

The zref-clever package*

Code documentation

Gustavo Barros[†]

2023-06-14

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	68
5.3	\zcLanguageSetup	73
6	User interface	83
6.1	\zceref	83
6.2	\zcpageref	85
7	Sorting	85
8	Typesetting	92

*This file describes v0.4.0, released 2023-06-14.

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

9	Compatibility	126
9.1	appendix	126
9.2	appendices	127
9.3	memoir	129
9.4	amsmath	130
9.5	mathtools	132
9.6	breqn	133
9.7	listings	134
9.8	enumitem	134
9.9	subcaption	135
9.10	subfig	135
10	Language files	136
10.1	Localization guidelines	136
10.2	English	139
10.3	German	142
10.4	French	151
10.5	Portuguese	155
10.6	Spanish	160
10.7	Dutch	164
10.8	Italian	168
	Index	173

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

Identify the internal prefix (L^AT_EX3 DocStrip convention).

```
2 <@=zrefclever>
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from l3candidates). We presume xparse (which made to the kernel in the 2020-10-01 release), and expl3 as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Also, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (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>). Finally, and critically, the new `label` hook introduced in the 2023-06-01 release, alongside the corresponding new hooks with arguments, just simplifies and improves label setting so much, by allowing `\zlabel` to be set with `\label`, that it is definitely a must for `zref-clever`, so we require that too. Hence we make the cut at the 2023-06-01 kernel release.

```
3 \NeedsTeXFormat{LaTeX2e}
```

```
4 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
```

```

5 \IfFormatAtLeastTF{2023-06-01}
6 {}
7 {%
8   \PackageError{zref-clever}{LaTeX kernel too old}
9   {%
10    'zref-clever' requires a LaTeX kernel 2023-06-01 or newer.%
11    \MessageBreak Loading will abort!%
12   }%
13 \endinput
14 }%

Identify the package.

15 \ProvidesExplPackage {zref-clever} {2023-06-14} {0.4.0}
16 {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>).

```

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

```

3 zref setup

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

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

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

```

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

```

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

```

23 \zref@newprop { thecounter }

```

```

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

```

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

```

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

```

Since the default/`thecounter` and `page` properties store the “*printed* representation” of their respective counters, for sorting and compressing purposes, we are also interested in their numeric values. So we store them in `zc@cntval` and `zc@pgval`. For this, we use `\c@⟨counter⟩`, which contains the counter’s numerical value (see ‘`texdoc source2e`’, section ‘`ltcounts.dtx`’). 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.

```

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

```

```

57         { \cs_if_exist_p:c { c@ \@currentcounter } }
58         { \int_use:c { c@ \@currentcounter } }
59         { 0 }
60     }
61 }
62 \zref@addprop \ZREF@mainlist { zc@cntval }
63 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
64 \zref@addprop \ZREF@mainlist { zc@pgval }

```

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

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

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

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

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

`__zrefclever_get_enclosing_counters_value:n`

Recursively generate a *sequence* of “enclosing counters” values, for a given $\langle counter \rangle$ 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 {\langle counter \rangle}

65 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
66 {
67   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
68   {
69     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
70     \__zrefclever_get_enclosing_counters_value:e
71     { \__zrefclever_counter_reset_by:n {#1} }
72   }
73 }

```

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).

```

74 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

```

(End of definition for `__zrefclever_get_enclosing_counters_value:n`.)

`__zrefclever_counter_reset_by:n`

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

```

\__zrefclever_counter_reset_by:n {\langle counter \rangle}

75 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
76 {
77   \bool_if:nTF
78   { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
79   { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
80   {
81     \seq_map_tokens:Nn \l__zrefclever_counter_resettters_seq
82     { \__zrefclever_counter_reset_by_aux:nn {#1} }
83   }
84 }
85 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
86 {

```

```

87 \cs_if_exist:cT { c@ #2 }
88 {
89     \tl_if_empty:cF { cl@ #2 }
90     {
91         \tl_map_tokens:cn { cl@ #2 }
92         { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
93     }
94 }
95 }
96 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
97 {
98     \str_if_eq:nnT {#2} {#3}
99     { \tl_map_break:n { \seq_map_break:n {#1} } }
100 }

```

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

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

```

101 \zref@newprop { zc@enclval }
102 {
103     \__zrefclever_get_enclosing_counters_value:e
104     \l__zrefclever_current_counter_tl
105 }
106 \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}`.

```

107 \tl_new:N \g__zrefclever_page_format_tl
108 \AddToHook { shipout / before }
109 {
110     \group_begin:
111     \int_set:Nn \c@page { 1 }
112     \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
113     \group_end:
114 }
115 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
116 \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.

```

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

```

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

4.2 Messages

```

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

```



```

153 }
154 \msg_new:nnn { zref-clever } { unknown-language-decl }
155 {
156   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:..~
157   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
158   '\iow_char:N\zcDeclareLanguageAlias'.
159 }
160 \msg_new:nnn { zref-clever } { language-no-decl-ref }
161 {
162   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
163   Nothing~to~do~with~option~'d=#2'.
164 }
165 \msg_new:nnn { zref-clever } { language-no-gender }
166 {
167   Language~'#1'~has~no~declared~gender~\msg_line_context:..~
168   Nothing~to~do~with~option~'#2=#3'.
169 }
170 \msg_new:nnn { zref-clever } { language-no-decl-setup }
171 {
172   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
173   Nothing~to~do~with~option~'case=#2'.
174 }
175 \msg_new:nnn { zref-clever } { unknown-decl-case }
176 {
177   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:..~
178   Using~default~declension~case.
179 }
180 \msg_new:nnn { zref-clever } { nudge-multitype }
181 {
182   Reference~with~multiple~types~\msg_line_context:..~
183   You~may~wish~to~separate~them~or~review~language~around~it.
184 }
185 \msg_new:nnn { zref-clever } { nudge-comptosing }
186 {
187   Multiple~labels~have~been~compressed~into~singular~type~name~
188   for~type~'#1'~\msg_line_context:.
189 }
190 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
191 {
192   Option~'sg'~signals~that~a~singular~type~name~was~expected~
193   \msg_line_context:..~But~type~'#1'~has~plural~type~name.
194 }
195 \msg_new:nnn { zref-clever } { gender-not-declared }
196 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
197 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
198 {
199   Gender~mismatch~for~type~'#1'~\msg_line_context:..~
200   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
201 }
202 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
203 {
204   You've~specified~'g=#1'~\msg_line_context:..~
205   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
206 }

```

```

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

```

```

261 { Range-for-type-~'~#1'~resulted-in~single-element~\msg_line_context:. }
262 \msg_new:nnn { zref-clever } { compat-package }
263 { Loaded-support-for-~'~#1'~package. }
264 \msg_new:nnn { zref-clever } { compat-class }
265 { Loaded-support-for-~'~#1'~documentclass. }
266 \msg_new:nnn { zref-clever } { option-deprecated }
267 {
268   Option-~'~#1'~has-been-deprecated~\msg_line_context:.\iow_newline:
269   Use-~'~#2'~instead.
270 }
271 \msg_new:nnn { zref-clever } { load-time-options }
272 {
273   'zref-clever'~does-not-accept-load-time-options.~
274   To-configure-package-options,~use-~'\iow_char:N\zcsetup'.
275 }

```

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

```

(End of definition for `_zrefclever_extract_default:Nnnn`.)

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

```

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

```

(End of definition for `_zrefclever_extract_unexp:nnn`.)

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

```

\__zrefclever_extract:nnn{\langle label \rangle}{\langle prop \rangle}{\langle default \rangle}
288 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
289 { \zref@extractdefault {#1} {#2} {#3} }

```

(End of definition for `_zrefclever_extract:nnn`.)

4.4 Option infra

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

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

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

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

(End of definition for `_zrefclever_opt_varname_general:nn`.)

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

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

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

(End of definition for `_zrefclever_opt_varname_type:nnn`.)

`_zrefclever_opt_varname_language:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the language $\langle option \rangle$ 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}
```

```

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

```

(End of definition for __zrefclever_opt_varname_language:nnn.)

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

```

\__zrefclever_opt_varname_lang_default:nnn {<lang>} {<option>} {<data type>}

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

```

(End of definition for __zrefclever_opt_varname_lang_default:nnn.)

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

```

\__zrefclever_opt_varname_lang_type:nnnn {<lang>} {<ref type>}
{<option>} {<data type>}

317 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
318 {
319   \__zrefclever_language_if_declared:nTF {#1}
320   {
321     g__zrefclever_opt_lang_
322     \tl_use:c { \__zrefclever_language_varname:n {#1} }
323     _type_ #2 _ #3 _ #4
324   }
325   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
326 }
327 \cs_generate_variant:Nn
328 \__zrefclever_opt_varname_lang_type:nnnn { eenn , eenen }

```

(End of definition for __zrefclever_opt_varname_lang_type:nnnn.)

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

```

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

(End of definition for \_zrefclever_opt_varname_fallback:nn.)

```

_zrefclever_opt_var_set_bool:n

The L^AT_EX3 programming layer does not have the concept of a variable *existing* only locally, it also considers an “error” if an assignment is made to a variable which was not previously declared, but declaration is always global, which means that “setting a local variable at a local scope”, given these requirements, results in it existing, and being empty, globally. Therefore, we need an independent mechanism from the mere existence of a variable to keep track of whether variables are “set” or “unset”, within the logic of the precedence rules for options in different scopes. _zrefclever_opt_var_set_bool:n expands to the name of the boolean variable used to track this state for *<option var>*. See discussion with Phelype Oleinik at https://tex.stackexchange.com/questions/633341/#comment1579825_633347

```

    \_zrefclever_opt_var_set_bool:n {\<option var>}}
333 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
332 { \cs_to_str:N #1 _is_set_bool }

(End of definition for \_zrefclever_opt_var_set_bool:n.)

```

```

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

```

```

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

```

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

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

```

\__zrefclever_opt_tl_unset:N {\langle option\ tl \rangle}

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

```

(End of definition for __zrefclever_opt_tl_unset:N.)

__zrefclever_opt_tl_if_set:N \underline{TF} 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}

378 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
379 {
380   \tl_if_exist:NTF #1
381   {
382     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
383     {
384       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
385         { \prg_return_true: }
386         { \prg_return_false: }
387     }
388     { \prg_return_true: }
389   }
390   { \prg_return_false: }
391 }

```

(End of definition for __zrefclever_opt_tl_if_set:N \underline{TF} .)

```

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

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

```

```

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

```

(End of definition for __zrefclever_opt_tl_get:NNTF.)

```

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

```



```

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

```

(End of definition for __zrefclever_opt_seq_set_clist_split:Nn and others.)

__zrefclever_opt_seq_unset:N Unset \langle option seq \rangle .

```

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

```

(End of definition for __zrefclever_opt_seq_unset:N.)

__zrefclever_opt_seq_if_set:N \underline{TF} This conditional *defines* what means to be unset for a sequence option.

```

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

```

(End of definition for __zrefclever_opt_seq_if_set:N \underline{TF} .)

```

__zrefclever_opt_seq_get:NNTF      \__zrefclever_opt_seq_get:NN(TF) {\langle option seq to get \rangle} {\langle seq var to set \rangle}
                                   {\langle true \rangle} {\langle false \rangle}
471 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
472 {
473   \__zrefclever_opt_seq_if_set:NTF #1
474   {
475     \seq_set_eq:NN #2 #1
476     \prg_return_true:
477   }
478   { \prg_return_false: }
479 }
480 \prg_generate_conditional_variant:Nnn
481   \__zrefclever_opt_seq_get:NN { cN } { F }

(End of definition for \__zrefclever_opt_seq_get:NTTF.)

```

_zrefclever_opt_bool_unset:N Unset $\langle option\ bool \rangle$.

```

      \__zrefclever_opt_bool_unset:N {\langle option bool \rangle}
482 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
483 {
484   \bool_if_exist:NT #1
485   {
486     % \bool_set_false:N #1 % ?
487     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
488       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
489       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
490   }
491 }
492 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }

(End of definition for \__zrefclever_opt_bool_unset:N.)

```

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

```

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

(End of definition for \__zrefclever_opt_bool_if_set:NTF.)

```

```

\__zrefclever_opt_bool_set_true:N
\__zrefclever_opt_bool_set_false:N
\__zrefclever_opt_bool_gset_true:N
\__zrefclever_opt_bool_gset_false:N
509 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
510 {
511   \bool_if_exist:NF #1
512   { \bool_new:N #1 }
513   \bool_set_true:N #1
514   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
515   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
516   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
517 }
518 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
519 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
520 {
521   \bool_if_exist:NF #1
522   { \bool_new:N #1 }
523   \bool_set_false:N #1
524   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
525   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
526   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
527 }
528 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
529 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
530 {
531   \bool_if_exist:NF #1
532   { \bool_new:N #1 }
533   \bool_gset_true:N #1
534 }
535 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
536 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
537 {
538   \bool_if_exist:NF #1
539   { \bool_new:N #1 }
540   \bool_gset_false:N #1
541 }
542 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }

```

(End of definition for __zrefclever_opt_bool_set_true:N and others.)

```

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

```

(End of definition for `__zrefclever_opt_bool_get:NNTF`.)

```

\__zrefclever_opt_bool_if:NNTF      \__zrefclever_opt_bool_if:N(TF) {\langle option bool \rangle} {\langle true \rangle} {\langle false \rangle}
554 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
555 {
556   \__zrefclever_opt_bool_if_set:NNTF #1
557   { \bool_if:NNTF #1 { \prg_return_true: } { \prg_return_false: } }
558   { \prg_return_false: }
559 }
560 \prg_generate_conditional_variant:Nnn
561   \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

(End of definition for `__zrefclever_opt_bool_if:NNTF`.)

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 `\zceref`, 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
562 \tl_new:N \l_zrefclever_setup_type_tl
563 \tl_new:N \l_zrefclever_setup_language_tl
564 \tl_new:N \l_zrefclever_lang_decl_case_tl
565 \seq_new:N \l_zrefclever_lang_declension_seq
566 \seq_new:N \l_zrefclever_lang_gender_seq
```

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

`zrefclever_rf_opts_tl_not_type_specific_seq` Lists of reference format options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent. These variables are *constants*, but I don’t seem to be able to find a way to concatenate two constants into a third one without triggering L^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.

