 type = solution ,
7455     gender = f ,
7456     Name-sg = Soluzione ,
7457     name-sg = soluzione ,
7458     Name-pl = Soluzioni ,
7459     name-pl = soluzioni ,
7460 </lang-italian>

```

Index

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

A	
\A	1883
\AddToHook	107, 2029, 2073, 2096, 2127, 2129, 2167, 2240, 2282, 2433, 2446, 2454, 5313, 5349, 5366, 5368, 5373, 5418, 5420, 5422, 5424, 5426, 5428, 5430, 5432, 5436, 5440, 5442, 5447, 5457, 5473, 5484, 5518, 5569, 5594
\alph	5499, 5585
\appendix	<i>2</i> , <i>126</i> , <i>127</i> , <i>129</i> , <i>138</i>
\appendixname	<i>126</i> , <i>140</i>
\AtEndOfPackage	2444
B	
\babelname	2083
\babelprovide	<i>30</i> , <i>55</i>
\bicaption	<i>129</i>
\bionenumcaption	<i>129</i>
\bitwonumcaption	<i>129</i>
bool commands:	
\bool_case_true:	<i>2</i>
\bool_gset_false:N	539
\bool_gset_true:N	532, 2420
\bool_if:NTF	383, 460, 498, 556, 1759, 1810, 2033, 2037, 2456, 3430, 3833, 3974, 4101, 4137, 4171, 4238, 4251, 4263, 4314, 4331, 4341, 4346, 4392, 4396, 4452, 4477, 4484, 4490, 4501, 4507, 4535, 4580, 4602, 4631, 4780, 4933, 4935, 5521, 5597
\bool_if:nTF	76, 3544, 3554, 3578, 3595, 3610, 3675, 3683, 4324, 4701, 4742, 4823, 4840
\bool_if_exist:NTF	337, 347, 371, 381, 431, 448, 458, 483, 486, 494, 496, 510, 513, 520, 523, 530, 537
\bool_lazy_all:nTF	4420, 4945
\bool_lazy_and:nnTF	49, 54, 2577, 3401, 3422, 4205, 4276, 4654, 4918, 4960
\bool_lazy_any:nTF	5097, 5106
\bool_lazy_or:nnTF	1376, 1409, 1959, 2526, 2790, 3289, 3314, 3405, 4906
\bool_new:N	338, 348, 373, 432, 450, 488, 511, 514, 521, 524, 531, 538, 798, 1573, 1574, 1601, 1625, 1977, 1984, 1991, 2004, 2005, 2175, 2176, 2177, 2178, 2179, 2275, 2276, 2413, 3438, 3453, 3715, 3716, 3727, 3728, 3735, 3737, 3738, 3751, 3752, 3753, 3765, 3766, 5483, 5528, 5568
\bool_set:Nn	3398
\bool_set_eq:NN	546
\bool_set_false:N	372, 449, 485, 487, 522, 1586, 1590, 1762, 1860, 1905, 1947, 2012, 2021, 2022, 2039, 2046, 2187, 2191, 2198, 2206, 2207, 2208, 2300, 2312, 3443, 3536, 3782, 3783, 3800, 3839, 3850, 4250, 4442, 4443, 4450, 4451, 4684, 5104, 5121
\bool_set_true:N	339, 349, 433, 512, 515, 525, 1580, 1581, 1585, 1591, 1784, 1806, 1869, 1871, 1909, 1911, 1920, 1922, 1951, 1953, 1966, 1968, 2011, 2016, 2017, 2185, 2192, 2197, 2214, 2216, 2218, 2221, 2222, 2223, 2288, 2293, 3550, 3560, 3564, 3586, 3601, 3616, 3640, 3808, 3834, 3840, 3844, 3851, 3997, 4008, 4019, 4069, 4105, 4141, 4175, 4192, 4273, 4307, 4479, 4539, 4585, 4607, 4636, 4894, 4901, 5023, 5082, 5120, 5157, 5164, 5165, 5183, 5200, 5202, 5497, 5535, 5582
\bool_until_do:Nn	1861, 1912, 1954, 3576, 3801
\bool_while_do:nn	5629
\l_tmpa_bool	1762, 1784, 1806, 1810, 1860, 1861, 1869, 1871, 1905, 1909, 1911, 1912, 1920, 1922, 1947, 1951, 1953, 1954, 1966, 1968
C	
\chaptername	140
clist commands:	
\clist_map_inline:nn	631, 679, 696, 1015, 2209, 2361, 2422, 2926, 5500, 5587
\contsubbottom	129
\contsubcaption	129
\counterwithin	5
\crefstripprefix	45
cs commands:	
\cs_generate_variant:Nn	73, 280, 286, 293, 304, 315, 326, 341, 351, 358, 365, 376, 400, 410, 435, 442, 453, 491, 517, 527, 534, 541,

969, 1843, 1879, 1930, 1976, 2585, 4736, 4774, 5140, 5244, 5277, 5310	4798, 4800, 4804, 4815, 4816, 4818, 4826, 4830, 4831, 4834, 4861, 4876
<code>\cs_if_exist:NTF</code> 24, 27, 66, 86, 5337, 5343	<code>\ExplSyntaxOn</code> 30, 929
<code>\cs_if_exist_p:N</code> 51, 56, 4207, 4278, 4656, 5631	F
<code>\cs_if_exist_use:N</code> 5435, 5669	<code>\figurename</code> 140
<code>\cs_new:Npn</code> 64, 74, 84, 95, 281, 287, 289, 291, 294, 305, 316, 328, 330, 718, 4697, 4737, 4775, 5130	file commands:
<code>\cs_new_eq:NN</code> 5475, 5480	<code>\file_get:nnNTF</code> 923
<code>\cs_new_protected:Npn</code> 275, 332, 342, 352, 359, 366, 391, 401, 422, 424, 426, 436, 443, 481, 508, 518, 528, 535, 811, 913, 1522, 1541, 1745, 1844, 1889, 1931, 2452, 2583, 3393, 3457, 3499, 3510, 3523, 3653, 3705, 3767, 3981, 4446, 4693, 4695, 4889, 5124, 5141, 5212, 5245, 5278, 5408, 5536	<code>\fmtversion</code> 3
<code>\cs_set:Npn</code> 5411	<code>\footnote</code> 2, 129, 130
<code>\cs_set_eq:NN</code> 720, 5410, 5416, 5476, 5481	G
<code>\cs_set_nopar:Npn</code> 5466	group commands:
<code>\cs_to_str:N</code> 331	<code>\group_begin:</code> 109, 734, 915, 1761, 1848, 1893, 1935, 2850, 3395, 3409, 3441, 4348, 4364, 4713, 4725, 4754, 4765, 4796, 4803, 4814, 4829, 4860, 4875, 5423, 5427, 5431
D	<code>\group_end:</code> 112, 746, 967, 1825, 1874, 1925, 1971, 2878, 3412, 3435, 3445, 4361, 4367, 4717, 4729, 4757, 4768, 4799, 4808, 4817, 4832, 4865, 4880, 5425, 5429, 5433
<code>\d</code> 1883	H
E	<code>\hyperlink</code> 5127
<code>\endinput</code> 12	I
<code>\endsidecaption</code> 128, 129	<code>\IfBooleanT</code> 3442
exp commands:	<code>\IfClassLoadedTF</code> 119
<code>\exp_args:NNe</code> 36, 39	<code>\ifdraft</code> 2190
<code>\exp_args:NNNo</code> 277	<code>\IfFormatAtLeastTF</code> 3, 4
<code>\exp_args:NNNV</code> 1824, 1873, 1924, 1970	<code>\ifoptionfinal</code> 2196
<code>\exp_args:NNo</code> 277, 283	<code>\IfPackageAtLeastTF</code> 2286
<code>\exp_args:No</code> 283	<code>\IfPackageLoadedTF</code> 117
<code>\exp_args:Nx</code> 923, 5541, 5549	<code>\input</code> 30
<code>\exp_args:Nxx</code> 1763, 1775, 1787, 1797, 1863, 1914, 1956, 5143, 5149, 5171, 5175, 5190	int commands:
<code>\exp_not:N</code> 113, 4348, 4361, 4364, 4367, 4713, 4717, 4725, 4729, 4754, 4757, 4765, 4768, 4796, 4799, 4803, 4808, 4814, 4817, 4829, 4832, 4860, 4865, 4875, 4880	<code>\int_case:nnTF</code> 3984, 4026, 4088, 4399, 4514, 4571
<code>\exp_not:n</code> 284, 4013, 4036, 4044, 4062, 4075, 4079, 4114, 4123, 4145, 4153, 4160, 4184, 4198, 4215, 4225, 4267, 4290, 4300, 4349, 4360, 4365, 4366, 4523, 4546, 4558, 4592, 4614, 4623, 4643, 4664, 4674, 4714, 4726, 4755, 4756, 4766, 4767, 4797,	<code>\int_compare:nNnTF</code> 3587, 3602, 3617, 3629, 3641, 3661, 3663, 3707, 3881, 4004, 4032, 4054, 4109, 4179, 4384, 4386, 4463, 4493, 4552, 5153, 5159, 5179, 5185, 5647
	<code>\int_compare_p:nNn</code> 1379, 1412, 1960, 1961, 2528, 2792, 3292, 3317, 3677, 3685, 4424, 4910, 4921, 4950, 5117
	<code>\int_incr:N</code> 4439, 4471, 4483, 4485, 4500, 4502, 4506, 4508, 4520, 4543, 4555, 4589, 4611, 4620, 4640, 4691, 5645
	<code>\int_new:N</code> 3454, 3455, 3717, 3718, 3719, 3732, 3733
	<code>\int_rand:n</code> 302, 313, 324

<code>\int_set:Nn</code>	110, 3662, 3664, 3668, 3671, 5628	<code>\msg_line_context:</code>	122, 128, 132, 134, 137, 143, 149, 155, 161, 166, 171, 176, 181, 187, 192, 195, 198, 203, 207, 214, 219, 224, 231, 240, 245, 250, 254, 256, 258, 260, 267
<code>\int_to_roman:n</code>	5632, 5639, 5640, 5643	<code>\msg_new:nnn</code>	120, 126, 131, 133, 135, 141, 147, 153, 159, 164, 169, 174, 179, 184, 189, 194, 196, 201, 206, 208, 210, 212, 217, 222, 228, 230, 232, 237, 243, 248, 253, 255, 257, 259, 261, 263, 265, 270
<code>\int_use:N</code>	52, 57, 62, 68	<code>\msg_warning:nn</code>	2038, 2044, 2316, 2580, 4426
<code>\int_zero:N</code>	3655, 3656, 3777, 3778, 3779, 3780, 3781, 4437, 4438, 4440, 4441, 4686, 4687	<code>\msg_warning:nnn</code>	738, 759, 1554, 1561, 1717, 1723, 2116, 2153, 2165, 2227, 2238, 2280, 2305, 2389, 2439, 2449, 2599, 2713, 2719, 2877, 2921, 2967, 3159, 3165, 3246, 3865, 4245, 4939, 4955
<code>\l_tmpa_int</code>	5628, 5632, 5639, 5640, 5643, 5645, 5647	<code>\msg_warning:nnnn</code>	827, 844, 878, 896, 2069, 2256, 2265, 2334, 2487, 2493, 2499, 2505, 2535, 2748, 2754, 2760, 2766, 2802, 2894, 2901, 2931, 3214, 3220, 3226, 3232, 3305, 3330, 3873, 5024, 5084
iow commands:		<code>\msg_warning:nnnnn</code>	864, 903, 2915, 4974
<code>\iow_char:N</code>	123, 138, 139, 144, 145, 150, 151, 156, 157, 209, 226, 273, 2257, 2266	<code>\msg_warning:nnnnnn</code>	4981
<code>\iow_newline:</code>	267		
J			
<code>\jobname</code>	5457		
K			
keys commands:			
<code>\l_keys_choice_tl</code>	803		
<code>\keys_define:nn</code>	20, 639, 685, 702, 763, 970, 1059, 1084, 1112, 1321, 1360, 1438, 1548, 1575, 1602, 1611, 1626, 1635, 1978, 1985, 1992, 1998, 2006, 2041, 2050, 2064, 2092, 2131, 2162, 2169, 2181, 2242, 2249, 2251, 2260, 2270, 2277, 2289, 2301, 2313, 2321, 2328, 2357, 2383, 2407, 2415, 2435, 2464, 2482, 2512, 2546, 2569, 2595, 2607, 2631, 2743, 2773, 2813, 2881, 2952, 2976, 3003, 3209, 3239, 3279, 3341		
<code>\keys_set:nn</code>	27, 30, 57, 58, 83, 743, 891, 954, 2294, 2584, 2589, 2875, 3396		
keyval commands:			
<code>\keyval_parse:nnn</code> ...	1527, 2332, 2387		
L			
<code>\label</code> ..	60, 128, 129, 131, 134, 5410, 5416		
<code>\labelformat</code>	3		
<code>\languagename</code>	24, 140, 2077		
M			
<code>\mainbabelname</code>	24, 2084		
<code>\MessageBreak</code>	10		
MH commands:			
<code>\MH_if_boolean:nTF</code>	5533		
msg commands:			
<code>\msg_info:nnn</code>	957, 1010, 1075, 1268, 1274, 1328, 5387, 5459, 5525, 5560, 5601, 5621, 5648	<code>\newcounter</code>	5, 5333, 5334
<code>\msg_info:nnnn</code>	983, 990, 1020, 1392, 1426	<code>\NewDocumentCommand</code>	732, 749, 2581, 2586, 2848, 3391, 3439
<code>\msg_info:nnnnn</code>	1004	<code>\newfloat</code>	129
		<code>\NewHook</code>	1634
		<code>\newsubfloat</code>	129
		<code>\newtheorem</code>	139
		<code>\nobreakspace</code>	1535, 5705, 5706, 5708, 5709, 5711, 5713, 5910, 5911, 5913, 5914, 5916, 5918, 6357, 6358, 6360, 6361, 6363, 6365, 6574, 6575, 6577, 6578, 6580, 6582, 6800, 6801, 6803, 6804, 6806, 6808, 7014, 7015, 7017, 7018, 7020, 7022, 7232, 7233, 7235, 7236, 7238, 7240, 7241, 7366
		<code>\NumCheckSetup</code>	45
		<code>\NumsCheckSetup</code>	45
P			
<code>\PackageError</code>	7		
<code>\pagename</code>	140		
<code>\pagenote</code>	129, 130		
<code>\pagenumbering</code>	7		
<code>\pageref</code>	85		
<code>\PagesCheckSetup</code>	45		

<code>\paragraph</code>	61	<code>\seq_gset_from_clist:Nn</code>	
<code>\part</code>	138 567, 576, 588, 603, 611, 772, 787	
<code>\partname</code>	140	<code>\seq_gset_split:Nnn</code>	425
prg commands:		<code>\seq_if_empty:NTF</code> ..	823, 860, 941, 981, 1002, 2862, 2892, 2913, 3805, 4972
<code>\prg_generate_conditional_</code>		<code>\seq_if_exist:NTF</code> ..	428, 438, 445, 456
variant:Nnn	420, 468, 479, 506, 551, 559, 728, 1887	<code>\seq_if_in:NnTF</code>	
<code>\prg_new_conditional:Npnn</code> 841, 875, 919, 987, 1017, 2363, 2424, 2457, 2898, 2928, 3503, 4968	
.....	116, 118, 377, 454, 492, 553, 722	<code>\seq_item:Nn</code> 4707, 4712, 4718, 4720, 4723, 4724, 4730, 4731, 4748, 4753, 4758, 4760, 4763, 4764, 4769, 4770, 4801, 4802, 4809, 4811, 4846, 4858, 4866, 4869, 4873, 4874, 4881, 4882	