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

```

598     refbounds-mid-re ,
599     refbounds-last ,
600     refbounds-last-pe ,
601     refbounds-last-re ,
602   }
603 \seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
604 \seq_gset_from_clist:Nn
605   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
606   {
607     cap ,
608     abbrev ,
609     rangetopair ,
610   }

```

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:.`

```

611 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
612 \seq_gset_from_clist:Nn
613   \g__zrefclever_rf_opts_tl_type_names_seq
614   {
615     Name-sg ,
616     name-sg ,
617     Name-pl ,
618     name-pl ,
619     Name-sg-ab ,
620     name-sg-ab ,
621     Name-pl-ab ,
622     name-pl-ab ,
623   }

```

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

```

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

```

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

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

```

632 \clist_map_inline:nn
633   {
634     reference ,
635     typesetup ,
636     langsetup ,
637     langfile ,
638   }
639   {
640     \keys_define:nn { zref-clever/ #1 }
641     {

```

```

642 +refbounds-first .meta:n =
643 {
644     refbounds-first = {##1} ,
645     refbounds-first-sg = {##1} ,
646     refbounds-first-pb = {##1} ,
647     refbounds-first-rb = {##1} ,
648 } ,
649 +refbounds-mid .meta:n =
650 {
651     refbounds-mid = {##1} ,
652     refbounds-mid-rb = {##1} ,
653     refbounds-mid-re = {##1} ,
654 } ,
655 +refbounds-last .meta:n =
656 {
657     refbounds-last = {##1} ,
658     refbounds-last-pe = {##1} ,
659     refbounds-last-re = {##1} ,
660 } ,
661 +refbounds-rb .meta:n =
662 {
663     refbounds-first-rb = {##1} ,
664     refbounds-mid-rb = {##1} ,
665 } ,
666 +refbounds-re .meta:n =
667 {
668     refbounds-mid-re = {##1} ,
669     refbounds-last-re = {##1} ,
670 } ,
671 +refbounds .meta:n =
672 {
673     +refbounds-first = {##1} ,
674     +refbounds-mid = {##1} ,
675     +refbounds-last = {##1} ,
676 } ,
677 refbounds .meta:n = { +refbounds = {##1} } ,
678 }
679 }
680 \clist_map_inline:nn
681 {
682     reference ,
683     typesetup ,
684 }
685 {
686     \keys_define:nn { zref-clever/ #1 }
687     {
688         +refbounds-first .default:o = \c_novalue_tl ,
689         +refbounds-mid .default:o = \c_novalue_tl ,
690         +refbounds-last .default:o = \c_novalue_tl ,
691         +refbounds-rb .default:o = \c_novalue_tl ,
692         +refbounds-re .default:o = \c_novalue_tl ,
693         +refbounds .default:o = \c_novalue_tl ,
694         refbounds .default:o = \c_novalue_tl ,
695     }

```

```

696 }
697 \clist_map_inline:nn
698 {
699   langsetup ,
700   langfile ,
701 }
702 {
703   \keys_define:nn { zref-clever/ #1 }
704   {
705     +refbounds-first .value_required:n = true ,
706     +refbounds-mid .value_required:n = true ,
707     +refbounds-last .value_required:n = true ,
708     +refbounds-rb .value_required:n = true ,
709     +refbounds-re .value_required:n = true ,
710     +refbounds .value_required:n = true ,
711     refbounds .value_required:n = true ,
712   }
713 }

```

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.

```

714 \tl_new:N \l__zrefclever_ref_language_tl
715 \tl_new:N \l__zrefclever_current_language_tl
716 \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.

```

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

```

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

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

```

    \__zrefclever_language_varname:n {<language>}
719 \cs_new:Npn \__zrefclever_language_varname:n #1
720 { g__zrefclever_declared_language_ #1 _tl }

```

(End of definition for __zrefclever_language_varname:n.)

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

```

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

```

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

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

```

\__zrefclever_language_if_declared:n(TF) {<language>}

723 \prg_new_conditional:Npnn \__zrefclever_language_if_declared:n #1 { T , F , TF }
724 {
725   \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
726   { \prg_return_true: }
727   { \prg_return_false: }
728 }
729 \prg_generate_conditional_variant:Nnn
730 \__zrefclever_language_if_declared:n { x } { T , F , TF }

```

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

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

```

731 \prg_set_eq_conditional:Nnn \zrefclever_language_if_declared:n
732 \__zrefclever_language_if_declared:n { TF }

```

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

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

```

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

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

```

```

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

```

(End of definition for \zcDeclareLanguage.)

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

```

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

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

```

(End of definition for \zcDeclareLanguageAlias.)

```

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

```

```

790         \__zrefclever_opt_varname_language:enn
791         { \l__zrefclever_setup_language_tl } { gender } { seq }
792     }
793     {#1}
794 } ,
795 gender .value_required:n = true ,
796 allcaps .choices:nn =
797     { true , false }
798     {
799         \bool_new:c
800         {
801             \__zrefclever_opt_varname_language:enn
802             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
803         }
804         \use:c { bool_gset_ \l_keys_choice_tl :c }
805         {
806             \__zrefclever_opt_varname_language:enn
807             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
808         }
809     } ,
810 allcaps .default:n = true ,
811 }

```

`__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.

```

812 \cs_new_protected:Npn \__zrefclever_process_language_settings:
813 {
814     \__zrefclever_language_if_declared:xTF
815     { \l__zrefclever_ref_language_tl }
816     {

```

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.

```

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

```

```

827         {
828             \msg_warning:nxxx { zref-clever }
829             { language-no-decl-ref }
830             { \l__zrefclever_ref_language_tl }
831             { \l__zrefclever_ref_decl_case_tl }
832             \tl_clear:N \l__zrefclever_ref_decl_case_tl
833         }
834     }
835     {
836         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
837         {
838             \seq_get_left:NN \l__zrefclever_lang_declension_seq
839             \l__zrefclever_ref_decl_case_tl
840         }
841         {
842             \seq_if_in:NVF \l__zrefclever_lang_declension_seq
843             \l__zrefclever_ref_decl_case_tl
844             {
845                 \msg_warning:nxxx { zref-clever }
846                 { unknown-decl-case }
847                 { \l__zrefclever_ref_decl_case_tl }
848                 { \l__zrefclever_ref_language_tl }
849                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
850                 \l__zrefclever_ref_decl_case_tl
851             }
852         }
853     }

```

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.

```

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

```

```

877         \l__zrefclever_ref_gender_tl
878     {
879         \msg_warning:nxxx { zref-clever }
880         { gender-not-declared }
881         { \l__zrefclever_ref_language_tl }
882         { \l__zrefclever_ref_gender_tl }
883         \tl_clear:N \l__zrefclever_ref_gender_tl
884     }
885 }
886

```

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

```

887     \__zrefclever_opt_bool_if:cT
888     {
889         \__zrefclever_opt_varname_language:enn
890         { \l__zrefclever_ref_language_tl } { allcaps } { bool }
891     }
892     { \keys_set:nn { zref-clever/reference } { cap = true } }
893 }
894 {

```

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

```

895     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
896     {
897         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
898         { \l__zrefclever_ref_decl_case_tl }
899         { \l__zrefclever_ref_language_tl }
900         \tl_clear:N \l__zrefclever_ref_decl_case_tl
901     }
902     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
903     {
904         \msg_warning:nxxx { zref-clever }
905         { language-no-gender }
906         { \l__zrefclever_ref_language_tl }
907         { g }
908         { \l__zrefclever_ref_gender_tl }
909         \tl_clear:N \l__zrefclever_ref_gender_tl
910     }
911 }
912 }

```

(End of definition for `__zrefclever_process_language_settings:`)

4.7 Language files

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

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

parsimony, of “loading the least possible”. Therefore, we load at `begindocument` one single language (see `lang option`), as specified by the user in the preamble with the `lang` option or, failing any specification, the current language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

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

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

```
913 \seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(End of definition for `\g__zrefclever_loaded_langfiles_seq`.)

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

```

\__zrefclever_provide_langfile:n {<language>}

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

```

```

926      zref-clever-
927      \tl_use:c { \__zrefclever_language_varname:n {#1} }
928      .lang
929    }
930    { \ExplSyntaxOn }
931    \l_tmpa_tl
932    {
933      \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
934      \tl_clear:N \l__zrefclever_setup_type_tl
935      \__zrefclever_opt_seq_get:cNF
936      {
937        \__zrefclever_opt_varname_language:nnn
938        {#1} { declension } { seq }
939      }
940      \l__zrefclever_lang_declension_seq
941      { \seq_clear:N \l__zrefclever_lang_declension_seq }
942      \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
943      { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
944      {
945        \seq_get_left:NN \l__zrefclever_lang_declension_seq
946        \l__zrefclever_lang_decl_case_tl
947      }
948      \__zrefclever_opt_seq_get:cNF
949      {
950        \__zrefclever_opt_varname_language:nnn
951        {#1} { gender } { seq }
952      }
953      \l__zrefclever_lang_gender_seq
954      { \seq_clear:N \l__zrefclever_lang_gender_seq }
955      \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
956      \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
957      { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
958      \msg_info:nnx { zref-clever } { langfile-loaded }
959      { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
960    }
961    {

```

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.

```

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

```

(End of definition for __zrefclever_provide_langfile:n.)

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

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

```

971 \keys_define:nn { zref-clever/langfile }
972 {
973   type .code:n =
974   {
975     \tl_if_empty:NTF {#1}
976     { \tl_clear:N \l__zrefclever_setup_type_tl }
977     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
978   } ,
979
980   case .code:n =
981   {
982     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
983     {
984       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
985       { \l__zrefclever_setup_language_tl } {#1}
986     }
987     {
988       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
989       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
990       {
991         \msg_info:nxxx { zref-clever } { unknown-decl-case }
992         {#1} { \l__zrefclever_setup_language_tl }
993         \seq_get_left:NN \l__zrefclever_lang_declension_seq
994         \l__zrefclever_lang_decl_case_tl
995       }
996     }
997   } ,
998   case .value_required:n = true ,
999
1000   gender .value_required:n = true ,
1001   gender .code:n =
1002   {
1003     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1004     {
1005       \msg_info:nxxx { zref-clever } { language-no-gender }
1006       { \l__zrefclever_setup_language_tl } { gender } {#1}
1007     }
1008     {
1009       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1010       {
1011         \msg_info:nnn { zref-clever }
1012         { option-only-type-specific } { gender }
1013       }
1014       {
1015         \seq_clear:N \l_tmpa_seq
1016         \clist_map_inline:nn {#1}
1017         {
1018           \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1019           { \seq_put_right:Nn \l_tmpa_seq {##1} }
1020           {
1021             \msg_info:nxxx { zref-clever }
1022             { gender-not-declared }

```



```

1023         { \l__zrefclever_setup_language_tl } {##1}
1024     }
1025 }
1026 \__zrefclever_opt_seq_if_set:cF
1027 {
1028     \__zrefclever_opt_varname_lang_type:eenn
1029     { \l__zrefclever_setup_language_tl }
1030     { \l__zrefclever_setup_type_tl }
1031     { gender }
1032     { seq }
1033 }
1034 {
1035     \seq_new:c
1036     {
1037         \__zrefclever_opt_varname_lang_type:eenn
1038         { \l__zrefclever_setup_language_tl }
1039         { \l__zrefclever_setup_type_tl }
1040         { gender }
1041         { seq }
1042     }
1043     \seq_gset_eq:cN
1044     {
1045         \__zrefclever_opt_varname_lang_type:eenn
1046         { \l__zrefclever_setup_language_tl }
1047         { \l__zrefclever_setup_type_tl }
1048         { gender }
1049         { seq }
1050     }
1051     \l_tmpa_seq
1052 }
1053 }
1054 }
1055 },
1056 }
1057 \seq_map_inline:Nn
1058 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1059 {
1060     \keys_define:nn { zref-clever/langfile }
1061     {
1062         #1 .value_required:n = true ,
1063         #1 .code:n =
1064         {
1065             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1066             {
1067                 \__zrefclever_opt_tl_gset_if_new:cn
1068                 {
1069                     \__zrefclever_opt_varname_lang_default:enn
1070                     { \l__zrefclever_setup_language_tl }
1071                     {#1} { tl }
1072                 }
1073                 {##1}
1074             }
1075         }
1076         \msg_info:nnn { zref-clever }

```

```

1077         { option-not-type-specific } {#1}
1078     }
1079 },
1080 }
1081 }
1082 \seq_map_inline:Nn
1083 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1084 {
1085     \keys_define:nn { zref-clever/langfile }
1086     {
1087         #1 .value_required:n = true ,
1088         #1 .code:n =
1089         {
1090             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1091             {
1092                 \__zrefclever_opt_tl_gset_if_new:cn
1093                 {
1094                     \__zrefclever_opt_varname_lang_default:enn
1095                     { \l__zrefclever_setup_language_tl }
1096                     {#1} { t1 }
1097                 }
1098                 {##1}
1099             }
1100             {
1101                 \__zrefclever_opt_tl_gset_if_new:cn
1102                 {
1103                     \__zrefclever_opt_varname_lang_type:enn
1104                     { \l__zrefclever_setup_language_tl }
1105                     { \l__zrefclever_setup_type_tl }
1106                     {#1} { t1 }
1107                 }
1108                 {##1}
1109             }
1110         },
1111     }
1112 }
1113 \keys_define:nn { zref-clever/langfile }
1114 {
1115     endrange .value_required:n = true ,
1116     endrange .code:n =
1117     {
1118         \str_case:nnF {#1}
1119         {
1120             { ref }
1121             {
1122                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1123                 {
1124                     \__zrefclever_opt_tl_gclear_if_new:c
1125                     {
1126                         \__zrefclever_opt_varname_lang_default:enn
1127                         { \l__zrefclever_setup_language_tl }
1128                         { endrangefunc } { t1 }
1129                     }
1130                     \__zrefclever_opt_tl_gclear_if_new:c

```

```

1131         {
1132             \__zrefclever_opt_varname_lang_default:enn
1133             { \l__zrefclever_setup_language_tl }
1134             { endrangeprop } { tl }
1135         }
1136     }
1137     {
1138         \__zrefclever_opt_tl_gclear_if_new:c
1139         {
1140             \__zrefclever_opt_varname_lang_type:eenn
1141             { \l__zrefclever_setup_language_tl }
1142             { \l__zrefclever_setup_type_tl }
1143             { endrangefunc } { tl }
1144         }
1145         \__zrefclever_opt_tl_gclear_if_new:c
1146         {
1147             \__zrefclever_opt_varname_lang_type:eenn
1148             { \l__zrefclever_setup_language_tl }
1149             { \l__zrefclever_setup_type_tl }
1150             { endrangeprop } { tl }
1151         }
1152     }
1153 }
1154
1155 { stripprefix }
1156 {
1157     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1158     {
1159         \__zrefclever_opt_tl_gset_if_new:cn
1160         {
1161             \__zrefclever_opt_varname_lang_default:enn
1162             { \l__zrefclever_setup_language_tl }
1163             { endrangefunc } { tl }
1164         }
1165         { __zrefclever_get_endrange_stripprefix }
1166         \__zrefclever_opt_tl_gclear_if_new:c
1167         {
1168             \__zrefclever_opt_varname_lang_default:enn
1169             { \l__zrefclever_setup_language_tl }
1170             { endrangeprop } { tl }
1171         }
1172     }
1173     {
1174         \__zrefclever_opt_tl_gset_if_new:cn
1175         {
1176             \__zrefclever_opt_varname_lang_type:eenn
1177             { \l__zrefclever_setup_language_tl }
1178             { \l__zrefclever_setup_type_tl }
1179             { endrangefunc } { tl }
1180         }
1181         { __zrefclever_get_endrange_stripprefix }
1182         \__zrefclever_opt_tl_gclear_if_new:c
1183         {
1184             \__zrefclever_opt_varname_lang_type:eenn