<code>\prg_new_protected_conditional:Npnn</code>	411, 470, 542, 1880	
<code>\prg_return_false:</code> 117, 119, 385, 389, 418, 462, 466, 477, 500, 504, 549, 556, 557, 726, 1885	
<code>\prg_return_true:</code> 117, 119, 384, 387, 416, 461, 464, 475, 499, 502, 547, 556, 725, 1884	
<code>\prg_set_eq_conditional:NNn</code> ...	730	<code>\seq_map_break:n</code>	98, 3696, 3699
prop commands:		<code>\seq_map_function:NN</code>	3462
<code>\prop_if_in:NnTF</code>	36	<code>\seq_map_indexed_inline:Nn</code> .	44, 3657
<code>\prop_if_in_p:Nn</code>	77	<code>\seq_map_inline:Nn</code> ...	1056, 1081, 1318, 1357, 1435, 2448, 2461, 2509, 2543, 2592, 2604, 2770, 2810, 2949, 2973, 3236, 3276, 3338, 3693, 5539
<code>\prop_item:Nn</code>	39, 78	<code>\seq_map_tokens:Nn</code>	80
<code>\prop_new:N</code>	2327, 2382	<code>\seq_new:N</code>	429, 439, 564, 565, 566, 575, 587, 602, 610, 623, 627, 767, 782, 912, 1034, 1610, 2356, 2414, 3437, 3456, 3714, 3731, 3754, 3755, 3756, 3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764
<code>\prop_put:Nnn</code>	1545	<code>\seq_pop_left:NN</code>	3803
<code>\prop_remove:Nn</code>	1544	<code>\seq_put_right:Nn</code>	
<code>\providecommand</code>	3 1018, 2365, 2929, 3506	
<code>\ProvidesExplPackage</code>	14	<code>\seq_reverse:N</code>	1616
<code>\ProvidesFile</code>	30	<code>\seq_set_eq:NN</code>	
		. 430, 474, 3769, 3995, 4006, 4017, 4067, 4103, 4139, 4173, 4189, 4271, 4305, 4316, 4537, 4582, 4604, 4633	
		<code>\seq_set_from_clist:Nn</code> ...	1615, 3397
		<code>\seq_set_split:Nnn</code>	423
		<code>\seq_sort:Nn</code>	87, 3465
		<code>\seq_use:Nn</code>	4986
		<code>\g_tmpa_seq</code>	1373, 1375, 1380, 1389, 1394, 1406, 1408, 1413, 1423, 1428, 3286, 3288, 3293, 3302, 3307, 3311, 3313, 3318, 3327, 3332
		<code>\l_tmpa_seq</code> 1014, 1018, 1050, 2523, 2525, 2528, 2532, 2537, 2787, 2789, 2792, 2799, 2804, 2925, 2929, 2944	
		<code>\setcounter</code> ..	5335, 5336, 5352, 5367, 5371
		<code>\sidefootnote</code>	129, 130
		sort commands:	
		<code>\sort_return_same:</code>	87, 91, 3472, 3477, 3551, 3571, 3592, 3607, 3621, 3646, 3681, 3696, 3712

R

<code>\refstepcounter</code> 3, 4, 128, 129, 132, 134, 136	
regex commands:	
<code>\regex_match:nnTF</code>	1883
<code>\renewlist</code>	136
<code>\RequirePackage</code>	16, 17, 18, 19, 2034

S

<code>\scantokens</code>	127
seq commands:	
<code>\seq_clear:N</code>	447, 822, 859, 940, 953, 1014, 1622, 2523, 2787, 2861, 2874, 2925, 3459, 5271
<code>\seq_const_from_clist:Nn</code>	21
<code>\seq_count:N</code>	
.....	1380, 1394, 1413, 1428, 2528, 2537, 2792, 2804, 3293, 3307, 3318, 3332
<code>\seq_gclear:N</code> .	1373, 1406, 3286, 3311
<code>\seq_gconcat:NNN</code>	624, 628
<code>\seq_get_left:NN</code>	
.....	837, 848, 944, 992, 2865, 2903, 3811
<code>\seq_gput_right:Nn</code> ...	955, 961, 2426
<code>\seq_gremove_all:Nn</code>	2458
<code>\seq_gset_eq:NN</code>	440, 1042

\sort_return_swapped:	5618
.. 87, 91, 3485, 3561, 3570, 3591, 3606, 3622, 3645, 3689, 3699, 3711	
\stepcounter	134, 5351, 5370
str commands:	
\str_case:nn	45
\str_case:nnTF	1117, 1639, 2098, 2135, 2211, 2635, 3008
\str_compare:nNnTF	3567
\str_if_eq:nnTF	97, 4739
\str_if_eq_p:nn	5102, 5108, 5110, 5114
\str_new:N	2049
\str_set:Nn ...	2054, 2056, 2058, 2060
\string	5546, 5554
\subbottom	129
\subcaption	129
\subcaptionref	129
\subparagraph	138
\subref	137
\subsection	138
\subsubsection	61
\subsubsections	138
\subsubsubsection	61
\subtop	129
T	
\tablename	140
\tag	123, 132, 134
TeX and L ^A T _E X commands:	
\@sidecaption	128
\@Alph	126
\@addtoreset	5
\@auxout	5545, 5553
\@bsphack	916, 5538
\@capttype	5435, 5669
\@chapapp	126
\@currentcounter	2–5, 63, 129, 132, 136, 27, 28, 55, 56, 57, 2410, 2411
\@currentlabel	3, 123, 129, 136
\@elt	5
\@esphack	966, 5558
\@ifl@t@r	3
\@mem@scap@afterhook	129
\@memsubcaption	129
\@onlypreamble	748, 762, 2880
\bbl@loaded	55
\bbl@main@language	24, 2078
\c@lstnumber	136
\c@page	7, 110
\caption@subtyphook	5670
\hyper@link	113, 121
\hyper@linkfile	5128
\lst@AddToHook	5617, 5619
\lst@Init	136
\lst@label	5618
\lst@MakeCaption	136
\ltx@gobble	131
\ltx@label	131, 5475, 5476, 5480, 5481
\m@mscaplabel	128
\MT@newlabel	5546, 5554
\protected@write	5545, 5553
\zref@addprop	21, 31, 46, 61, 63, 105, 115
\zref@default	113, 4694, 4696
\zref@extractdefault	11, 122, 278, 284, 288
\zref@ifpropundefined	43, 1272, 1559, 1721, 2717, 3163, 5132, 5584
\zref@ifrefcontainsprop	43, 45, 1749, 1757, 1816, 1834, 1846, 1891, 1933, 3868, 4699, 4782, 4836, 5135
\zref@ifrefundefined	3467, 3469, 3481, 3836, 3838, 3843, 3860, 4242, 4253, 4454, 4777, 4891
\zref@label	131, 5469
\zref@localaddprop	5437, 5522, 5598, 5671
\ZREF@mainlist	21, 31, 46, 61, 63, 105, 115, 5437, 5522, 5598, 5671
\zref@newprop	6, 7, 20, 22, 32, 47, 62, 100, 114, 5434, 5499, 5585, 5668
\zref@refused	3858
\zref@wrapper@babel	83, 131, 3392, 5469
\textendash	1539, 5760, 6024, 6636, 6858, 7072, 7295
\thechapter	126
\thelstnumber	136
\thepage	7, 111
\thesection	126
tl commands:	
\c_novalue_tl	687, 688, 689, 690, 691, 692, 693, 2466, 2514, 2609, 2775
\tl_clear:N	346, 370, 831, 869, 882, 899, 908, 933, 942, 975, 2590, 2853, 2863, 2886, 3771, 3772, 3773, 3774, 3775, 3776, 3807, 4432, 4433, 4434, 4435, 4436, 4482, 4499, 4893, 4900, 4930, 5022, 5081, 5238