```

```

1185         { \l__zrefclever_setup_language_tl }
1186         { \l__zrefclever_setup_type_tl }
1187         { endrangeprop } { tl }
1188     }
1189 }
1190 }
1191
1192 { pagecomp }
1193 {
1194     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1195     {
1196         \__zrefclever_opt_tl_gset_if_new:cn
1197         {
1198             \__zrefclever_opt_varname_lang_default:enn
1199             { \l__zrefclever_setup_language_tl }
1200             { endrangefunc } { tl }
1201         }
1202         { __zrefclever_get_endrange_pagecomp }
1203         \__zrefclever_opt_tl_gclear_if_new:c
1204         {
1205             \__zrefclever_opt_varname_lang_default:enn
1206             { \l__zrefclever_setup_language_tl }
1207             { endrangeprop } { tl }
1208         }
1209     }
1210     {
1211         \__zrefclever_opt_tl_gset_if_new:cn
1212         {
1213             \__zrefclever_opt_varname_lang_type:eenn
1214             { \l__zrefclever_setup_language_tl }
1215             { \l__zrefclever_setup_type_tl }
1216             { endrangefunc } { tl }
1217         }
1218         { __zrefclever_get_endrange_pagecomp }
1219         \__zrefclever_opt_tl_gclear_if_new:c
1220         {
1221             \__zrefclever_opt_varname_lang_type:eenn
1222             { \l__zrefclever_setup_language_tl }
1223             { \l__zrefclever_setup_type_tl }
1224             { endrangeprop } { tl }
1225         }
1226     }
1227 }
1228
1229 { pagecomp2 }
1230 {
1231     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1232     {
1233         \__zrefclever_opt_tl_gset_if_new:cn
1234         {
1235             \__zrefclever_opt_varname_lang_default:enn
1236             { \l__zrefclever_setup_language_tl }
1237             { endrangefunc } { tl }
1238         }

```

```

1239         { __zrefclever_get_endrange_pagecomptwo }
1240     \__zrefclever_opt_tl_gclear_if_new:c
1241     {
1242         \__zrefclever_opt_varname_lang_default:enn
1243         { \l__zrefclever_setup_language_tl }
1244         { endrangeprop } { tl }
1245     }
1246 }
1247 {
1248     \__zrefclever_opt_tl_gset_if_new:cn
1249     {
1250         \__zrefclever_opt_varname_lang_type:enn
1251         { \l__zrefclever_setup_language_tl }
1252         { \l__zrefclever_setup_type_tl }
1253         { endrangefunc } { tl }
1254     }
1255     { __zrefclever_get_endrange_pagecomptwo }
1256     \__zrefclever_opt_tl_gclear_if_new:c
1257     {
1258         \__zrefclever_opt_varname_lang_type:enn
1259         { \l__zrefclever_setup_language_tl }
1260         { \l__zrefclever_setup_type_tl }
1261         { endrangeprop } { tl }
1262     }
1263 }
1264 }
1265 }
1266 {
1267     \tl_if_empty:nTF {#1}
1268     {
1269         \msg_info:nnn { zref-clever }
1270         { endrange-property-undefined } {#1}
1271     }
1272     {
1273         \zref@ifpropundefined {#1}
1274         {
1275             \msg_info:nnn { zref-clever }
1276             { endrange-property-undefined } {#1}
1277         }
1278         {
1279             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1280             {
1281                 \__zrefclever_opt_tl_gset_if_new:cn
1282                 {
1283                     \__zrefclever_opt_varname_lang_default:enn
1284                     { \l__zrefclever_setup_language_tl }
1285                     { endrangefunc } { tl }
1286                 }
1287                 { __zrefclever_get_endrange_property }
1288                 \__zrefclever_opt_tl_gset_if_new:cn
1289                 {
1290                     \__zrefclever_opt_varname_lang_default:enn
1291                     { \l__zrefclever_setup_language_tl }
1292                     { endrangeprop } { tl }

```

```

1293     }
1294     {#1}
1295   }
1296   {
1297     \__zrefclever_opt_tl_gset_if_new:cn
1298     {
1299       \__zrefclever_opt_varname_lang_type:eenn
1300       { \l__zrefclever_setup_language_tl }
1301       { \l__zrefclever_setup_type_tl }
1302       { endrangefunc } { tl }
1303     }
1304     { __zrefclever_get_endrange_property }
1305     \__zrefclever_opt_tl_gset_if_new:cn
1306     {
1307       \__zrefclever_opt_varname_lang_type:eenn
1308       { \l__zrefclever_setup_language_tl }
1309       { \l__zrefclever_setup_type_tl }
1310       { endrangeprop } { tl }
1311     }
1312     {#1}
1313   }
1314 }
1315 }
1316 }
1317 } ,
1318 }
1319 \seq_map_inline:Nn
1320 \g__zrefclever_rf_opts_tl_type_names_seq
1321 {
1322   \keys_define:nn { zref-clever/langfile }
1323   {
1324     #1 .value_required:n = true ,
1325     #1 .code:n =
1326     {
1327       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1328       {
1329         \msg_info:nnn { zref-clever }
1330         { option-only-type-specific } {#1}
1331       }
1332       {
1333         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1334         {
1335           \__zrefclever_opt_tl_gset_if_new:cn
1336           {
1337             \__zrefclever_opt_varname_lang_type:eenn
1338             { \l__zrefclever_setup_language_tl }
1339             { \l__zrefclever_setup_type_tl }
1340             {#1} { tl }
1341           }
1342           {##1}
1343         }
1344         {
1345           \__zrefclever_opt_tl_gset_if_new:cn
1346           {

```

```

1347         \_zrefclever_opt_varname_lang_type:eeen
1348         { \l__zrefclever_setup_language_tl }
1349         { \l__zrefclever_setup_type_tl }
1350         { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
1351     }
1352     {##1}
1353 }
1354 }
1355 } ,
1356 }
1357 }
1358 \seq_map_inline:Nn
1359 \g__zrefclever_rf_opts_seq_refbounds_seq
1360 {
1361     \keys_define:nn { zref-clever/langfile }
1362     {
1363         #1 .value_required:n = true ,
1364         #1 .code:n =
1365         {
1366             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1367             {
1368                 \_zrefclever_opt_seq_if_set:cF
1369                 {
1370                     \_zrefclever_opt_varname_lang_default:enn
1371                     { \l__zrefclever_setup_language_tl } {#1} { seq }
1372                 }
1373                 {
1374                     \seq_gclear:N \g_tmpa_seq
1375                     \_zrefclever_opt_seq_gset_clist_split:Nn
1376                     \g_tmpa_seq {##1}
1377                     \bool_lazy_or:nnTF
1378                     { \tl_if_empty_p:n {##1} }
1379                     {
1380                         \int_compare_p:nNn
1381                         { \seq_count:N \g_tmpa_seq } = { 4 }
1382                     }
1383                     {
1384                         \_zrefclever_opt_seq_gset_eq:cN
1385                         {
1386                             \_zrefclever_opt_varname_lang_default:enn
1387                             { \l__zrefclever_setup_language_tl }
1388                             {#1} { seq }
1389                         }
1390                         \g_tmpa_seq
1391                     }
1392                     {
1393                         \msg_info:nxxx { zref-clever }
1394                         { refbounds-must-be-four }
1395                         {#1} { \seq_count:N \g_tmpa_seq }
1396                     }
1397                 }
1398             }
1399             {
1400                 \_zrefclever_opt_seq_if_set:cF

```

```

1401 {
1402     \__zrefclever_opt_varname_lang_type:eenn
1403     { \l__zrefclever_setup_language_tl }
1404     { \l__zrefclever_setup_type_tl } {#1} { seq }
1405 }
1406 {
1407     \seq_gclear:N \g_tmpa_seq
1408     \__zrefclever_opt_seq_gset_clist_split:Nn
1409     \g_tmpa_seq {##1}
1410     \bool_lazy_or:nnTF
1411     { \tl_if_empty_p:n {##1} }
1412     {
1413         \int_compare_p:nNn
1414         { \seq_count:N \g_tmpa_seq } = { 4 }
1415     }
1416     {
1417         \__zrefclever_opt_seq_gset_eq:cN
1418         {
1419             \__zrefclever_opt_varname_lang_type:eenn
1420             { \l__zrefclever_setup_language_tl }
1421             { \l__zrefclever_setup_type_tl }
1422             {#1} { seq }
1423         }
1424         \g_tmpa_seq
1425     }
1426     {
1427         \msg_info:nxxx { zref-clever }
1428         { refbounds-must-be-four }
1429         {#1} { \seq_count:N \g_tmpa_seq }
1430     }
1431 }
1432 }
1433 } ,
1434 }
1435 }
1436 \seq_map_inline:Nn
1437 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1438 {
1439     \keys_define:nn { zref-clever/langfile }
1440     {
1441         #1 .choice: ,
1442         #1 / true .code:n =
1443         {
1444             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1445             {
1446                 \__zrefclever_opt_bool_if_set:cF
1447                 {
1448                     \__zrefclever_opt_varname_lang_default:enn
1449                     { \l__zrefclever_setup_language_tl }
1450                     {#1} { bool }
1451                 }
1452                 {
1453                     \__zrefclever_opt_bool_gset_true:c
1454                     {

```



```

1455         \_zrefclever_opt_varname_lang_default:enn
1456         { \l__zrefclever_setup_language_tl }
1457         {#1} { bool }
1458     }
1459 }
1460 }
1461 {
1462     \_zrefclever_opt_bool_if_set:cF
1463     {
1464         \_zrefclever_opt_varname_lang_type:enn
1465         { \l__zrefclever_setup_language_tl }
1466         { \l__zrefclever_setup_type_tl }
1467         {#1} { bool }
1468     }
1469     {
1470         \_zrefclever_opt_bool_gset_true:c
1471         {
1472             \_zrefclever_opt_varname_lang_type:enn
1473             { \l__zrefclever_setup_language_tl }
1474             { \l__zrefclever_setup_type_tl }
1475             {#1} { bool }
1476         }
1477     }
1478 }
1479 },
1480 #1 / false .code:n =
1481 {
1482     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1483     {
1484         \_zrefclever_opt_bool_if_set:cF
1485         {
1486             \_zrefclever_opt_varname_lang_default:enn
1487             { \l__zrefclever_setup_language_tl }
1488             {#1} { bool }
1489         }
1490         {
1491             \_zrefclever_opt_bool_gset_false:c
1492             {
1493                 \_zrefclever_opt_varname_lang_default:enn
1494                 { \l__zrefclever_setup_language_tl }
1495                 {#1} { bool }
1496             }
1497         }
1498     }
1499 }
1500 \_zrefclever_opt_bool_if_set:cF
1501 {
1502     \_zrefclever_opt_varname_lang_type:enn
1503     { \l__zrefclever_setup_language_tl }
1504     { \l__zrefclever_setup_type_tl }
1505     {#1} { bool }
1506 }
1507 {
1508     \_zrefclever_opt_bool_gset_false:c

```

```

1509         {
1510             \__zrefclever_opt_varname_lang_type:eenn
1511             { \l__zrefclever_setup_language_tl }
1512             { \l__zrefclever_setup_type_tl }
1513             {#1} { bool }
1514         }
1515     }
1516 }
1517 } ,
1518 #1 .default:n = true ,
1519 no #1 .meta:n = { #1 = false } ,
1520 no #1 .value_forbidden:n = true ,
1521 }
1522 }

```

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.

```

1523 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1524 {
1525     \tl_const:cn
1526     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1527 }
1528 \keyval_parse:nnn
1529 { }
1530 { \__zrefclever_opt_tl_cset_fallback:nn }
1531 {
1532     tpairsep = {,~} ,
1533     tlistsep = {,~} ,
1534     tlastsep = {,~} ,
1535     notesep  = {~} ,
1536     namesep  = {\nobreakspace} ,
1537     pairsep  = {,~} ,
1538     listsep  = {,~} ,
1539     lastsep  = {,~} ,
1540     rangesep = {\textendash} ,
1541 }

```

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>}

1542 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1543 {
1544     \tl_if_empty:nTF {#3}
1545     { \prop_remove:Nn #1 {#2} }

```

```

1546     { \prop_put:Nnn #1 {#2} {#3} }
1547   }

```

(End of definition for `_zrefclever_prop_put_non_empty:Nnn`.)

ref option

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

```

1548 \tl_new:N \l__zrefclever_ref_property_tl
1549 \keys_define:nn { zref-clever/reference }
1550 {
1551   ref .code:n =
1552   {
1553     \tl_if_empty:nTF {#1}
1554     {
1555       \msg_warning:nnn { zref-clever }
1556       { zref-property-undefined } {#1}
1557       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1558     }
1559     {
1560       \zref@ifpropundefined {#1}
1561       {
1562         \msg_warning:nnn { zref-clever }
1563         { zref-property-undefined } {#1}
1564         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1565       }
1566       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1567     }
1568   } ,
1569   ref .initial:n = default ,
1570   ref .value_required:n = true ,
1571   page .meta:n = { ref = page },
1572   page .value_forbidden:n = true ,
1573 }

```

typeset option

```

1574 \bool_new:N \l__zrefclever_typeset_ref_bool
1575 \bool_new:N \l__zrefclever_typeset_name_bool
1576 \keys_define:nn { zref-clever/reference }
1577 {
1578   typeset .choice: ,
1579   typeset / both .code:n =
1580   {
1581     \bool_set_true:N \l__zrefclever_typeset_ref_bool

```

```

1582     \bool_set_true:N \l__zrefclever_typeset_name_bool
1583   } ,
1584   typeset / ref .code:n =
1585   {
1586     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1587     \bool_set_false:N \l__zrefclever_typeset_name_bool
1588   } ,
1589   typeset / name .code:n =
1590   {
1591     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1592     \bool_set_true:N \l__zrefclever_typeset_name_bool
1593   } ,
1594   typeset .initial:n = both ,
1595   typeset .value_required:n = true ,
1596
1597   noname .meta:n = { typeset = ref } ,
1598   noname .value_forbidden:n = true ,
1599   noref .meta:n = { typeset = name } ,
1600   noref .value_forbidden:n = true ,
1601 }

```

sort option

```

1602 \bool_new:N \l__zrefclever_typeset_sort_bool
1603 \keys_define:nn { zref-clever/reference }
1604 {
1605   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1606   sort .initial:n = true ,
1607   sort .default:n = true ,
1608   nosort .meta:n = { sort = false } ,
1609   nosort .value_forbidden:n = true ,
1610 }

```

typesort option

\l__zrefclever_typesort_seq is stored reversed, since the sort priorities are computed in the negative range in __zrefclever_sort_default_different_types:nn, so that we can implicitly rely on ‘0’ being the “last value”, and spare creating an integer variable using \seq_map_indexed_inline:Nn.

```

1611 \seq_new:N \l__zrefclever_typesort_seq
1612 \keys_define:nn { zref-clever/reference }
1613 {
1614   typesort .code:n =
1615   {
1616     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1617     \seq_reverse:N \l__zrefclever_typesort_seq
1618   } ,
1619   typesort .initial:n =
1620   { part , chapter , section , paragraph } ,
1621   typesort .value_required:n = true ,
1622   notypesort .code:n =
1623   { \seq_clear:N \l__zrefclever_typesort_seq } ,
1624   notypesort .value_forbidden:n = true ,
1625 }

```

comp option

```
1626 \bool_new:N \l__zrefclever_typeset_compress_bool
1627 \keys_define:nn { zref-clever/reference }
1628 {
1629   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1630   comp .initial:n = true ,
1631   comp .default:n = true ,
1632   nocomp .meta:n = { comp = false },
1633   nocomp .value_forbidden:n = true ,
1634 }
```

endrange option

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

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

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

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

```
1635 \NewHook { zref-clever/endrange-setup }
1636 \keys_define:nn { zref-clever/reference }
1637 {
1638   endrange .code:n =
1639   {
1640     \str_case:nnF {#1}
1641     {
1642       { ref }

```

```

1643 {
1644   \_zrefclever_opt_tl_clear:c
1645   {
1646     \_zrefclever_opt_varname_general:nn
1647     { endrangefunc } { t1 }
1648   }
1649   \_zrefclever_opt_tl_clear:c
1650   {
1651     \_zrefclever_opt_varname_general:nn
1652     { endrangeprop } { t1 }
1653   }
1654 }
1655
1656 { stripprefix }
1657 {
1658   \_zrefclever_opt_tl_set:cn
1659   {
1660     \_zrefclever_opt_varname_general:nn
1661     { endrangefunc } { t1 }
1662   }
1663   { __zrefclever_get_endrange_stripprefix }
1664   \_zrefclever_opt_tl_clear:c
1665   {
1666     \_zrefclever_opt_varname_general:nn
1667     { endrangeprop } { t1 }
1668   }
1669 }
1670
1671 { pagecomp }
1672 {
1673   \_zrefclever_opt_tl_set:cn
1674   {
1675     \_zrefclever_opt_varname_general:nn
1676     { endrangefunc } { t1 }
1677   }
1678   { __zrefclever_get_endrange_pagecomp }
1679   \_zrefclever_opt_tl_clear:c
1680   {
1681     \_zrefclever_opt_varname_general:nn
1682     { endrangeprop } { t1 }
1683   }
1684 }
1685
1686 { pagecomp2 }
1687 {
1688   \_zrefclever_opt_tl_set:cn
1689   {
1690     \_zrefclever_opt_varname_general:nn
1691     { endrangefunc } { t1 }
1692   }
1693   { __zrefclever_get_endrange_pagecomptwo }
1694   \_zrefclever_opt_tl_clear:c
1695   {
1696     \_zrefclever_opt_varname_general:nn

```

```

1697         { endrangeprop } { t1 }
1698     }
1699 }
1700
1701 { unset }
1702 {
1703     \__zrefclever_opt_t1_unset:c
1704     {
1705         \__zrefclever_opt_varname_general:nn
1706         { endrangefunc } { t1 }
1707     }
1708     \__zrefclever_opt_t1_unset:c
1709     {
1710         \__zrefclever_opt_varname_general:nn
1711         { endrangeprop } { t1 }
1712     }
1713 }
1714 }
1715 {
1716     \tl_if_empty:nTF {#1}
1717     {
1718         \msg_warning:nnn { zref-clever }
1719         { endrange-property-undefined } {#1}
1720     }
1721     {
1722         \zref@ifpropundefined {#1}
1723         {
1724             \msg_warning:nnn { zref-clever }
1725             { endrange-property-undefined } {#1}
1726         }
1727         {
1728             \__zrefclever_opt_t1_set:cn
1729             {
1730                 \__zrefclever_opt_varname_general:nn
1731                 { endrangefunc } { t1 }
1732             }
1733             { __zrefclever_get_endrange_property }
1734             \__zrefclever_opt_t1_set:cn
1735             {
1736                 \__zrefclever_opt_varname_general:nn
1737                 { endrangeprop } { t1 }
1738             }
1739             {#1}
1740         }
1741     }
1742 }
1743 } ,
1744 endrange .value_required:n = true ,
1745 }
1746 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1747 {
1748     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1749     {
1750         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }

```

```

1751     {
1752       \__zrefclever_extract_default:Nnvn #3
1753       {#2} { l__zrefclever_ref_property_tl } { }
1754     }
1755     { \tl_set:Nn #3 { zc@missingproperty } }
1756   }
1757   {
1758     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1759     {

```

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.

```

1760     \bool_if:NTF \l__zrefclever_typeset_range_bool
1761     {
1762       \group_begin:
1763       \bool_set_false:N \l_tmpa_bool
1764       \exp_args:Nxx \tl_if_eq:nnT
1765       {
1766         \__zrefclever_extract_unexp:nnn
1767         {#1} { externaldocument } { }
1768       }
1769       {
1770         \__zrefclever_extract_unexp:nnn
1771         {#2} { externaldocument } { }
1772       }
1773     }
1774     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1775     {
1776       \exp_args:Nxx \tl_if_eq:nnT
1777       {
1778         \__zrefclever_extract_unexp:nnn
1779         {#1} { zc@pgfmt } { }
1780       }
1781       {
1782         \__zrefclever_extract_unexp:nnn
1783         {#2} { zc@pgfmt } { }
1784       }
1785       { \bool_set_true:N \l_tmpa_bool }
1786     }
1787     {
1788       \exp_args:Nxx \tl_if_eq:nnT
1789       {
1790         \__zrefclever_extract_unexp:nnn
1791         {#1} { zc@counter } { }
1792       }
1793       {
1794         \__zrefclever_extract_unexp:nnn
1795         {#2} { zc@counter } { }
1796       }
1797     }
1798     \exp_args:Nxx \tl_if_eq:nnT

```



```

1799         {
1800             \__zrefclever_extract_unexp:nnn
1801             {#1} { zc@enclval } { }
1802         }
1803         {
1804             \__zrefclever_extract_unexp:nnn
1805             {#2} { zc@enclval } { }
1806         }
1807         { \bool_set_true:N \l_tmpa_bool }
1808     }
1809 }
1810 }
1811 \bool_if:NTF \l_tmpa_bool
1812 {
1813     \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1814     {#2} { l__zrefclever_endrangeprop_tl } { }
1815 }
1816 {
1817     \zref@ifrefcontainsprop
1818     {#2} { \l__zrefclever_ref_property_tl }
1819     {
1820         \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1821         {#2} { l__zrefclever_ref_property_tl } { }
1822     }
1823     { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1824 }
1825 \exp_args:NNNV
1826 \group_end:
1827 \tl_set:Nn #3 \l_tmpb_tl
1828 }
1829 {
1830     \__zrefclever_extract_default:Nnvn #3
1831     {#2} { l__zrefclever_endrangeprop_tl } { }
1832 }
1833 }
1834 {
1835     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1836     {
1837         \__zrefclever_extract_default:Nnvn #3
1838         {#2} { l__zrefclever_ref_property_tl } { }
1839     }
1840     { \tl_set:Nn #3 { zc@missingproperty } }
1841 }
1842 }
1843 }
1844 \cs_generate_variant:Nn \__zrefclever_get_endrange_property:nnN { VVN }

For the technique for smuggling the assignment out of the group, see Enrico Grego-
rio's answer at https://tex.stackexchange.com/a/56314.

1845 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1846 {
1847     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1848     {
1849         \group_begin:

```

```

1850 \UseHook { zref-clever/endrange-setup }
1851 \tl_set:Nx \l_tmpa_tl
1852 {
1853   \__zrefclever_extract:nnn
1854   {#1} { \l__zrefclever_ref_property_tl } { }
1855 }
1856 \tl_set:Nx \l_tmpb_tl
1857 {
1858   \__zrefclever_extract:nnn
1859   {#2} { \l__zrefclever_ref_property_tl } { }
1860 }
1861 \bool_set_false:N \l_tmpa_bool
1862 \bool_until_do:Nn \l_tmpa_bool
1863 {
1864   \exp_args:Nxx \tl_if_eq:nnTF
1865   { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1866   {
1867     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1868     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1869     \tl_if_empty:NT \l_tmpb_tl
1870     { \bool_set_true:N \l_tmpa_bool }
1871   }
1872   { \bool_set_true:N \l_tmpa_bool }
1873 }
1874 \exp_args:NNNV
1875 \group_end:
1876 \tl_set:Nn #3 \l_tmpb_tl
1877 }
1878 { \tl_set:Nn #3 { zc@missingproperty } }
1879 }
1880 \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>).

```

1881 \prg_new_protected_conditional:Npnn
1882 \__zrefclever_is_integer_rgx:n #1 { F , TF }
1883 {
1884   \regex_match:nnTF { \A\d+\Z } {#1}
1885   { \prg_return_true: }
1886   { \prg_return_false: }
1887 }
1888 \prg_generate_conditional_variant:Nnn
1889 \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

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

```

1890 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1891 {
1892   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1893   {
1894     \group_begin:
1895     \UseHook { zref-clever/endrange-setup }
1896     \tl_set:Nx \l_tmpa_tl
1897     {
1898       \__zrefclever_extract:nnn

```

```

1899         {#1} { \l__zrefclever_ref_property_tl } { }
1900     }
1901     \tl_set:Nx \l_tmpb_tl
1902     {
1903         \__zrefclever_extract:nnn
1904         {#2} { \l__zrefclever_ref_property_tl } { }
1905     }
1906     \bool_set_false:N \l_tmpa_bool
1907     \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1908     {
1909         \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1910         { \bool_set_true:N \l_tmpa_bool }
1911     }
1912     { \bool_set_true:N \l_tmpa_bool }
1913     \bool_until_do:Nn \l_tmpa_bool
1914     {
1915         \exp_args:Nxx \tl_if_eq:nnTF
1916         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1917         {
1918             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1919             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1920             \tl_if_empty:NT \l_tmpb_tl
1921             { \bool_set_true:N \l_tmpa_bool }
1922         }
1923         { \bool_set_true:N \l_tmpa_bool }
1924     }
1925     \exp_args:NNNV
1926     \group_end:
1927     \tl_set:Nn #3 \l_tmpb_tl
1928 }
1929 { \tl_set:Nn #3 { zc@missingproperty } }
1930 }
1931 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1932 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1933 {
1934     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1935     {
1936         \group_begin:
1937         \UseHook { zref-clever/endrange-setup }
1938         \tl_set:Nx \l_tmpa_tl
1939         {
1940             \__zrefclever_extract:nnn
1941             {#1} { \l__zrefclever_ref_property_tl } { }
1942         }
1943         \tl_set:Nx \l_tmpb_tl
1944         {
1945             \__zrefclever_extract:nnn
1946             {#2} { \l__zrefclever_ref_property_tl } { }
1947         }
1948         \bool_set_false:N \l_tmpa_bool
1949         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1950         {
1951             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1952             { \bool_set_true:N \l_tmpa_bool }

```

```

1953     }
1954     { \bool_set_true:N \l_tmpa_bool }
1955 \bool_until_do:Nn \l_tmpa_bool
1956 {
1957   \exp_args:Nxx \tl_if_eq:nnTF
1958   { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1959   {
1960     \bool_lazy_or:nnTF
1961     { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1962     { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1963     {
1964       \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1965       \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1966     }
1967     { \bool_set_true:N \l_tmpa_bool }
1968   }
1969   { \bool_set_true:N \l_tmpa_bool }
1970 }
1971 \exp_args:NNNV
1972 \group_end:
1973 \tl_set:Nn #3 \l_tmpb_tl
1974 }
1975 { \tl_set:Nn #3 { zc@missingproperty } }
1976 }
1977 \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`.

```

1978 \bool_new:N \l__zrefclever_typeset_range_bool
1979 \keys_define:nn { zref-clever/reference }
1980 {
1981   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1982   range .initial:n = false ,
1983   range .default:n = true ,
1984 }

```

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`.

```

1985 \bool_new:N \l__zrefclever_capfirst_bool
1986 \keys_define:nn { zref-clever/reference }
1987 {
1988   capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1989   capfirst .initial:n = false ,
1990   capfirst .default:n = true ,
1991 }

```

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`.

```
1992 \bool_new:N \l__zrefclever_noabbrev_first_bool
1993 \keys_define:nn { zref-clever/reference }
1994 {
1995     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1996     noabbrevfirst .initial:n = false ,
1997     noabbrevfirst .default:n = true ,
1998 }
```

S option

```
1999 \keys_define:nn { zref-clever/reference }
2000 {
2001     S .meta:n =
2002     { capfirst = {#1} , noabbrevfirst = {#1} },
2003     S .default:n = true ,
2004 }
```

hyperref option

```
2005 \bool_new:N \l__zrefclever_hyperlink_bool
2006 \bool_new:N \l__zrefclever_hyperref_warn_bool
2007 \keys_define:nn { zref-clever/reference }
2008 {
2009     hyperref .choice: ,
2010     hyperref / auto .code:n =
2011     {
2012         \bool_set_true:N \l__zrefclever_hyperlink_bool
2013         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2014     } ,
2015     hyperref / true .code:n =
2016     {
2017         \bool_set_true:N \l__zrefclever_hyperlink_bool
2018         \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2019     } ,
2020     hyperref / false .code:n =
2021     {
2022         \bool_set_false:N \l__zrefclever_hyperlink_bool
2023         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2024     } ,
2025     hyperref .initial:n = auto ,
2026     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.

```
2027     nohyperref .meta:n = { hyperref = false } ,
2028     nohyperref .value_forbidden:n = true ,
2029 }
2030 \AddToHook { begindocument }
```

```

2031 {
2032   \__zrefclever_if_package_loaded:nTF { hyperref }
2033   {
2034     \bool_if:NT \l__zrefclever_hyperlink_bool
2035     { \RequirePackage { zref-hyperref } }
2036   }
2037   {
2038     \bool_if:NT \l__zrefclever_hyperref_warn_bool
2039     { \msg_warning:nn { zref-clever } { missing-hyperref } }
2040     \bool_set_false:N \l__zrefclever_hyperlink_bool
2041   }
2042   \keys_define:nn { zref-clever/reference }
2043   {
2044     hyperref .code:n =
2045     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2046     nohyperref .code:n =
2047     { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2048   }
2049 }

```

nameinlink option

```

2050 \str_new:N \l__zrefclever_nameinlink_str
2051 \keys_define:nn { zref-clever/reference }
2052 {
2053   nameinlink .choice: ,
2054   nameinlink / true .code:n =
2055   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2056   nameinlink / false .code:n =
2057   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2058   nameinlink / single .code:n =
2059   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2060   nameinlink / tsingle .code:n =
2061   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2062   nameinlink .initial:n = tsingle ,
2063   nameinlink .default:n = true ,
2064 }

```

preposinlink option (deprecated)

```

2065 \keys_define:nn { zref-clever/reference }
2066 {
2067   preposinlink .code:n =
2068   {
2069     % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2070     \msg_warning:nnnn { zref-clever } { option-deprecated }
2071     { preposinlink } { refbounds }
2072   } ,
2073 }

```

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`.

```

2074 \AddToHook { begindocument }
2075 {
2076   \__zrefclever_if_package_loaded:nTF { babel }
2077   {
2078     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
2079     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2080   }
2081   {
2082     \__zrefclever_if_package_loaded:nTF { polyglossia }
2083     {
2084       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2085       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2086     }
2087     {
2088       \tl_set:Nn \l__zrefclever_current_language_tl { english }
2089       \tl_set:Nn \l__zrefclever_main_language_tl { english }
2090     }
2091   }
2092 }
2093 \keys_define:nn { zref-clever/reference }
2094 {
2095   lang .code:n =
2096   {
2097     \AddToHook { begindocument }
2098     {
2099       \str_case:nnF {#1}
2100       {
2101         { current }
2102         {
2103           \tl_set:Nn \l__zrefclever_ref_language_tl
2104             { \l__zrefclever_current_language_tl }
2105         }
2106       }
2107       { main }
2108       {
2109         \tl_set:Nn \l__zrefclever_ref_language_tl
2110           { \l__zrefclever_main_language_tl }

```

```

2111     }
2112   }
2113   {
2114     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2115     \__zrefclever_language_if_declared:nF {#1}
2116     {
2117       \msg_warning:nnn { zref-clever }
2118       { unknown-language-opt } {#1}
2119     }
2120   }
2121   \__zrefclever_provide_langfile:x
2122   { \l__zrefclever_ref_language_tl }
2123 }
2124 } ,
2125 lang .initial:n = current ,
2126 lang .value_required:n = true ,
2127 }
2128 \AddToHook { begindocument / before }
2129 {
2130   \AddToHook { begindocument }
2131   {

```

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.