\tl_const:Nn	1524
\tl_gclear:N	363, 407
\tl_gset:Nn	111, 356, 397, 741, 756
\tl_head:N	3605, 3618, 3630, 3632, 3642, 3644
\tl_head:n	1864, 1915, 1957, 1961
\tl_if_empty:NTF	34, 88, 825, 835, 862, 873, 894, 901, 1008, 1064, 1089, 1121, 1156, 1193, 1230, 1278, 1326, 1332, 1365, 1443, 1481, 1747, 1868, 1919, 2919, 2957, 2981, 3012,

3047, 3084, 3121, 3169, 3244, 3250, 3284, 3346, 3367, 3413, 4240, 4833, 4915, 4937, 4995, 5006, 5049, 5618	4288, 4298, 4344, 4467, 4469, 4480, 4497, 4912, 4913, 4926, 5439, 5441
\tl_if_empty:nTF	\tl_set_eq:NN 415, 4430
.. 735, 751, 974, 1266, 1543, 1552, 1715, 2419, 2711, 2885, 3157, 5126	\tl_show:N 4393
\tl_if_empty_p:N	\tl_tail:N 3635, 3637
..... 50, 55, 2579, 4206, 4277, 4655, 4948, 4962, 5101, 5111, 5115	\tl_tail:n
\tl_if_empty_p:n 1377, 1410, 2527, 2791, 3290, 3315, 3546, 3547, 3556, 3557, 3582, 3583, 3598, 3613	.. 1866, 1867, 1917, 1918, 1963, 1964
\tl_if_eq:NNTF 3517, 3540, 3847	\tl_use:N 299, 310, 321, 757, 921, 926, 956, 958, 962
\tl_if_eq:NnTF	\l_tmpa_tl 930, 954, 1850, 1864, 1866, 1895, 1906, 1915, 1917, 1937, 1948, 1957, 1963, 3418, 3419
.. 1773, 3460, 3492, 3667, 3670, 3695, 3698, 3815, 3863, 4897, 5147	\l_tmpb_tl . 1812, 1819, 1822, 1826, 1855, 1864, 1867, 1868, 1875, 1900, 1908, 1915, 1918, 1919, 1926, 1942, 1950, 1957, 1960, 1961, 1964, 1972
\tl_if_eq:nnTF .. 1763, 1775, 1787, 1797, 1863, 1914, 1956, 3659, 5143, 5149, 5171, 5175, 5190, 5541, 5549	U
\tl_if_exist:NNTF 334, 344, 354, 361, 368, 379, 395, 405, 724	use commands:
\tl_if_exist_p:N 2578	\use:N 25, 28, 803, 4209, 4284, 4658, 5304
\tl_if_novalue:nTF	\UseHook 1849, 1894, 1936
..... 2469, 2517, 2612, 2778	V
\tl_map_break:n 98	\value 5352, 5371
\tl_map_tokens:Nn 90	\verbfootnote 129, 130
\tl_new:N 106, 335, 345, 355, 362, 396, 406, 561, 562, 563, 713, 714, 715, 716, 740, 755, 1547, 2161, 2180, 2248, 2269, 2320, 2406, 3447, 3448, 3449, 3450, 3451, 3452, 3720, 3721, 3722, 3723, 3724, 3725, 3726, 3729, 3730, 3734, 3736, 3739, 3740, 3741, 3742, 3743, 3744, 3745, 3746, 3747, 3748, 3749, 3750, 5438	Z
\tl_put_left:Nn . 4327, 4334, 4377, 5008, 5009, 5051, 5053, 5055, 5057	\Z 1883
\tl_put_right:Nn . 4011, 4034, 4042, 4060, 4073, 4112, 4121, 4143, 4151, 4158, 4182, 4196, 4213, 4223, 4521, 4544, 4556, 4590, 4612, 4621, 4641, 4662, 4672, 4916, 4917, 4928, 5670	\zcDeclareLanguage . 12, 25, 732, 5695, 5900, 6353, 6568, 6797, 7011, 7229
\tl_reverse:N 3527, 3530	\zcDeclareLanguageAlias
\tl_set:Nn 26, 749, 5696, 5697, 5698, 5699, 5700, 5701, 5702, 5903, 5904, 5905, 5906, 5907, 6354, 6569, 6570, 6571
. 277, 336, 717, 742, 932, 976, 988, 1556, 1563, 1565, 1754, 1822, 1826, 1839, 1850, 1855, 1866, 1867, 1875, 1877, 1895, 1900, 1917, 1918, 1926, 1928, 1937, 1942, 1963, 1964, 1972, 1974, 2077, 2078, 2083, 2084, 2087, 2088, 2102, 2108, 2113, 2139, 2145, 2150, 2588, 2854, 2887, 2899, 3634, 3636, 3817, 3818, 3991, 3993, 4265,	\zcLanguageSetup
	. 20, 21, 29, 30, 67, 72, 73, 140, 2848
	\zcpageref 85, 3439
	\zcref 20, 64, 66, 83, 85–87, 93, 95, 133, 3391, 3444
	\zcRefTypeSetup 20, 21, 67, 2586
	\zcsetup 20, 55, 64, 66, 67, 2581
	\zlabel 128, 129, 131, 132, 134, 135, 5414, 5618
	zrefcheck commands:
	\zrefcheck_zcref_beg_label: . . 3404
	\zrefcheck_zcref_end_label_- maybe: 3426
	\zrefcheck_zcref_run_checks_on_- labels:n 3427
	zrefclever commands:
	\zrefclever_language_if_declared:n 730
	\zrefclever_language_if_declared:nTF 730

<code>\zrefclever_language_varname:n</code> ..	4710, 4715, 4727, 4751, 4792, 4805,
.....	720, 720
<code>\l_zrefclever_ref_language_tl</code> ..	716
zrefclever internal commands:	
<code>\l_zrefclever_abbrev_bool</code>	3739, 3926, 4919
<code>\l_zrefclever_amsmath_subequations_</code> <code>bool</code>	5483, 5497, 5521
<code>\l_zrefclever_breqn_dgroup_bool</code>	5568, 5582, 5597
<code>\l_zrefclever_cap_bool</code>	3739, 3922, 4907
<code>\l_zrefclever_capfirst_bool</code> ...	1984, 1987, 4909
<code>__zrefclever_compat_module:nn</code> 64,
.....	2452, 2452, 5311, 5329, 5390, 5462,
.....	5529, 5564, 5604, 5624, 5651, 5674
<code>__zrefclever_counter_reset_by:n</code>	6, 61, 62, 66, 68, 70, 74, 74, 5491, 5576
<code>__zrefclever_counter_reset_by_</code> <code>aux:nn</code>	81, 84
<code>__zrefclever_counter_reset_by_</code> <code>auxi:nnn</code>	91, 95
<code>\l_zrefclever_counter_resetby_</code> <code>prop</code>	5, 62, 77, 78, 2382, 2394
<code>\l_zrefclever_counter_resetters_</code> <code>seq</code> ..	5, 61, 62, 80, 2356, 2363, 2366
<code>\l_zrefclever_counter_type_prop</code>	4, 60, 36, 39, 2327, 2339
<code>\l_zrefclever_current_counter_</code> <code>tl</code>	3, 5, 63, 20, 24, 25,
.....	37, 40, 42, 50, 51, 52, 103, 2406, 2409
<code>\l_zrefclever_current_language_</code> <code>tl</code>	24,
.....	55, 714, 2077, 2083, 2087, 2103, 2140
<code>\l_zrefclever_endrangefunc_tl</code> 3739, 3914, 4206, 4207, 4209,
.....	4277, 4278, 4284, 4655, 4656, 4658
<code>\l_zrefclever_endrangeprop_tl</code> 48, 1747, 1757, 3739, 3918
<code>\zrefclever_extract:nnn</code> 11, 287, 287, 1852, 1857,
.....	1897, 1902, 1939, 1944, 3588, 3590,
.....	3603, 3620, 3708, 3710, 5154, 5156,
.....	5160, 5162, 5180, 5182, 5186, 5188
<code>__zrefclever_extract_default:Nnnn</code>	11, 275, 275, 280, 1751, 1812,
.....	1819, 1829, 1836, 3501, 3512, 3514,