```

2132   \keys_define:nn { zref-clever/reference }
2133   {
2134     lang .code:n =
2135     {
2136       \str_case:nnF {#1}
2137       {
2138         { current }
2139         {
2140           \tl_set:Nn \l__zrefclever_ref_language_tl
2141           { \l__zrefclever_current_language_tl }
2142         }
2143       }
2144       { main }
2145       {
2146         \tl_set:Nn \l__zrefclever_ref_language_tl
2147         { \l__zrefclever_main_language_tl }
2148       }
2149     }
2150     {
2151       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2152       \__zrefclever_language_if_declared:nF {#1}
2153       {
2154         \msg_warning:nnn { zref-clever }
2155         { unknown-language-opt } {#1}
2156       }
2157     }
2158   } ,
2159 }
2160 }

```



```
2161 }
```

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.

```
2162 \tl_new:N \l__zrefclever_ref_decl_case_tl
2163 \keys_define:nn { zref-clever/reference }
2164 {
2165     d .code:n =
2166     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2167 }
2168 \AddToHook { begindocument }
2169 {
2170     \keys_define:nn { zref-clever/reference }
2171     {
```

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

```
2172     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2173     d .value_required:n = true ,
2174 }
2175 }
```

nudge & co. options

```
2176 \bool_new:N \l__zrefclever_nudge_enabled_bool
2177 \bool_new:N \l__zrefclever_nudge_multitype_bool
2178 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2179 \bool_new:N \l__zrefclever_nudge_singular_bool
2180 \bool_new:N \l__zrefclever_nudge_gender_bool
2181 \tl_new:N \l__zrefclever_ref_gender_tl
2182 \keys_define:nn { zref-clever/reference }
2183 {
2184     nudge .choice: ,
2185     nudge / true .code:n =
2186     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2187     nudge / false .code:n =
2188     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2189     nudge / ifdraft .code:n =
2190     {
2191         \ifdraft
2192         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2193         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2194     } ,
2195     nudge / iffina .code:n =
2196     {
2197         \ifoptionfinal
2198         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2199     }
```

```

2199         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2200     } ,
2201     nudge .initial:n = false ,
2202     nudge .default:n = true ,
2203     nonnudge .meta:n = { nudge = false } ,
2204     nonnudge .value_forbidden:n = true ,
2205     nudgeif .code:n =
2206     {
2207         \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2208         \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2209         \bool_set_false:N \l__zrefclever_nudge_gender_bool
2210         \clist_map_inline:nn {#1}
2211         {
2212             \str_case:nnF {##1}
2213             {
2214                 { multitype }
2215                 { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2216                 { comptosing }
2217                 { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2218                 { gender }
2219                 { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2220                 { all }
2221                 {
2222                     \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2223                     \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2224                     \bool_set_true:N \l__zrefclever_nudge_gender_bool
2225                 }
2226             }
2227         {
2228             \msg_warning:nnn { zref-clever }
2229             { nudgeif-unknown-value } {##1}
2230         }
2231     }
2232 } ,
2233 nudgeif .value_required:n = true ,
2234 nudgeif .initial:n = all ,
2235 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2236 sg .initial:n = false ,
2237 sg .default:n = true ,
2238 g .code:n =
2239 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2240 }
2241 \AddToHook { begindocument }
2242 {
2243     \keys_define:nn { zref-clever/reference }
2244     {

```

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

```

2245         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2246         g .value_required:n = true ,
2247     }
2248 }

```

font option

```
2249 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2250 \keys_define:nn { zref-clever/reference }
2251 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```
2252 \keys_define:nn { zref-clever/reference }
2253 {
2254   titleref .code:n =
2255   {
2256     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2257     \msg_warning:nxx { zref-clever }{ option-deprecated } { titleref }
2258     { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2259   } ,
2260 }
```

vario option

```
2261 \keys_define:nn { zref-clever/reference }
2262 {
2263   vario .code:n =
2264   {
2265     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2266     \msg_warning:nxx { zref-clever }{ option-deprecated } { vario }
2267     { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2268   } ,
2269 }
```

note option

```
2270 \tl_new:N \l__zrefclever_zcref_note_tl
2271 \keys_define:nn { zref-clever/reference }
2272 {
2273   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2274   note .value_required:n = true ,
2275 }
```

check option

Integration with zref-check.

```
2276 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2277 \bool_new:N \l__zrefclever_zcref_with_check_bool
2278 \keys_define:nn { zref-clever/reference }
2279 {
2280   check .code:n =
2281   { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2282 }
2283 \AddToHook { begindocument }
2284 {
2285   \__zrefclever_if_package_loaded:nTF { zref-check }
2286   {
2287     \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2288     {
2289       \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2290       \keys_define:nn { zref-clever/reference }
2291       {
```

```

2292         check .code:n =
2293         {
2294             \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2295             \keys_set:nn { zref-check / zcheck } {#1}
2296         } ,
2297         check .value_required:n = true ,
2298     }
2299 }
2300 {
2301     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2302     \keys_define:nn { zref-clever/reference }
2303     {
2304         check .code:n =
2305         {
2306             \msg_warning:nnn { zref-clever }
2307             { zref-check-too-old } { 2021-09-16-v0.2.1 }
2308         } ,
2309     }
2310 }
2311 }
2312 {
2313     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2314     \keys_define:nn { zref-clever/reference }
2315     {
2316         check .code:n =
2317         { \msg_warning:nnn { zref-clever } { missing-zref-check } } ,
2318     }
2319 }
2320 }

```

reftype option

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

```

2321 \tl_new:N \l__zrefclever_reftype_override_tl
2322 \keys_define:nn { zref-clever/label }
2323 {
2324     reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2325     reftype .default:n = {} ,
2326     reftype .initial:n = {} ,
2327 }

```

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`.

```

2328 \prop_new:N \l__zrefclever_counter_type_prop
2329 \keys_define:nn { zref-clever/label }
2330 {
2331     countertype .code:n =
2332     {

```

```

2333     \keyval_parse:nnn
2334     {
2335         \msg_warning:nnnn { zref-clever }
2336         { key-requires-value } { countertype }
2337     }
2338     {
2339         \__zrefclever_prop_put_non_empty:Nnn
2340         \l__zrefclever_counter_type_prop
2341     }
2342     {#1}
2343 } ,
2344 countertype .value_required:n = true ,
2345 countertype .initial:n =
2346 {
2347     subsection      = section ,
2348     subsubsection    = section ,
2349     subparagraph     = paragraph ,
2350     enumi            = item ,
2351     enumii           = item ,
2352     enumiii          = item ,
2353     enumiv           = item ,
2354     mpfootnote       = footnote ,
2355 } ,
2356 }

```

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.

```

2357 \seq_new:N \l__zrefclever_counter_resetters_seq
2358 \keys_define:nn { zref-clever/label }
2359 {

```

```

2360 counterresetters .code:n =
2361 {
2362   \clist_map_inline:nn {#1}
2363   {
2364     \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2365     {
2366       \seq_put_right:Nn
2367       \l__zrefclever_counter_resetters_seq {##1}
2368     }
2369   }
2370 },
2371 counterresetters .initial:n =
2372 {
2373   part ,
2374   chapter ,
2375   section ,
2376   subsection ,
2377   subsubsection ,
2378   paragraph ,
2379   subparagraph ,
2380 },
2381 counterresetters .value_required:n = true ,
2382 }

```

counterresetby option

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

```

2383 \prop_new:N \l__zrefclever_counter_resetby_prop
2384 \keys_define:nn { zref-clever/label }
2385 {
2386   counterresetby .code:n =
2387   {
2388     \keyval_parse:nnn
2389     {
2390       \msg_warning:nnn { zref-clever }
2391       { key-requires-value } { counterresetby }
2392     }
2393     {
2394       \__zrefclever_prop_put_non_empty:Nnn
2395       \l__zrefclever_counter_resetby_prop
2396       {#1}
2397     }
2398   },
2399   counterresetby .value_required:n = true ,
2400   counterresetby .initial:n =
2401   {

```

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.

```
2402         enumii = enumi ,
2403         enumiii = enumii ,
2404         enumiv = enumiii ,
2405     } ,
2406 }
```

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.

```
2407 \tl_new:N \l__zrefclever_current_counter_tl
2408 \keys_define:nn { zref-clever/label }
2409 {
2410     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2411     currentcounter .default:n = \@currentcounter ,
2412     currentcounter .initial:n = \@currentcounter ,
2413 }
```

labelhook option

```
2414 \bool_new:N \l__zrefclever_labelhook_bool
2415 \keys_define:nn { zref-clever/label }
2416 {
2417     labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2418     labelhook .initial:n = true ,
2419     labelhook .default:n = true ,
2420 }
```

We *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that’s precisely the case inside the `amsmath`’s `multline` environment (and possibly elsewhere?). See <https://tex.stackexchange.com/a/402297> and <https://github.com/ho-tex/zref/issues/4>.

```
2421 \AddToHookWithArguments { label }
2422 {
2423     \bool_if:NT \l__zrefclever_labelhook_bool
2424     { \zref@wrapper@babel \zref@label {#1} }
2425 }
```

nocompat option

```
2426 \bool_new:N \g__zrefclever_nocompat_bool
2427 \seq_new:N \g__zrefclever_nocompat_modules_seq
2428 \keys_define:nn { zref-clever/reference }
2429 {
2430     nocompat .code:n =
2431     {
2432         \tl_if_empty:nTF {#1}
2433         { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2434         {
2435             \clist_map_inline:nn {#1}
```

```

2436         {
2437             \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2438             {
2439                 \seq_gput_right:Nn
2440                 \g__zrefclever_nocompat_modules_seq {##1}
2441             }
2442         }
2443     } ,
2444 } ,
2445 }
2446 \AddToHook { begindocument }
2447 {
2448     \keys_define:nn { zref-clever/reference }
2449     {
2450         nocompat .code:n =
2451         {
2452             \msg_warning:nnn { zref-clever }
2453             { option-preamble-only } { nocompat }
2454         }
2455     }
2456 }
2457 \AtEndOfPackage
2458 {
2459     \AddToHook { begindocument }
2460     {
2461         \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2462         { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2463     }
2464 }

```

`_zrefclever_compat_module:nn`

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

```

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

2465 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2466 {
2467     \AddToHook { begindocument }
2468     {
2469         \bool_if:NF \g__zrefclever_nocompat_bool
2470         { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2471         \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2472     }
2473 }

```

(End of definition for `__zrefclever_compat_module:nn`.)

Reference options

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

```

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

```

```

2523 \g__zrefclever_rf_opts_seq_refbounds_seq
2524 {
2525   \keys_define:nn { zref-clever/reference }
2526   {
2527     #1 .default:o = \c_novalue_tl ,
2528     #1 .code:n =
2529     {
2530       \tl_if_novalue:nTF {##1}
2531       {
2532         \__zrefclever_opt_seq_unset:c
2533         { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2534       }
2535       {
2536         \seq_clear:N \l_tmpa_seq
2537         \__zrefclever_opt_seq_set_clist_split:Nn
2538         \l_tmpa_seq {##1}
2539         \bool_lazy_or:nnTF
2540         { \tl_if_empty_p:n {##1} }
2541         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2542         {
2543           \__zrefclever_opt_seq_set_eq:cN
2544           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2545           \l_tmpa_seq
2546         }
2547         {
2548           \msg_warning:nxxx { zref-clever }
2549           { refbounds-must-be-four }
2550           {#1} { \seq_count:N \l_tmpa_seq }
2551         }
2552       }
2553     } ,
2554   }
2555 }
2556 \seq_map_inline:Nn
2557 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2558 {
2559   \keys_define:nn { zref-clever/reference }
2560   {
2561     #1 .choice: ,
2562     #1 / true .code:n =
2563     {
2564       \__zrefclever_opt_bool_set_true:c
2565       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2566     } ,
2567     #1 / false .code:n =
2568     {
2569       \__zrefclever_opt_bool_set_false:c
2570       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2571     } ,
2572     #1 / unset .code:n =
2573     {
2574       \__zrefclever_opt_bool_unset:c
2575       { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2576     } ,

```

```

2577         #1 .default:n = true ,
2578         no #1 .meta:n = { #1 = false } ,
2579         no #1 .value_forbidden:n = true ,
2580     }
2581 }

```

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.

```

2582 \keys_define:nn { }
2583 {
2584     zref-clever/zcsetup .inherit:n =
2585     {
2586         zref-clever/label ,
2587         zref-clever/reference ,
2588     }
2589 }

```

`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.

```

2590 \bool_lazy_and:nnT
2591 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2592 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2593 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

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

```

2594 \NewDocumentCommand \zcsetup { m }
2595 { \__zrefclever_zcsetup:n {#1} }

```

(End of definition for `\zcsetup`.)

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

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

```

2596 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2597 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2598 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End of definition for `__zrefclever_zcsetup:n`.)

5.2 \zcRefTypeSetup

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

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

2599 \NewDocumentCommand \zcRefTypeSetup { m m }
2600 {
2601   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2602   \keys_set:nn { zref-clever/typesetup } {#2}
2603   \tl_clear:N \l__zrefclever_setup_type_tl
2604 }

(End of definition for \zcRefTypeSetup.)

2605 \seq_map_inline:Nn
2606   \g__zrefclever_rf_opts_tl_not_type_specific_seq
2607   {
2608     \keys_define:nn { zref-clever/typesetup }
2609     {
2610       #1 .code:n =
2611       {
2612         \msg_warning:nnn { zref-clever }
2613         { option-not-type-specific } {#1}
2614       } ,
2615     }
2616   }
2617 \seq_map_inline:Nn
2618   \g__zrefclever_rf_opts_tl_typesetup_seq
2619   {
2620     \keys_define:nn { zref-clever/typesetup }
2621     {
2622       #1 .default:o = \c_novalue_tl ,
2623       #1 .code:n =
2624       {
2625         \tl_if_novalue:nTF {##1}
2626         {
2627           \__zrefclever_opt_tl_unset:c
2628           {
2629             \__zrefclever_opt_varname_type:enn
2630             { \l__zrefclever_setup_type_tl } {#1} { tl }
2631           }
2632         }
2633         {
2634           \__zrefclever_opt_tl_set:cn
2635           {
2636             \__zrefclever_opt_varname_type:enn
2637             { \l__zrefclever_setup_type_tl } {#1} { tl }
2638           }
2639           {##1}

```

```

2640     }
2641   } ,
2642 }
2643 }
2644 \keys_define:nn { zref-clever/typesetup }
2645 {
2646   endrange .code:n =
2647   {
2648     \str_case:nnF {#1}
2649     {
2650       { ref }
2651       {
2652         \__zrefclever_opt_tl_clear:c
2653         {
2654           \__zrefclever_opt_varname_type:enn
2655             { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2656         }
2657         \__zrefclever_opt_tl_clear:c
2658         {
2659           \__zrefclever_opt_varname_type:enn
2660             { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2661         }
2662       }
2663     }
2664     { stripprefix }
2665     {
2666       \__zrefclever_opt_tl_set:cn
2667       {
2668         \__zrefclever_opt_varname_type:enn
2669           { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2670       }
2671       { __zrefclever_get_endrange_stripprefix }
2672       \__zrefclever_opt_tl_clear:c
2673       {
2674         \__zrefclever_opt_varname_type:enn
2675           { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2676       }
2677     }
2678     { pagecomp }
2679     {
2680       \__zrefclever_opt_tl_set:cn
2681       {
2682         \__zrefclever_opt_varname_type:enn
2683           { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2684       }
2685       { __zrefclever_get_endrange_pagecomp }
2686       \__zrefclever_opt_tl_clear:c
2687       {
2688         \__zrefclever_opt_varname_type:enn
2689           { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2690       }
2691     }
2692   }
2693 }

```

```

2694 { pagecomp2 }
2695 {
2696   \_zrefclever_opt_tl_set:cn
2697   {
2698     \_zrefclever_opt_varname_type:enn
2699     { \l\_zrefclever_setup_type_tl } { endrangefunc } { tl }
2700   }
2701   { \_zrefclever_get_endrange_pagecomptwo }
2702   \_zrefclever_opt_tl_clear:c
2703   {
2704     \_zrefclever_opt_varname_type:enn
2705     { \l\_zrefclever_setup_type_tl } { endrangeprop } { tl }
2706   }
2707 }
2708
2709 { unset }
2710 {
2711   \_zrefclever_opt_tl_unset:c
2712   {
2713     \_zrefclever_opt_varname_type:enn
2714     { \l\_zrefclever_setup_type_tl } { endrangefunc } { tl }
2715   }
2716   \_zrefclever_opt_tl_unset:c
2717   {
2718     \_zrefclever_opt_varname_type:enn
2719     { \l\_zrefclever_setup_type_tl } { endrangeprop } { tl }
2720   }
2721 }
2722 }
2723 {
2724   \tl_if_empty:nTF {#1}
2725   {
2726     \msg_warning:nnn { zref-clever }
2727     { endrange-property-undefined } {#1}
2728   }
2729   {
2730     \zref@ifpropundefined {#1}
2731     {
2732       \msg_warning:nnn { zref-clever }
2733       { endrange-property-undefined } {#1}
2734     }
2735     {
2736       \_zrefclever_opt_tl_set:cn
2737       {
2738         \_zrefclever_opt_varname_type:enn
2739         { \l\_zrefclever_setup_type_tl }
2740         { endrangefunc } { tl }
2741       }
2742       { \_zrefclever_get_endrange_property }
2743       \_zrefclever_opt_tl_set:cn
2744       {
2745         \_zrefclever_opt_varname_type:enn
2746         { \l\_zrefclever_setup_type_tl }
2747         { endrangeprop } { tl }

```

```

2748         }
2749         {#1}
2750     }
2751 }
2752 }
2753 },
2754     endrange .value_required:n = true ,
2755 }
2756 \keys_define:nn { zref-clever/typesetup }
2757 {
2758     refpre .code:n =
2759     {
2760         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2761         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2762         { refpre } { refbounds }
2763     } ,
2764     refpos .code:n =
2765     {
2766         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2767         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2768         { refpos } { refbounds }
2769     } ,
2770     preref .code:n =
2771     {
2772         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2773         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2774         { preref } { refbounds }
2775     } ,
2776     postref .code:n =
2777     {
2778         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2779         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2780         { postref } { refbounds }
2781     } ,
2782 }
2783 \seq_map_inline:Nn
2784   \g__zrefclever_rf_opts_seq_refbounds_seq
2785   {
2786     \keys_define:nn { zref-clever/typesetup }
2787     {
2788         #1 .default:o = \c_novalue_tl ,
2789         #1 .code:n =
2790         {
2791             \tl_if_novalue:nTF {##1}
2792             {
2793                 \__zrefclever_opt_seq_unset:c
2794                 {
2795                     \__zrefclever_opt_varname_type:enn
2796                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2797                 }
2798             }
2799             {
2800                 \seq_clear:N \l_tmpa_seq
2801                 \__zrefclever_opt_seq_set_clist_split:Nn

```

```

2802         \l_tmpa_seq {##1}
2803     \bool_lazy_or:nnTF
2804     { \tl_if_empty_p:n {##1} }
2805     { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2806     {
2807         \__zrefclever_opt_seq_set_eq:cN
2808         {
2809             \__zrefclever_opt_varname_type:enn
2810             { \l__zrefclever_setup_type_tl } {#1} { seq }
2811         }
2812         \l_tmpa_seq
2813     }
2814     {
2815         \msg_warning:nxxx { zref-clever }
2816         { refbounds-must-be-four }
2817         {#1} { \seq_count:N \l_tmpa_seq }
2818     }
2819 }
2820 } ,
2821 }
2822 }
2823 \seq_map_inline:Nn
2824 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2825 {
2826     \keys_define:nn { zref-clever/typesetup }
2827     {
2828         #1 .choice: ,
2829         #1 / true .code:n =
2830         {
2831             \__zrefclever_opt_bool_set_true:c
2832             {
2833                 \__zrefclever_opt_varname_type:enn
2834                 { \l__zrefclever_setup_type_tl }
2835                 {#1} { bool }
2836             }
2837         } ,
2838         #1 / false .code:n =
2839         {
2840             \__zrefclever_opt_bool_set_false:c
2841             {
2842                 \__zrefclever_opt_varname_type:enn
2843                 { \l__zrefclever_setup_type_tl }
2844                 {#1} { bool }
2845             }
2846         } ,
2847         #1 / unset .code:n =
2848         {
2849             \__zrefclever_opt_bool_unset:c
2850             {
2851                 \__zrefclever_opt_varname_type:enn
2852                 { \l__zrefclever_setup_type_tl }
2853                 {#1} { bool }
2854             }
2855         } ,

```



```

2856     #1 .default:n = true ,
2857     no #1 .meta:n = { #1 = false } ,
2858     no #1 .value_forbidden:n = true ,
2859   }
2860 }

```

5.3 \zcLanguageSetup

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

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}

2861 \NewDocumentCommand \zcLanguageSetup { m m }
2862 {
2863   \group_begin:
2864   \__zrefclever_language_if_declared:NTF {#1}
2865   {
2866     \tl_clear:N \l__zrefclever_setup_type_tl
2867     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2868     \__zrefclever_opt_seq_get:cNF
2869     {
2870       \__zrefclever_opt_varname_language:nnn
2871       {#1} { declension } { seq }
2872     }
2873     \l__zrefclever_lang_declension_seq
2874     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2875     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2876     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2877     {
2878       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2879       \l__zrefclever_lang_decl_case_tl
2880     }
2881     \__zrefclever_opt_seq_get:cNF
2882     {
2883       \__zrefclever_opt_varname_language:nnn
2884       {#1} { gender } { seq }
2885     }
2886     \l__zrefclever_lang_gender_seq
2887     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2888     \keys_set:nn { zref-clever/langsetup } {#2}
2889   }
2890   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2891   \group_end:
2892 }
2893 \@onlypreamble \zcLanguageSetup

```

(End of definition for \zcLanguageSetup.)

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

```

2894 \keys_define:nn { zref-clever/langsetup }
2895 {
2896   type .code:n =
2897   {
2898     \tl_if_empty:NTF {#1}
2899     { \tl_clear:N \l__zrefclever_setup_type_tl }
2900     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2901   } ,
2902
2903   case .code:n =
2904   {
2905     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2906     {
2907       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2908       { \l__zrefclever_setup_language_tl } {#1}
2909     }
2910     {
2911       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2912       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2913       {
2914         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2915         {#1} { \l__zrefclever_setup_language_tl }
2916         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2917         \l__zrefclever_lang_decl_case_tl
2918       }
2919     }
2920   } ,
2921   case .value_required:n = true ,
2922
2923   gender .value_required:n = true ,
2924   gender .code:n =
2925   {
2926     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2927     {
2928       \msg_warning:nxxx { zref-clever } { language-no-gender }
2929       { \l__zrefclever_setup_language_tl } { gender } {#1}
2930     }
2931     {
2932       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2933       {
2934         \msg_warning:nnn { zref-clever }
2935         { option-only-type-specific } { gender }
2936       }
2937       {
2938         \seq_clear:N \l_tmpa_seq
2939         \clist_map_inline:nn {#1}
2940         {
2941           \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2942           { \seq_put_right:Nn \l_tmpa_seq {##1} }
2943           {
2944             \msg_warning:nxxx { zref-clever }
2945             { gender-not-declared }

```

```

2946         { \l__zrefclever_setup_language_tl } {##1}
2947     }
2948 }
2949 \__zrefclever_opt_seq_gset_eq:cn
2950 {
2951     \__zrefclever_opt_varname_lang_type:enn
2952     { \l__zrefclever_setup_language_tl }
2953     { \l__zrefclever_setup_type_tl }
2954     { gender }
2955     { seq }
2956 }
2957 \l_tmpa_seq
2958 }
2959 }
2960 } ,
2961 }
2962 \seq_map_inline:Nn
2963   \g__zrefclever_rf_opts_tl_not_type_specific_seq
2964   {
2965     \keys_define:nn { zref-clever/langsetup }
2966     {
2967       #1 .value_required:n = true ,
2968       #1 .code:n =
2969       {
2970         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2971         {
2972           \__zrefclever_opt_tl_gset:cn
2973           {
2974             \__zrefclever_opt_varname_lang_default:enn
2975             { \l__zrefclever_setup_language_tl } {#1} { tl }
2976           }
2977           {##1}
2978         }
2979         {
2980           \msg_warning:nnn { zref-clever }
2981           { option-not-type-specific } {#1}
2982         }
2983       } ,
2984     }
2985   }
2986 \seq_map_inline:Nn
2987   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2988   {
2989     \keys_define:nn { zref-clever/langsetup }
2990     {
2991       #1 .value_required:n = true ,
2992       #1 .code:n =
2993       {
2994         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2995         {
2996           \__zrefclever_opt_tl_gset:cn
2997           {
2998             \__zrefclever_opt_varname_lang_default:enn
2999             { \l__zrefclever_setup_language_tl } {#1} { tl }

```

```

3000     }
3001     {##1}
3002   }
3003   {
3004     \__zrefclever_opt_tl_gset:cn
3005     {
3006       \__zrefclever_opt_varname_lang_type:enn
3007       { \l__zrefclever_setup_language_tl }
3008       { \l__zrefclever_setup_type_tl }
3009       {#1} { t1 }
3010     }
3011     {##1}
3012   }
3013   } ,
3014 }
3015 }
3016 \keys_define:nn { zref-clever/langsetup }
3017 {
3018   endrange .value_required:n = true ,
3019   endrange .code:n =
3020   {
3021     \str_case:nnF {#1}
3022     {
3023       { ref }
3024       {
3025         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3026         {
3027           \__zrefclever_opt_tl_gclear:c
3028           {
3029             \__zrefclever_opt_varname_lang_default:enn
3030             { \l__zrefclever_setup_language_tl }
3031             { endrangefunc } { t1 }
3032           }
3033           \__zrefclever_opt_tl_gclear:c
3034           {
3035             \__zrefclever_opt_varname_lang_default:enn
3036             { \l__zrefclever_setup_language_tl }
3037             { endrangeprop } { t1 }
3038           }
3039         }
3040       {
3041         \__zrefclever_opt_tl_gclear:c
3042         {
3043           \__zrefclever_opt_varname_lang_type:enn
3044           { \l__zrefclever_setup_language_tl }
3045           { \l__zrefclever_setup_type_tl }
3046           { endrangefunc } { t1 }
3047         }
3048         \__zrefclever_opt_tl_gclear:c
3049         {
3050           \__zrefclever_opt_varname_lang_type:enn
3051           { \l__zrefclever_setup_language_tl }
3052           { \l__zrefclever_setup_type_tl }
3053           { endrangeprop } { t1 }

```

```

3054     }
3055   }
3056 }
3057
3058 { stripprefix }
3059 {
3060   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3061   {
3062     \__zrefclever_opt_tl_gset:cn
3063     {
3064       \__zrefclever_opt_varname_lang_default:enn
3065       { \l__zrefclever_setup_language_tl }
3066       { endrangefunc } { t1 }
3067     }
3068     { __zrefclever_get_endrange_stripprefix }
3069     \__zrefclever_opt_tl_gclear:c
3070     {
3071       \__zrefclever_opt_varname_lang_default:enn
3072       { \l__zrefclever_setup_language_tl }
3073       { endrangeprop } { t1 }
3074     }
3075   }
3076   {
3077     \__zrefclever_opt_tl_gset:cn
3078     {
3079       \__zrefclever_opt_varname_lang_type:enn
3080       { \l__zrefclever_setup_language_tl }
3081       { \l__zrefclever_setup_type_tl }
3082       { endrangefunc } { t1 }
3083     }
3084     { __zrefclever_get_endrange_stripprefix }
3085     \__zrefclever_opt_tl_gclear:c
3086     {
3087       \__zrefclever_opt_varname_lang_type:enn
3088       { \l__zrefclever_setup_language_tl }
3089       { \l__zrefclever_setup_type_tl }
3090       { endrangeprop } { t1 }
3091     }
3092   }
3093 }
3094
3095 { pagecomp }
3096 {
3097   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3098   {
3099     \__zrefclever_opt_tl_gset:cn
3100     {
3101       \__zrefclever_opt_varname_lang_default:enn
3102       { \l__zrefclever_setup_language_tl }
3103       { endrangefunc } { t1 }
3104     }
3105     { __zrefclever_get_endrange_pagecomp }
3106     \__zrefclever_opt_tl_gclear:c
3107     {

```

```

3108         \_zrefclever_opt_varname_lang_default:enn
3109         { \l_zrefclever_setup_language_tl }
3110         { endrangeprop } { tl }
3111     }
3112 }
3113 {
3114     \_zrefclever_opt_tl_gset:cn
3115     {
3116         \_zrefclever_opt_varname_lang_type:eenn
3117         { \l_zrefclever_setup_language_tl }
3118         { \l_zrefclever_setup_type_tl }
3119         { endrangefunc } { tl }
3120     }
3121     { __zrefclever_get_endrange_pagecomp }
3122     \_zrefclever_opt_tl_gclear:c
3123     {
3124         \_zrefclever_opt_varname_lang_type:eenn
3125         { \l_zrefclever_setup_language_tl }
3126         { \l_zrefclever_setup_type_tl }
3127         { endrangeprop } { tl }
3128     }
3129 }
3130 }
3131
3132 { pagecomp2 }
3133 {
3134     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3135     {
3136         \_zrefclever_opt_tl_gset:cn
3137         {
3138             \_zrefclever_opt_varname_lang_default:enn
3139             { \l_zrefclever_setup_language_tl }
3140             { endrangefunc } { tl }
3141         }
3142         { __zrefclever_get_endrange_pagecomptwo }
3143         \_zrefclever_opt_tl_gclear:c
3144         {
3145             \_zrefclever_opt_varname_lang_default:enn
3146             { \l_zrefclever_setup_language_tl }
3147             { endrangeprop } { tl }
3148         }
3149     }
3150     {
3151         \_zrefclever_opt_tl_gset:cn
3152         {
3153             \_zrefclever_opt_varname_lang_type:eenn
3154             { \l_zrefclever_setup_language_tl }
3155             { \l_zrefclever_setup_type_tl }
3156             { endrangefunc } { tl }
3157         }
3158         { __zrefclever_get_endrange_pagecomptwo }
3159         \_zrefclever_opt_tl_gclear:c
3160         {
3161             \_zrefclever_opt_varname_lang_type:eenn

```