.....	3525, 3528, 3531, 3533, 3821, 3824
<code>__zrefclever_extract_unexp:nnn</code> 11, 122,
.....	281, 281, 286, 1765, 1769, 1777,
.....	1781, 1789, 1793, 1799, 1803, 4356,
<code>__zrefclever_extract_url_</code> <code>unexp:n</code>	4352, 4709,
.....	4750, 4788, 4850, 5130, 5130, 5140
<code>__zrefclever_get_enclosing_</code> <code>counters_value:n</code> 6, 64, 64, 69, 73, 102
<code>\zrefclever_get_endrange_</code> <code>pagecomp:nnN</code>	1889, 1930
<code>__zrefclever_get_endrange_</code> <code>pagecomptwo:nnN</code>	1931, 1976
<code>__zrefclever_get_endrange_</code> <code>property:nnN</code>	45, 1745, 1843
<code>\zrefclever_get_endrange_</code> <code>stripprefix:nnN</code>	1844, 1879
<code>\zrefclever_get_ref:nN</code>
.....	112, 113, 4014, 4037, 4045, 4063,
.....	4076, 4080, 4115, 4124, 4146, 4154,
.....	4161, 4185, 4199, 4226, 4268, 4301,
.....	4336, 4524, 4547, 4559, 4593, 4615,
.....	4624, 4644, 4675, 4697, 4697, 4736
<code>__zrefclever_get_ref_endrange:nnN</code>	45, 113,
.....	114, 4216, 4291, 4665, 4737, 4737, 4774
<code>\zrefclever_get_ref_first:</code> ...	112, 113, 117, 4328, 4378, 4775, 4775
<code>\zrefclever_get_rf_opt_bool:nN</code> 125	
<code>\zrefclever_get_rf_opt_</code> <code>bool:nnnnN</code>	20,
.....	3919, 3923, 3927, 5278, 5278, 5310
<code>\zrefclever_get_rf_opt_</code> <code>seq:nnnN</code> ..	20, 124, 3931, 3935,
.....	3939, 3943, 3947, 3951, 3955, 3959,
.....	3963, 3967, 4964, 5245, 5245, 5277
<code>__zrefclever_get_rf_opt_tl:nnnN</code>	20,
.....	22, 45, 124, 3415, 3786, 3790, 3794,
.....	3883, 3887, 3891, 3895, 3899, 3903,
.....	3907, 3911, 3915, 5212, 5212, 5244
<code>\zrefclever_hyperlink:nnn</code> 122, 4350,
.....	4708, 4749, 4786, 4848, 5124, 5124
<code>\l_zrefclever_hyperlink_bool</code> ...	2004, 2011, 2016, 2021, 2033, 2039,
.....	2046, 3443, 4703, 4744, 4842, 5099
<code>\l_zrefclever_hyperref_warn_</code> <code>bool</code> ...	2005, 2012, 2017, 2022, 2037
<code>__zrefclever_if_class_loaded:n</code> 116, 118
<code>\zrefclever_if_class_loaded:nTF</code>	5392

```

\__zrefclever_if_package_-
  loaded:n ..... 116, 116
\__zrefclever_if_package_-
  loaded:nTF ..... 2031,
  2075, 2081, 2284, 5331, 5464, 5471,
  5531, 5566, 5606, 5626, 5653, 5676
\__zrefclever_is_integer_rgx:n ..
  ..... 1880, 1881, 1888
\__zrefclever_is_integer_rgx:nTF
  ..... 1906, 1908, 1948, 1950
\l__zrefclever_label_a_tl .....
  . 92, 3720, 3804, 3823, 3836, 3858,
  3860, 3866, 3869, 3875, 3992, 4014,
  4037, 4045, 4080, 4146, 4161, 4211,
  4217, 4226, 4258, 4268, 4286, 4292,
  4301, 4454, 4458, 4468, 4481, 4498,
  4524, 4560, 4624, 4660, 4666, 4675
\l__zrefclever_label_b_tl .....
  ..... 92, 3720,
  3807, 3812, 3826, 3838, 3843, 4458
\l__zrefclever_label_count_int ..
  ..... 92, 3717, 3777,
  3881, 3984, 4437, 4463, 4691, 4951
\l__zrefclever_label_enclval_a_-
  tl ..... 3447, 3525, 3527, 3582,
  3598, 3618, 3630, 3634, 3635, 3642
\l__zrefclever_label_enclval_b_-
  tl ..... 3447, 3528, 3530, 3583,
  3605, 3613, 3632, 3636, 3637, 3644
\l__zrefclever_label_extdoc_a_tl
  .. 3447, 3531, 3541, 3546, 3556, 3569
\l__zrefclever_label_extdoc_b_tl
  .. 3447, 3533, 3542, 3547, 3557, 3568
\l__zrefclever_label_type_a_tl ..
  ..... 3416, 3447, 3502, 3504,
  3507, 3513, 3518, 3667, 3695, 3787,
  3791, 3795, 3817, 3822, 3848, 3863,
  3884, 3888, 3892, 3896, 3900, 3904,
  3908, 3912, 3916, 3920, 3924, 3928,
  3932, 3936, 3940, 3944, 3948, 3952,
  3956, 3960, 3964, 3968, 3994, 4470
\l__zrefclever_label_type_b_tl ..
  ..... 3447, 3515,
  3519, 3670, 3698, 3818, 3825, 3849
\__zrefclever_label_type_put_-
  new_right:n 86, 87, 3463, 3499, 3499
\l__zrefclever_label_types_seq ..
  .... 87, 3456, 3459, 3503, 3506, 3693
\__zrefclever_labels_in_sequence:nn
  .. 48, 93, 122, 4256, 4457, 5141, 5141
\l__zrefclever_lang_decl_case_tl
  561, 942, 945, 988, 993, 1332, 1349,
  2863, 2866, 2899, 2904, 3250, 3267
\l__zrefclever_lang_declension_-
  seq ..... 561,
  821, 822, 823, 837, 841, 848, 939,
  940, 941, 944, 981, 987, 992, 2860,
  2861, 2862, 2865, 2892, 2898, 2903
\l__zrefclever_lang_gender_seq ..
  .... 561, 858, 859, 860, 875, 952,
  953, 1002, 1017, 2873, 2874, 2913, 2928
\__zrefclever_language_if_-
  declared:n ..... 25, 722, 729, 731
\__zrefclever_language_if_-
  declared:n(TF) ..... 25
\__zrefclever_language_if_-
  declared:nTF 296, 307, 318, 722,
  737, 753, 813, 917, 2114, 2151, 2851
\__zrefclever_language_varname:n
  ..... 24, 25, 299, 310,
  321, 718, 718, 721, 724, 740, 741,
  755, 756, 757, 921, 926, 956, 958, 962
\l__zrefclever_last_of_type_bool
  ..... 92, 3714, 3834,
  3839, 3840, 3844, 3850, 3851, 3974
\l__zrefclever_lastsep_tl . 3739,
  3898, 4044, 4079, 4123, 4160, 4198
\l__zrefclever_link_star_bool ...
  .. 3398, 3437, 4704, 4745, 4843, 5100
\l__zrefclever_listsep_tl .....
  ... 3739, 3894, 4075, 4153, 4523,
  4546, 4558, 4592, 4614, 4623, 4643
\g__zrefclever_loaded_langfiles_-
  seq ..... 912, 920, 955, 961
\__zrefclever_ltxlabel:n .....
  ..... 131, 5466, 5476, 5481
\l__zrefclever_main_language_tl .
  ..... 24,
  55, 715, 2078, 2084, 2088, 2109, 2146
\__zrefclever_mathtools_showonlyrefs:n
  ..... 3432, 5536
\l__zrefclever_mathtools_-
  showonlyrefs_bool 3430, 5528, 5535
\__zrefclever_memoir_both_-
  labels: .....