```

3162         { \l__zrefclever_setup_language_tl }
3163         { \l__zrefclever_setup_type_tl }
3164         { endrangeprop } { tl }
3165     }
3166 }
3167 }
3168 }
3169 {
3170     \tl_if_empty:nTF {#1}
3171     {
3172         \msg_warning:nnn { zref-clever }
3173         { endrange-property-undefined } {#1}
3174     }
3175     {
3176         \zref@ifpropundefined {#1}
3177         {
3178             \msg_warning:nnn { zref-clever }
3179             { endrange-property-undefined } {#1}
3180         }
3181         {
3182             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3183             {
3184                 \__zrefclever_opt_tl_gset:cn
3185                 {
3186                     \__zrefclever_opt_varname_lang_default:enn
3187                     { \l__zrefclever_setup_language_tl }
3188                     { endrangefunc } { tl }
3189                 }
3190                 { __zrefclever_get_endrange_property }
3191                 \__zrefclever_opt_tl_gset:cn
3192                 {
3193                     \__zrefclever_opt_varname_lang_default:enn
3194                     { \l__zrefclever_setup_language_tl }
3195                     { endrangeprop } { tl }
3196                 }
3197                 {#1}
3198             }
3199             {
3200                 \__zrefclever_opt_tl_gset:cn
3201                 {
3202                     \__zrefclever_opt_varname_lang_type:enn
3203                     { \l__zrefclever_setup_language_tl }
3204                     { \l__zrefclever_setup_type_tl }
3205                     { endrangefunc } { tl }
3206                 }
3207                 { __zrefclever_get_endrange_property }
3208                 \__zrefclever_opt_tl_gset:cn
3209                 {
3210                     \__zrefclever_opt_varname_lang_type:enn
3211                     { \l__zrefclever_setup_language_tl }
3212                     { \l__zrefclever_setup_type_tl }
3213                     { endrangeprop } { tl }
3214                 }
3215                 {#1}

```

```

3216     }
3217   }
3218 }
3219 }
3220 },
3221 }
3222 \keys_define:nn { zref-clever/langsetup }
3223 {
3224   refpre .code:n =
3225   {
3226     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3227     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3228     { refpre } { refbounds }
3229   },
3230   refpos .code:n =
3231   {
3232     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3233     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3234     { refpos } { refbounds }
3235   },
3236   preref .code:n =
3237   {
3238     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3239     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3240     { preref } { refbounds }
3241   },
3242   postref .code:n =
3243   {
3244     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3245     \msg_warning:nnnn { zref-clever }{ option-deprecated }
3246     { postref } { refbounds }
3247   },
3248 }
3249 \seq_map_inline:Nn
3250 \g__zrefclever_rf_opts_tl_type_names_seq
3251 {
3252   \keys_define:nn { zref-clever/langsetup }
3253   {
3254     #1 .value_required:n = true ,
3255     #1 .code:n =
3256     {
3257       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3258       {
3259         \msg_warning:nnn { zref-clever }
3260         { option-only-type-specific } {#1}
3261       }
3262       {
3263         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3264         {
3265           \__zrefclever_opt_tl_gset:cn
3266           {
3267             \__zrefclever_opt_varname_lang_type:eenn
3268             { \l__zrefclever_setup_language_tl }
3269             { \l__zrefclever_setup_type_tl }

```



```

3270         {#1} { t1 }
3271     }
3272     {##1}
3273 }
3274 {
3275     \__zrefclever_opt_tl_gset:cn
3276     {
3277         \__zrefclever_opt_varname_lang_type:eeen
3278         { \l__zrefclever_setup_language_tl }
3279         { \l__zrefclever_setup_type_tl }
3280         { \l__zrefclever_lang_decl_case_tl - #1 }
3281         { t1 }
3282     }
3283     {##1}
3284 }
3285 }
3286 } ,
3287 }
3288 }
3289 \seq_map_inline:Nn
3290 \g__zrefclever_rf_opts_seq_refbounds_seq
3291 {
3292     \keys_define:nn { zref-clever/langsetup }
3293     {
3294         #1 .value_required:n = true ,
3295         #1 .code:n =
3296         {
3297             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3298             {
3299                 \seq_gclear:N \g_tmpa_seq
3300                 \__zrefclever_opt_seq_gset_clist_split:Nn
3301                 \g_tmpa_seq {##1}
3302                 \bool_lazy_or:nnTF
3303                 { \tl_if_empty_p:n {##1} }
3304                 {
3305                     \int_compare_p:nNn
3306                     { \seq_count:N \g_tmpa_seq } = { 4 }
3307                 }
3308                 {
3309                     \__zrefclever_opt_seq_gset_eq:cN
3310                     {
3311                         \__zrefclever_opt_varname_lang_default:enn
3312                         { \l__zrefclever_setup_language_tl }
3313                         {#1} { seq }
3314                     }
3315                     \g_tmpa_seq
3316                 }
3317                 {
3318                     \msg_warning:nxxx { zref-clever }
3319                     { refbounds-must-be-four }
3320                     {#1} { \seq_count:N \g_tmpa_seq }
3321                 }
3322             }
3323         }

```

```

3324         \seq_gclear:N \g_tmpa_seq
3325         \__zrefclever_opt_seq_gset_clist_split:Nn
3326         \g_tmpa_seq {##1}
3327         \bool_lazy_or:nnTF
3328         { \tl_if_empty_p:n {##1} }
3329         {
3330             \int_compare_p:nNn
3331             { \seq_count:N \g_tmpa_seq } = { 4 }
3332         }
3333         {
3334             \__zrefclever_opt_seq_gset_eq:cN
3335             {
3336                 \__zrefclever_opt_varname_lang_type:eenn
3337                 { \l__zrefclever_setup_language_tl }
3338                 { \l__zrefclever_setup_type_tl } {#1} { seq }
3339             }
3340             \g_tmpa_seq
3341         }
3342         {
3343             \msg_warning:nxxx { zref-clever }
3344             { refbounds-must-be-four }
3345             {#1} { \seq_count:N \g_tmpa_seq }
3346         }
3347     } ,
3348 } ,
3349 }
3350 }
3351 \seq_map_inline:Nn
3352 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3353 {
3354     \keys_define:nn { zref-clever/langsetup }
3355     {
3356         #1 .choice: ,
3357         #1 / true .code:n =
3358         {
3359             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3360             {
3361                 \__zrefclever_opt_bool_gset_true:c
3362                 {
3363                     \__zrefclever_opt_varname_lang_default:enn
3364                     { \l__zrefclever_setup_language_tl }
3365                     {#1} { bool }
3366                 }
3367             }
3368             {
3369                 \__zrefclever_opt_bool_gset_true:c
3370                 {
3371                     \__zrefclever_opt_varname_lang_type:eenn
3372                     { \l__zrefclever_setup_language_tl }
3373                     { \l__zrefclever_setup_type_tl }
3374                     {#1} { bool }
3375                 }
3376             }
3377         } ,

```

```

3378     #1 / false .code:n =
3379     {
3380         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3381         {
3382             \__zrefclever_opt_bool_gset_false:c
3383             {
3384                 \__zrefclever_opt_varname_lang_default:enn
3385                 { \l__zrefclever_setup_language_tl }
3386                 {#1} { bool }
3387             }
3388         }
3389         {
3390             \__zrefclever_opt_bool_gset_false:c
3391             {
3392                 \__zrefclever_opt_varname_lang_type:enn
3393                 { \l__zrefclever_setup_language_tl }
3394                 { \l__zrefclever_setup_type_tl }
3395                 {#1} { bool }
3396             }
3397         }
3398     } ,
3399     #1 .default:n = true ,
3400     no #1 .meta:n = { #1 = false } ,
3401     no #1 .value_forbidden:n = true ,
3402 }
3403 }

```

6 User interface

6.1 \zcref

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

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

```

3404 \NewDocumentCommand \zcref { s O { } m }
3405 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End of definition for \zcref.)

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

```
\__zrefclever_zcref:nnnn {<labels>} {<*>} {<options>}
```

```

3406 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3407 {
3408     \group_begin:

```

Set options.

```
3409     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

3410     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3411     \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.

```
3412 \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3413 \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3414 \bool_lazy_and:nnT
3415 { \l__zrefclever_zrefcheck_available_bool }
3416 { \l__zrefclever_zcref_with_check_bool }
3417 { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3418 \bool_lazy_or:nnT
3419 { \l__zrefclever_typeset_sort_bool }
3420 { \l__zrefclever_typeset_range_bool }
3421 { \__zrefclever_sort_labels: }
```

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

```
3422 \group_begin:
3423 \l__zrefclever_ref_typeset_font_tl
3424 \__zrefclever_typeset_refs:
3425 \group_end:
```

Typeset note.

```
3426 \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3427 {
3428   \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3429   { \l__zrefclever_label_type_a_tl }
3430   { \l__zrefclever_ref_language_tl }
3431   \l_tmpa_tl
3432   \l_tmpa_tl
3433   \l__zrefclever_zcref_note_tl
3434 }
```

Integration with `zref-check`.

```
3435 \bool_lazy_and:nnT
3436 { \l__zrefclever_zrefcheck_available_bool }
3437 { \l__zrefclever_zcref_with_check_bool }
3438 {
3439   \zrefcheck_zcref_end_label_maybe:
3440   \zrefcheck_zcref_run_checks_on_labels:n
3441   { \l__zrefclever_zcref_labels_seq }
3442 }
```

Integration with `mathtools`.

```
3443 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3444 {
3445   \__zrefclever_mathtools_showonlyrefs:n
3446   { \l__zrefclever_zcref_labels_seq }
3447 }
3448 \group_end:
3449 }
```

(End of definition for `_zrefclever_zcref:nnnn`.)

`\l_zrefclever_zcref_labels_seq`
`\l_zrefclever_link_star_bool`

```
3450 \seq_new:N \l__zrefclever_zcref_labels_seq
3451 \bool_new:N \l__zrefclever_link_star_bool
```

(End of definition for `\l__zrefclever_zcref_labels_seq` and `\l__zrefclever_link_star_bool`.)

6.2 `\zcpageref`

`\zcpageref` A `\pageref` equivalent of `\zcref`.

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

```
3452 \NewDocumentCommand \zcpageref { s O { } m }
3453 {
3454   \group_begin:
3455   \IfBooleanT {#1}
3456     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3457   \zcref [#2, ref = page] {#3}
3458   \group_end:
3459 }
```

(End of definition for `\zcpageref`.)

7 Sorting

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

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

`\l_zrefclever_label_type_a_tl`
`\l_zrefclever_label_type_b_tl`
`\l_zrefclever_label_enclval_a_tl`
`\l_zrefclever_label_enclval_b_tl`
`\l_zrefclever_label_extdoc_a_tl`
`\l_zrefclever_label_extdoc_b_tl`

```
3460 \tl_new:N \l__zrefclever_label_type_a_tl
3461 \tl_new:N \l__zrefclever_label_type_b_tl
3462 \tl_new:N \l__zrefclever_label_enclval_a_tl
3463 \tl_new:N \l__zrefclever_label_enclval_b_tl
3464 \tl_new:N \l__zrefclever_label_extdoc_a_tl
3465 \tl_new:N \l__zrefclever_label_extdoc_b_tl
```

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

`\l_zrefclever_sort_decided_bool`

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

```
3466 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End of definition for \l__zrefclever_sort_decided_bool.)

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

```
3467 \int_new:N \l__zrefclever_sort_prior_a_int
3468 \int_new:N \l__zrefclever_sort_prior_b_int
```

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

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

```
3469 \seq_new:N \l__zrefclever_label_types_seq
```

(End of definition for \l__zrefclever_label_types_seq.)

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

```
3470 \cs_new_protected:Npn \__zrefclever_sort_labels:
3471 {
```

Store label types sequence.

```
3472   \seq_clear:N \l__zrefclever_label_types_seq
3473   \tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }
3474   {
3475     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3476     \__zrefclever_label_type_put_new_right:n
3477   }
```

Sort.

```
3478   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3479   {
3480     \zref@ifrefundefined {##1}
3481     {
3482       \zref@ifrefundefined {##2}
3483       {
3484         % Neither label is defined.
3485         \sort_return_same:
3486       }
3487       {
3488         % The second label is defined, but the first isn't, leave the
3489         % undefined first (to be more visible).
3490         \sort_return_same:
3491       }
3492     }
3493     {
3494       \zref@ifrefundefined {##2}
3495       {
3496         % The first label is defined, but the second isn't, bring the
```

```

3497         % second forward.
3498         \sort_return_swapped:
3499     }
3500     {
3501         % The interesting case: both labels are defined.  References
3502         % to the "default" property or to the "page" are quite
3503         % different with regard to sorting, so we branch them here to
3504         % specialized functions.
3505         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3506             { \__zrefclever_sort_page:nn {##1} {##2} }
3507             { \__zrefclever_sort_default:nn {##1} {##2} }
3508     }
3509 }
3510 }
3511 }

```

(End of definition for __zrefclever_sort_labels:.)

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

```

\__zrefclever_label_type_put_new_right:n {\label}

3512 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3513 {
3514     \__zrefclever_extract_default:Nnnn
3515     \l__zrefclever_label_type_a_tl {#1} {zc@type} {}
3516     \seq_if_in:NVF \l__zrefclever_label_types_seq
3517     \l__zrefclever_label_type_a_tl
3518     {
3519         \seq_put_right:NV \l__zrefclever_label_types_seq
3520         \l__zrefclever_label_type_a_tl
3521     }
3522 }

```

(End of definition for __zrefclever_label_type_put_new_right:n.)

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

```

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

3523 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3524 {
3525     \__zrefclever_extract_default:Nnnn
3526     \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}

```

```

3527 \__zrefclever_extract_default:Nnnn
3528 \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3529
3530 \tl_if_eq:NNTF
3531 \l__zrefclever_label_type_a_tl
3532 \l__zrefclever_label_type_b_tl
3533 { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3534 { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3535 }

```

(End of definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
3536 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3537 {
3538 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3539 {#1} {zc@enclval} { }
3540 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3541 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3542 {#2} {zc@enclval} { }
3543 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3544 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3545 {#1} {externaldocument} { }
3546 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3547 {#2} {externaldocument} { }
3548
3549 \bool_set_false:N \l__zrefclever_sort_decided_bool
3550
3551 % First we check if there's any "external document" difference (coming
3552 % from 'zref-xr') and, if so, sort based on that.
3553 \tl_if_eq:NNTF
3554 \l__zrefclever_label_extdoc_a_tl
3555 \l__zrefclever_label_extdoc_b_tl
3556 {
3557 \bool_if:nTF
3558 {
3559 \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3560 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3561 }
3562 {
3563 \bool_set_true:N \l__zrefclever_sort_decided_bool
3564 \sort_return_same:
3565 }
3566 {
3567 \bool_if:nTF
3568 {
3569 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3570 \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3571 }
3572 {
3573 \bool_set_true:N \l__zrefclever_sort_decided_bool
3574 \sort_return_swapped:
3575 }
3576 {

```



```

3577         \bool_set_true:N \l__zrefclever_sort_decided_bool
3578         % Two different "external documents": last resort, sort by the
3579         % document name itself.
3580         \str_compare:eNeTF
3581         { \l__zrefclever_label_extdoc_b_tl } <
3582         { \l__zrefclever_label_extdoc_a_tl }
3583         { \sort_return_swapped: }
3584         { \sort_return_same: }
3585     }
3586 }
3587 }
3588
3589 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3590 {
3591     \bool_if:nTF
3592     {
3593         % Both are empty: neither label has any (further) "enclosing
3594         % counters" (left).
3595         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3596         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3597     }
3598     {
3599         \bool_set_true:N \l__zrefclever_sort_decided_bool
3600         \int_compare:nNnTF
3601         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3602         >
3603         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3604         { \sort_return_swapped: }
3605         { \sort_return_same: }
3606     }
3607     {
3608         \bool_if:nTF
3609         {
3610             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
3611             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3612         }
3613         {
3614             \bool_set_true:N \l__zrefclever_sort_decided_bool
3615             \int_compare:nNnTF
3616             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3617             >
3618             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3619             { \sort_return_swapped: }
3620             { \sort_return_same: }
3621         }
3622         {
3623             \bool_if:nTF
3624             {
3625                 % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3626                 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3627             }
3628             {
3629                 \bool_set_true:N \l__zrefclever_sort_decided_bool
3630                 \int_compare:nNnTF

```

```

3631         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3632         <
3633         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3634         { \sort_return_same:    }
3635         { \sort_return_swapped: }
3636     }
3637     {
3638         % Neither is empty: we can compare the values of the
3639         % current enclosing counter in the loop, if they are
3640         % equal, we are still in the loop, if they are not, a
3641         % sorting decision can be made directly.
3642         \int_compare:nNnTF
3643         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3644         =
3645         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3646         {
3647             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3648             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3649             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3650             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3651         }
3652         {
3653             \bool_set_true:N \l__zrefclever_sort_decided_bool
3654             \int_compare:nNnTF
3655             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3656             >
3657             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3658             { \sort_return_swapped: }
3659             { \sort_return_same:    }
3660         }
3661     }
3662 }
3663 }
3664 }
3665 }

```

(End of definition for __zrefclever_sort_default_same_type:nn.)