  .. 5408, 5419, 5421, 5423, 5427, 5431
\l__zrefclever_memoir_footnote_-
  type_tl .... 5438, 5439, 5441, 5445
\__zrefclever_memoir_label_and_-
  zlabel:n ..... 5411, 5416
\__zrefclever_memoir_orig_-
  label:n ..... 5410, 5413
\__zrefclever_name_default: ....
  ..... 4693, 4695, 4825
\l__zrefclever_name_format_-
  fallback_tl ..... 3726, 4926,
  4930, 4995, 5044, 5056, 5058, 5076

```

<code>\l_zrefclever_name_format_tl ...</code>	<code>_zrefclever_opt_bool_if:N(TF) .</code>	20
... 3726, 4912, 4913, 4916, 4917,	<code>_zrefclever_opt_bool_if:NTF ...</code>	
4927, 4928, 5001, 5008, 5009, 5017,	553, 886
5025, 5035, 5052, 5053, 5066, 5086	<code>_zrefclever_opt_bool_if_set:N .</code>	
<code>\l_zrefclever_name_in_link_bool</code>	492, 507
.....	<code>_zrefclever_opt_bool_if_-</code>	
115,	<code>set:N(TF)</code>	18
117, 3726, 4346, 4780, 5104, 5120, 5121	<code>_zrefclever_opt_bool_if_-</code>	
<code>\l_zrefclever_namefont_tl</code>	<code>set:NTF</code>	492,
3739,	544, 555, 1445, 1461, 1483, 1499	
3906, 4349, 4365, 4797, 4815, 4830	<code>_zrefclever_opt_bool_set_-</code>	
<code>\l_zrefclever_nameinlink_str ...</code>	<code>false:N</code>	19, 508, 518, 527, 2556, 2827
.....	<code>true:N</code>	19, 508, 508, 517, 2551, 2818
2049, 2054, 2056,	<code>_zrefclever_opt_bool_unset:N .</code>	
2058, 2060, 5102, 5108, 5110, 5114	18, 481, 481, 491, 2561, 2836
<code>\l_zrefclever_namesep_tl</code>	<code>_zrefclever_opt_seq_get:NN</code>	470, 480
.. 3739, 3886, 4800, 4818, 4826, 4834	<code>_zrefclever_opt_seq_get:NN(TF)</code>	18
<code>\l_zrefclever_next_is_same_bool</code>	<code>_zrefclever_opt_seq_get:NNTF ..</code>	
.....	... 470, 816, 853, 934, 947, 2855,	
93, 122, 3732,	2868, 5248, 5253, 5258, 5263, 5268	
4451, 4484, 4501, 4507, 5165, 5203	<code>_zrefclever_opt_seq_gset_-</code>	
<code>\l_zrefclever_next_maybe_range_-</code>	<code>clist_split:Nn</code>	16,
<code>bool</code>	422, 424, 1374, 1407, 3287, 3312	
. 93, 122, 3732, 4250, 4263, 4450,	<code>_zrefclever_opt_seq_gset_eq:NN</code>	
4477, 4490, 5157, 5164, 5183, 5201	16, 422,
<code>\l_zrefclever_noabbrev_first_-</code>	436, 442, 1383, 1416, 2936, 3296, 3321	
<code>bool</code>	<code>_zrefclever_opt_seq_if_set:N .</code>	
1991, 1994, 4923	454, 469
<code>\g_zrefclever_nocompat_bool ...</code>	<code>_zrefclever_opt_seq_if_-</code>	
.....	<code>set:N(TF)</code>	17
2413, 2420, 2456	<code>_zrefclever_opt_seq_if_set:NNTF</code>	
<code>\l_zrefclever_nocompat_bool ...</code>	454, 472, 1025, 1367, 1399
... 64	<code>_zrefclever_opt_seq_set_clist_-</code>	
<code>\g_zrefclever_nocompat_modules_-</code>	<code>split:Nn ..</code>	16, 422, 422, 2524, 2788
<code>seq</code>	<code>_zrefclever_opt_seq_set_eq:NN .</code>	
2414, 2424, 2427, 2448, 2457, 2458	16, 422, 426, 435, 2530, 2794
<code>\l_zrefclever_nocompat_modules_-</code>	<code>_zrefclever_opt_seq_unset:N ...</code>	
<code>seq</code>	17, 443, 443, 453, 2519, 2780
64	<code>_zrefclever_opt_tl_clear:N ...</code>	
<code>\l_zrefclever_nudge_comptosing_-</code>	14, 332,
<code>bool ...</code>	342, 351, 1643, 1648, 1663, 1678,	
2177, 2207, 2216, 2222, 4947	1693, 2639, 2644, 2659, 2674, 2689	
<code>\l_zrefclever_nudge_enabled_-</code>	<code>_zrefclever_opt_tl_cset_-</code>	
<code>bool</code>	<code>fallback:nn</code>	1522, 1529
2175, 2185, 2187,	<code>_zrefclever_opt_tl_gclear:N ...</code>	
2191, 2192, 2197, 2198, 4422, 4933	14, 332,
<code>\l_zrefclever_nudge_gender_bool</code>	359, 365, 3014, 3020, 3028, 3035,	
.....	3056, 3072, 3093, 3109, 3130, 3146	
2179, 2208, 2218, 2223, 4961	<code>_zrefclever_opt_tl_gclear_if_-</code>	
<code>\l_zrefclever_nudge_multitype_-</code>	<code>new:N</code>	16, 391,
<code>bool ...</code>	401, 410, 1123, 1129, 1137, 1144,	
2176, 2206, 2214, 2221, 4423	1165, 1181, 1202, 1218, 1239, 1255	
<code>\l_zrefclever_nudge_singular_-</code>	<code>_zrefclever_opt_tl_get:NN</code>	411, 421
<code>bool</code>		
2178, 2234, 4935		
<code>_zrefclever_opt_bool_get:NN ...</code>		
.....		
542, 552		
<code>_zrefclever_opt_bool_get:NN(TF)</code>		
.....		
19		
<code>_zrefclever_opt_bool_get:NNTF .</code>		
... 542, 5281, 5286, 5291, 5296, 5301		
<code>_zrefclever_opt_bool_gset_-</code>		
<code>false:N</code>		
19,		
508, 535, 541, 1490, 1507, 3369, 3377		
<code>_zrefclever_opt_bool_gset_-</code>		
<code>true:N</code>		
19,		
508, 528, 534, 1452, 1469, 3348, 3356		
<code>_zrefclever_opt_bool_if:N</code>		
553, 560		

<code>__zrefclever_opt_tl_get:NN(TF)</code>	. 16	1220, 1249, 1257, 1298, 1306, 1336,
<code>__zrefclever_opt_tl_get:NNTF</code>	...	1346, 1401, 1418, 1463, 1471, 1501,
	411 , 4997, 5012, 5031, 5040, 5061,	1509, 2938, 2993, 3030, 3037, 3066,
	5071, 5215, 5220, 5225, 5230, 5235	3074, 3103, 3111, 3140, 3148, 3189,
<code>__zrefclever_opt_tl_gset:N</code>	... 14	3197, 3254, 3264, 3323, 3358, 3379,
<code>__zrefclever_opt_tl_gset:Nn</code>	...	5014, 5063, 5073, 5226, 5259, 5292
	.. 332 , 352 , 358 , 2959, 2983, 2991,	<code>__zrefclever_opt_varname_-</code>
	3049 , 3064 , 3086 , 3101 , 3123 , 3138 ,	<code>language:nnn</code>
	3171 , 3178 , 3187 , 3195 , 3252 , 3262 12 , 294 ,
<code>__zrefclever_opt_tl_gset_if_-</code>		294, 304, 769, 774, 784, 789, 800,
<code>new:Nn</code> 16,	805, 818, 855, 888, 936, 949, 2857, 2870
	391 , 391, 400, 1066, 1091, 1100,	<code>__zrefclever_opt_varname_-</code>
	1158 , 1173 , 1195 , 1210 , 1232 , 1247 ,	<code>type:nnn</code>
	1280 , 1287 , 1296 , 1304 , 1334 , 1344	12, 291 , 291, 293, 2616,
<code>__zrefclever_opt_tl_if_set:N</code>	.. 377	2623, 2641, 2646, 2655, 2661, 2670,
<code>__zrefclever_opt_tl_if_set:N(TF)</code> 15	2676, 2685, 2691, 2700, 2705, 2725,
 377 , 393 , 403 , 413	2732, 2782, 2796, 2820, 2829, 2838,
<code>__zrefclever_opt_tl_if_set:NNTF</code>	4999, 5033, 5042, 5221, 5254, 5287
 377 , 393 , 403 , 413	<code>__zrefclever_orig_ltxlabel:n</code>
<code>__zrefclever_opt_tl_set:N</code> 14 5468, 5475, 5480
<code>__zrefclever_opt_tl_set:Nn</code>	<code>\g_zrefclever_page_format_tl</code>
 332 , 332 , 341 , 7, 106, 111, 114
	1657 , 1672 , 1687 , 1727 , 1733 , 2475 ,	<code>\l_zrefclever_pairsep_tl</code>
	2621 , 2653 , 2668 , 2683 , 2723 , 2730 3739 , 3890, 4013,
<code>__zrefclever_opt_tl_unset:N</code>	...	4036, 4062, 4114, 4145, 4184, 4267
 15, 366 , 366 ,	<code>__zrefclever_process_language_-</code>
	376 , 1702 , 1707 , 2471 , 2614 , 2698 , 2703	<code>settings:</code>
<code>__zrefclever_opt_var_set_bool:n</code> 57, 58, 811 , 811, 3400
 14, 330 , 330, 337 ,	<code>__zrefclever_prop_put_non_-</code>
	338 , 339, 347, 348, 349, 371, 372,	<code>empty:Nnn</code>
	373 , 381, 383, 431, 432, 433, 448,	42, 1541 , 1541, 2338, 2393
	449 , 450, 458, 460, 486, 487, 488,	<code>__zrefclever_provide_langfile:n</code>
	496 , 498, 513, 514, 515, 523, 524, 525 21,
<code>__zrefclever_opt_varname_-</code>		30, 31, 83, 913 , 913, 969, 2120, 3399
<code>fallback:nn</code> 14,	<code>\l_zrefclever_range_beg_is_-</code>
	328 , 328, 1525, 5236, 5269, 5302	<code>first_bool</code>
<code>__zrefclever_opt_varname_-</code>	 3732 ,
<code>general:nn</code> 12,	3782, 4101, 4137, 4171, 4442,
	289 , 289, 1645, 1650, 1659, 1665,	4479, 4535, 4580, 4602, 4631, 4684
	1674 , 1680, 1689, 1695, 1704, 1709,	<code>\l_zrefclever_range_beg_label_-</code>
	1729 , 1735, 2472, 2476, 2520, 2531,	<code>tl</code>
	2552 , 2557, 2562, 5216, 5249, 5282 93,
<code>__zrefclever_opt_varname_lang_-</code>		3732 , 3775, 4064, 4077, 4116, 4125,
<code>default:nnn</code>	.. 13, 305, 305, 315,	4155, 4186, 4200, 4210, 4435, 4480,
	1068 , 1093 , 1125 , 1131 , 1160 , 1167 ,	4497, 4548, 4594, 4616, 4645, 4659
	1197 , 1204 , 1234 , 1241 , 1282 , 1289 ,	<code>\l_zrefclever_range_count_int</code>
	1369 , 1385 , 1447 , 1454 , 1485 , 1492 , 92,
	2961 , 2985 , 3016 , 3022 , 3051 , 3058 ,	3732 , 3780, 4026, 4090, 4440, 4483,
	3088 , 3095 , 3125 , 3132 , 3173 , 3180 ,	4494, 4500, 4506, 4514, 4573, 4686
	3298 , 3350 , 3371 , 5231 , 5264 , 5297	<code>\l_zrefclever_range_end_ref_tl</code>
<code>__zrefclever_opt_varname_lang_-</code>		... 3732 , 3776, 4212, 4218, 4287,
<code>type:nnnn</code> 13,	4293, 4436, 4482, 4499, 4661, 4667
	316 , 316, 327, 1027, 1036, 1044,	<code>\l_zrefclever_range_same_count_-</code>
	1102 , 1139 , 1146 , 1175 , 1183 , 1212 ,	<code>int</code>
	 92,
		3732 , 3781, 4004, 4055, 4091, 4441,
		4485, 4502, 4508, 4553, 4574, 4687
		<code>\l_zrefclever_rangeseq_tl</code>
	 3739 , 3902,
		4215, 4225, 4290, 4300, 4664, 4674

\l_zrefclever_rangetopair_bool .	\l_zrefclever_refbounds_mid_re-
..... 3739, 3930, 4251	seq 3754, 3958, 4668, 4676
\l_zrefclever_ref_count_int ...	\l_zrefclever_refbounds_mid_seq
..... 3717, 3779, 3754, 3950, 4078, 4156,
4032, 4110, 4180, 4438, 4471, 4520,	4525, 4549, 4561, 4595, 4617, 4625
4543, 4555, 4589, 4611, 4620, 4640	\l_zrefclever_reffont_tl
\l_zrefclever_ref_decl_case_tl 3739, 3910, 4714,
..... 27, 825, 830, 831, 835, 838,	4726, 4755, 4766, 4804, 4861, 4876
842, 846, 849, 894, 897, 899, 2161,	\l_zrefclever_reftype_override-
2171, 5006, 5010, 5049, 5054, 5059	tl 34, 44, 2320, 2323
_zrefclever_ref_default: 4693,	\g_zrefclever_rf_opts_bool-
4693, 4734, 4740, 4778, 4819, 4885	maybe_type_specific_seq
\l_zrefclever_ref_gender_tl 52, 53, 566, 1436, 2544, 2811, 3339
..... 28, 862, 868,	\g_zrefclever_rf_opts_seq-
869, 873, 876, 881, 882, 901, 907,	refbounds_seq
908, 2180, 2244, 4962, 4970, 4976, 4984 566, 1358, 2510, 2771, 3277
\l_zrefclever_ref_language_tl ..	\g_zrefclever_rf_opts_tl_maybe-
..... 24, 27, 55, 713, 717, 814,	type_specific_seq 566, 1082, 2974
819, 829, 847, 856, 866, 880, 889,	\g_zrefclever_rf_opts_tl_not-
898, 905, 2102, 2108, 2113, 2121,	type_specific_seq
2139, 2145, 2150, 3399, 3417, 3788, 566, 1057, 2593, 2950
3792, 3796, 3885, 3889, 3893, 3897,	\g_zrefclever_rf_opts_tl-
3901, 3905, 3909, 3913, 3917, 3921,	reference_seq 566, 2462
3925, 3929, 3933, 3937, 3941, 3945,	\g_zrefclever_rf_opts_tl_type-
3949, 3953, 3957, 3961, 3965, 3969,	names_seq 566, 1319, 3237
4966, 4978, 4989, 5015, 5064, 5074	\g_zrefclever_rf_opts_tl-
\l_zrefclever_ref_property_tl ..	typesetup_seq 566, 2605
..... 43, 48, 1547,	\l_zrefclever_setup_language_tl
1556, 1563, 1565, 1749, 1773, 1817, 561, 742, 770, 775, 785,
1834, 1846, 1853, 1858, 1891, 1898,	790, 801, 806, 932, 984, 991, 1005,
1903, 1933, 1940, 1945, 3492, 3815,	1022, 1028, 1037, 1045, 1069, 1094,
3870, 3874, 4699, 4784, 4838, 5147	1103, 1126, 1132, 1140, 1147, 1161,
\l_zrefclever_ref_propserity_tl 3460	1168, 1176, 1184, 1198, 1205, 1213,
\l_zrefclever_ref_typeset_font-	1221, 1235, 1242, 1250, 1258, 1283,
tl 2248, 2250, 3410	1290, 1299, 1307, 1337, 1347, 1370,
\l_zrefclever_refbounds_first-	1386, 1402, 1419, 1448, 1455, 1464,
pb_seq 3754,	1472, 1486, 1493, 1502, 1510, 2854,
3942, 4018, 4068, 4140, 4191, 4272	2895, 2902, 2916, 2933, 2939, 2962,
\l_zrefclever_refbounds_first-	2986, 2994, 3017, 3023, 3031, 3038,
rb_seq . 3754, 3946, 4174, 4306, 4635	3052, 3059, 3067, 3075, 3089, 3096,
\l_zrefclever_refbounds_first-	3104, 3112, 3126, 3133, 3141, 3149,
seq 3754, 3934, 4317, 4538, 4584, 4606	3174, 3181, 3190, 3198, 3255, 3265,
\l_zrefclever_refbounds_first-	3299, 3324, 3351, 3359, 3372, 3380
sg_seq . 3754, 3938, 3996, 4007, 4104	\l_zrefclever_setup_type_tl 561,
\l_zrefclever_refbounds_last-	933, 975, 976, 1008, 1029, 1038,
pe_seq 3754, 3966,	1046, 1064, 1089, 1104, 1121, 1141,
4015, 4038, 4065, 4117, 4147, 4269	1148, 1156, 1177, 1185, 1193, 1214,
\l_zrefclever_refbounds_last-	1222, 1230, 1251, 1259, 1278, 1300,
re_seq	1308, 1326, 1338, 1348, 1365, 1403,
.. 3754, 3970, 4219, 4227, 4294, 4302	1420, 1443, 1465, 1473, 1481, 1503,
\l_zrefclever_refbounds_last-	1511, 2588, 2590, 2617, 2624, 2642,
seq 3754, 3962, 4046, 4081, 4126, 4162	2647, 2656, 2662, 2671, 2677, 2686,
\l_zrefclever_refbounds_mid_rb-	2692, 2701, 2706, 2726, 2733, 2783,
seq ... 3754, 3954, 4187, 4201, 4646	2797, 2821, 2830, 2839, 2853, 2886,

2887, 2919, 2940, 2957, 2981, 2995,
3012, 3032, 3039, 3047, 3068, 3076,
3084, 3105, 3113, 3121, 3142, 3150,
3169, 3191, 3199, 3244, 3256, 3266,
3284, 3325, 3346, 3360, 3367, 3381
\l_zrefclever_sort_decided_bool
..... 3453, 3536, 3550, 3560,
3564, 3576, 3586, 3601, 3616, 3640
_zrefclever_sort_default:nn ...
..... 87, 3494, 3510, 3510
_zrefclever_sort_default_
different_types:nn
.... 44, 85, 86, 90, 3521, 3653, 3653
_zrefclever_sort_default_same_
type:nn ... 85, 88, 3520, 3523, 3523
_zrefclever_sort_labels:
..... 86, 87, 91, 3408, 3457, 3457
_zrefclever_sort_page:nn
..... 91, 3493, 3705, 3705
\l_zrefclever_sort_prior_a_int .
..... 3454,
3655, 3661, 3662, 3668, 3678, 3686
\l_zrefclever_sort_prior_b_int .
..... 3454,
3656, 3663, 3664, 3671, 3679, 3687
\l_zrefclever_tlastsep_tl
..... 3739, 3797, 4416
\l_zrefclever_tlistsep_tl
..... 3739, 3793, 4387
\l_zrefclever_tpairsep_tl
..... 3739, 3789, 4409
\l_zrefclever_type_count_int ...
. 92, 117, 3717, 3778, 4384, 4386,
4399, 4424, 4439, 4910, 4922, 5117
\l_zrefclever_type_first_label_
tl 92,
115, 3720, 3773, 3991, 4242, 4253,
4257, 4285, 4336, 4353, 4357, 4433,
4467, 4777, 4783, 4789, 4793, 4806,
4837, 4851, 4855, 4863, 4878, 4891
\l_zrefclever_type_first_label_
type_tl ... 92, 117, 3720, 3774,
3993, 4246, 4434, 4469, 4898, 4941,
4957, 4965, 4977, 4983, 5000, 5016,
5026, 5034, 5043, 5065, 5075, 5087
\l_zrefclever_type_first_
refbounds_seq
... 3754, 3995, 4006, 4017, 4067,
4103, 4139, 4173, 4190, 4271, 4305,
4316, 4337, 4537, 4583, 4605, 4634,
4801, 4802, 4809, 4811, 4847, 4859,
4867, 4870, 4873, 4874, 4881, 4882
\l_zrefclever_type_first_
refbounds_set_bool 3754,
3783, 3997, 4008, 4019, 4070, 4106,
4142, 4176, 4193, 4273, 4307,
4314, 4443, 4540, 4586, 4608, 4637
\l_zrefclever_type_name_gender_
seq ... 3726, 4967, 4969, 4972, 4987
\l_zrefclever_type_name_
missing_bool
.. 3726, 4823, 4894, 4901, 5023, 5083
_zrefclever_type_name_setup: ..
..... 20, 22, 115, 4323, 4889, 4889
\l_zrefclever_type_name_tl
..... 115, 117,
3726, 4360, 4366, 4798, 4816, 4831,
4833, 4893, 4900, 5004, 5020, 5022,
5038, 5047, 5069, 5079, 5081, 5101
\l_zrefclever_typeset_compress_
bool 1625, 1628, 4452
\l_zrefclever_typeset_labels_
seq 92, 3714, 3769, 3803, 3805, 3811
\l_zrefclever_typeset_last_bool
..... 92, 3714,
3800, 3801, 3808, 3833, 4396, 5116
\l_zrefclever_typeset_name_bool
.. 1574, 1581, 1586, 1591, 4325, 4341
\l_zrefclever_typeset_queue_
curr_tl 92,
95, 112, 117, 3720, 3772, 4011,
4034, 4042, 4060, 4073, 4112,
4121, 4143, 4151, 4158, 4182, 4196,
4213, 4223, 4240, 4265, 4288, 4298,
4327, 4334, 4344, 4377, 4393, 4404,
4410, 4417, 4431, 4432, 4521, 4544,
4556, 4590, 4612, 4621, 4641, 4662,
4672, 4915, 4937, 4948, 5111, 5115
\l_zrefclever_typeset_queue_
prev_tl . 92, 3720, 3771, 4388, 4430
\l_zrefclever_typeset_range_
bool ... 1759, 1977, 1980, 3407, 4238
\l_zrefclever_typeset_ref_bool .
.. 1573, 1580, 1585, 1590, 4325, 4331
_zrefclever_typeset_refs:
..... 92-94, 3411, 3767, 3767
_zrefclever_typeset_refs_last_
of_type:
.. 99, 112, 115, 117, 3976, 3981, 3981
_zrefclever_typeset_refs_not_
last_of_type:
... 93, 99, 112, 122, 3978, 4446, 4446
\l_zrefclever_typeset_sort_bool
..... 1601, 1604, 3406
\l_zrefclever_typesort_seq
. 44, 90, 1610, 1615, 1616, 1622, 3657
\l_zrefclever_verbose_testing_
bool 3766, 4392

_zrefclever_zcref:nnn	bool	2276, 2293, 3403, 3424
..... 27, 56, 3392, 3393	_zrefclever_zcsetup:n	
_zrefclever_zcref:nnnn 83, 86, 3393	. 67, 2582, 2583, 2583, 2585, 5315,	
\l_zrefclever_zcref_labels_seq .	5339, 5345, 5353, 5375, 5394, 5444,	
..... 86, 87, 3397,	5448, 5449, 5458, 5486, 5520, 5571,	
3428, 3433, 3437, 3462, 3465, 3770	5596, 5608, 5620, 5635, 5655, 5678	
\l_zrefclever_zcref_note_tl ...	\l_zrefclever_zrefcheck_-	
..... 2269, 2272, 3413, 3420	available_bool	
\l_zrefclever_zcref_with_check_-	.. 2275, 2288, 2300, 2312, 3402, 3423	