```

__zrefclever_sort_default_different_types:nn
\__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
3666 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3667 {

```

Retrieve sort priorities for $\langle label\ a \rangle$ and $\langle label\ b \rangle$. $\l__zrefclever_typesort_seq$ was stored in reverse sequence, and we compute the sort priorities in the negative range, so that we can implicitly rely on ‘0’ being the “last value”.

```

3668     \int_zero:N \l__zrefclever_sort_prior_a_int
3669     \int_zero:N \l__zrefclever_sort_prior_b_int
3670     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3671     {
3672         \tl_if_eq:nnTF {##2} {{othertypes}}
3673         {
3674             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3675             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3676             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }

```

```

3677         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3678     }
3679     {
3680         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3681         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3682         {
3683             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3684             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3685         }
3686     }
3687 }

```

Then do the actual sorting.

```

3688 \bool_if:nTF
3689 {
3690     \int_compare_p:nNn
3691     { \l__zrefclever_sort_prior_a_int } <
3692     { \l__zrefclever_sort_prior_b_int }
3693 }
3694 { \sort_return_same: }
3695 {
3696     \bool_if:nTF
3697     {
3698         \int_compare_p:nNn
3699         { \l__zrefclever_sort_prior_a_int } >
3700         { \l__zrefclever_sort_prior_b_int }
3701     }
3702     { \sort_return_swapped: }
3703     {
3704         % Sort priorities are equal: the type that occurs first in
3705         % ‘labels’, as given by the user, is kept (or brought) forward.
3706         \seq_map_inline:Nn \l__zrefclever_label_types_seq
3707         {
3708             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3709             { \seq_map_break:n { \sort_return_same: } }
3710             {
3711                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3712                 { \seq_map_break:n { \sort_return_swapped: } }
3713             }
3714         }
3715     }
3716 }
3717 }

```

(End of definition for `__zrefclever_sort_default_different_types:nn`.)

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

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

```

3718 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3719 {
3720   \int_compare:nNnTF
3721     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3722     >
3723     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3724     { \sort_return_swapped: }
3725     { \sort_return_same: }
3726 }

```

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

8 Typesetting

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

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

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

tl being its type). And, since we may need up to two type blocks in storage before typesetting, we have two of these “queues”: `\l__zrefclever_typeset_queue_curr_tl` and `\l__zrefclever_typeset_queue_prev_tl`.

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

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

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

```
\l__zrefclever_typeset_labels_seq
\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool
3727 \seq_new:N \l__zrefclever_typeset_labels_seq
3728 \bool_new:N \l__zrefclever_typeset_last_bool
3729 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

```
\l__zrefclever_type_count_int
\l__zrefclever_label_count_int
\l__zrefclever_ref_count_int
3730 \int_new:N \l__zrefclever_type_count_int
3731 \int_new:N \l__zrefclever_label_count_int
3732 \int_new:N \l__zrefclever_ref_count_int
```

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

\l__zrefclever_label_a_tl \l__zrefclever_label_b_tl \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl \l__zrefclever_type_first_label_tl \l__zrefclever_type_first_label_type_tl	<p>Auxiliary variables for __zrefclever_typeset_refs: main “queue” control and storage.</p> <pre> 3733 \tl_new:N \l__zrefclever_label_a_tl 3734 \tl_new:N \l__zrefclever_label_b_tl 3735 \tl_new:N \l__zrefclever_typeset_queue_prev_tl 3736 \tl_new:N \l__zrefclever_typeset_queue_curr_tl 3737 \tl_new:N \l__zrefclever_type_first_label_tl 3738 \tl_new:N \l__zrefclever_type_first_label_type_tl </pre> <p>(End of definition for \l__zrefclever_label_a_tl and others.)</p>
\l__zrefclever_type_name_tl \l__zrefclever_name_in_link_bool \l__zrefclever_type_name_missing_bool \l__zrefclever_name_format_tl \l__zrefclever_name_format_fallback_tl \l__zrefclever_type_name_gender_seq	<p>Auxiliary variables for __zrefclever_typeset_refs: type name handling.</p> <pre> 3739 \tl_new:N \l__zrefclever_type_name_tl 3740 \bool_new:N \l__zrefclever_name_in_link_bool 3741 \bool_new:N \l__zrefclever_type_name_missing_bool 3742 \tl_new:N \l__zrefclever_name_format_tl 3743 \tl_new:N \l__zrefclever_name_format_fallback_tl 3744 \seq_new:N \l__zrefclever_type_name_gender_seq </pre> <p>(End of definition for \l__zrefclever_type_name_tl and others.)</p>
\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	<p>Auxiliary variables for __zrefclever_typeset_refs: range handling.</p> <pre> 3745 \int_new:N \l__zrefclever_range_count_int 3746 \int_new:N \l__zrefclever_range_same_count_int 3747 \tl_new:N \l__zrefclever_range_beg_label_tl 3748 \bool_new:N \l__zrefclever_range_beg_is_first_bool 3749 \tl_new:N \l__zrefclever_range_end_ref_tl 3750 \bool_new:N \l__zrefclever_next_maybe_range_bool 3751 \bool_new:N \l__zrefclever_next_is_same_bool </pre> <p>(End of definition for \l__zrefclever_range_count_int and others.)</p>
\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	<p>Auxiliary variables for __zrefclever_typeset_refs: separators, and font and other options.</p> <pre> 3752 \tl_new:N \l__zrefclever_tpairsep_tl 3753 \tl_new:N \l__zrefclever_tlistsep_tl 3754 \tl_new:N \l__zrefclever_tlastsep_tl 3755 \tl_new:N \l__zrefclever_namesep_tl 3756 \tl_new:N \l__zrefclever_pairsep_tl 3757 \tl_new:N \l__zrefclever_listsep_tl 3758 \tl_new:N \l__zrefclever_lastsep_tl 3759 \tl_new:N \l__zrefclever_rangesep_tl 3760 \tl_new:N \l__zrefclever_namefont_tl 3761 \tl_new:N \l__zrefclever_reffont_tl 3762 \tl_new:N \l__zrefclever_endrangefunc_tl 3763 \tl_new:N \l__zrefclever_endrangeprop_tl 3764 \bool_new:N \l__zrefclever_cap_bool 3765 \bool_new:N \l__zrefclever_abbrev_bool 3766 \bool_new:N \l__zrefclever_rangetopair_bool </pre> <p>(End of definition for \l__zrefclever_tpairsep_tl and others.)</p>

`\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`

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

```

3767 \seq_new:N \l__zrefclever_refbounds_first_seq
3768 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3769 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3770 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3771 \seq_new:N \l__zrefclever_refbounds_mid_seq
3772 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3773 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3774 \seq_new:N \l__zrefclever_refbounds_last_seq
3775 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3776 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3777 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3778 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool
  
```

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

`\l_zrefclever_verbose_testing_bool`

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

```

3779 \bool_new:N \l__zrefclever_verbose_testing_bool
  
```

(End of definition for `\l_zrefclever_verbose_testing_bool`.)

Main functions

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

```

3780 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3781 {
3782   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3783     \l__zrefclever_zcref_labels_seq
3784   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3785   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3786   \tl_clear:N \l__zrefclever_type_first_label_tl
3787   \tl_clear:N \l__zrefclever_type_first_label_type_tl
3788   \tl_clear:N \l__zrefclever_range_beg_label_tl
3789   \tl_clear:N \l__zrefclever_range_end_ref_tl
3790   \int_zero:N \l__zrefclever_label_count_int
3791   \int_zero:N \l__zrefclever_type_count_int
3792   \int_zero:N \l__zrefclever_ref_count_int
3793   \int_zero:N \l__zrefclever_range_count_int
3794   \int_zero:N \l__zrefclever_range_same_count_int
3795   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3796   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3797
3798   % Get type block options (not type-specific).
3799   \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3800     { \l__zrefclever_label_type_a_tl }
3801     { \l__zrefclever_ref_language_tl }
3802     \l__zrefclever_tpairsep_tl
3803   \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3804     { \l__zrefclever_label_type_a_tl }
3805     { \l__zrefclever_ref_language_tl }
3806     \l__zrefclever_tlistsep_tl
  
```

```

3807 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3808 { \l__zrefclever_label_type_a_tl }
3809 { \l__zrefclever_ref_language_tl }
3810 \l__zrefclever_tlastsep_tl
3811
3812 % Process label stack.
3813 \bool_set_false:N \l__zrefclever_typeset_last_bool
3814 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3815 {
3816   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3817   \l__zrefclever_label_a_tl
3818   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3819   {
3820     \tl_clear:N \l__zrefclever_label_b_tl
3821     \bool_set_true:N \l__zrefclever_typeset_last_bool
3822   }
3823   {
3824     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3825     \l__zrefclever_label_b_tl
3826   }
3827
3828   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3829   {
3830     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3831     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3832   }
3833   {
3834     \__zrefclever_extract_default:NVnn
3835     \l__zrefclever_label_type_a_tl
3836     \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3837     \__zrefclever_extract_default:NVnn
3838     \l__zrefclever_label_type_b_tl
3839     \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3840   }
3841
3842   % First, we establish whether the "current label" (i.e. 'a') is the
3843   % last one of its type. This can happen because the "next label"
3844   % (i.e. 'b') is of a different type (or different definition status),
3845   % or because we are at the end of the list.
3846   \bool_if:NTF \l__zrefclever_typeset_last_bool
3847   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3848   {
3849     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3850     {
3851       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3852       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3853       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3854     }
3855     {
3856       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3857       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3858       {
3859         % Neither is undefined, we must check the types.
3860         \tl_if_eq:NNTF

```



```

3861         \l__zrefclever_label_type_a_tl
3862         \l__zrefclever_label_type_b_tl
3863         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3864         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3865     }
3866 }
3867 }
3868
3869 % Handle warnings in case of reference or type undefined.
3870 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3871 \zref@refused { \l__zrefclever_label_a_tl }
3872 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3873 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3874 {}
3875 {
3876     \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3877     {
3878         \msg_warning:nxx { zref-clever } { missing-type }
3879         { \l__zrefclever_label_a_tl }
3880     }
3881     \zref@ifrefcontainsprop
3882     { \l__zrefclever_label_a_tl }
3883     { \l__zrefclever_ref_property_tl }
3884     { }
3885     {
3886         \msg_warning:nxxx { zref-clever } { missing-property }
3887         { \l__zrefclever_ref_property_tl }
3888         { \l__zrefclever_label_a_tl }
3889     }
3890 }
3891
3892 % Get possibly type-specific separators, refbounds, font and other
3893 % options, once per type.
3894 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3895 {
3896     \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3897     { \l__zrefclever_label_type_a_tl }
3898     { \l__zrefclever_ref_language_tl }
3899     \l__zrefclever_namesep_tl
3900     \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3901     { \l__zrefclever_label_type_a_tl }
3902     { \l__zrefclever_ref_language_tl }
3903     \l__zrefclever_pairsep_tl
3904     \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3905     { \l__zrefclever_label_type_a_tl }
3906     { \l__zrefclever_ref_language_tl }
3907     \l__zrefclever_listsep_tl
3908     \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3909     { \l__zrefclever_label_type_a_tl }
3910     { \l__zrefclever_ref_language_tl }
3911     \l__zrefclever_lastsep_tl
3912     \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3913     { \l__zrefclever_label_type_a_tl }
3914     { \l__zrefclever_ref_language_tl }

```

```

3915 \l__zrefclever_rangeseptl
3916 \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3917 { \l__zrefclever_label_type_a_tl }
3918 { \l__zrefclever_ref_language_tl }
3919 \l__zrefclever_namefont_tl
3920 \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3921 { \l__zrefclever_label_type_a_tl }
3922 { \l__zrefclever_ref_language_tl }
3923 \l__zrefclever_reffont_tl
3924 \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3925 { \l__zrefclever_label_type_a_tl }
3926 { \l__zrefclever_ref_language_tl }
3927 \l__zrefclever_endrangefunc_tl
3928 \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3929 { \l__zrefclever_label_type_a_tl }
3930 { \l__zrefclever_ref_language_tl }
3931 \l__zrefclever_endrangeprop_tl
3932 \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3933 { \l__zrefclever_label_type_a_tl }
3934 { \l__zrefclever_ref_language_tl }
3935 \l__zrefclever_cap_bool
3936 \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3937 { \l__zrefclever_label_type_a_tl }
3938 { \l__zrefclever_ref_language_tl }
3939 \l__zrefclever_abbrev_bool
3940 \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3941 { \l__zrefclever_label_type_a_tl }
3942 { \l__zrefclever_ref_language_tl }
3943 \l__zrefclever_rangetopair_bool
3944 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3945 { \l__zrefclever_label_type_a_tl }
3946 { \l__zrefclever_ref_language_tl }
3947 \l__zrefclever_refbounds_first_seq
3948 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }
3949 { \l__zrefclever_label_type_a_tl }
3950 { \l__zrefclever_ref_language_tl }
3951 \l__zrefclever_refbounds_first_sg_seq
3952 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3953 { \l__zrefclever_label_type_a_tl }
3954 { \l__zrefclever_ref_language_tl }
3955 \l__zrefclever_refbounds_first_pb_seq
3956 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3957 { \l__zrefclever_label_type_a_tl }
3958 { \l__zrefclever_ref_language_tl }
3959 \l__zrefclever_refbounds_first_rb_seq
3960 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3961 { \l__zrefclever_label_type_a_tl }
3962 { \l__zrefclever_ref_language_tl }
3963 \l__zrefclever_refbounds_mid_seq
3964 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3965 { \l__zrefclever_label_type_a_tl }
3966 { \l__zrefclever_ref_language_tl }
3967 \l__zrefclever_refbounds_mid_rb_seq
3968 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }

```

```

3969         { \l__zrefclever_label_type_a_tl }
3970         { \l__zrefclever_ref_language_tl }
3971         \l__zrefclever_refbounds_mid_re_seq
3972     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3973     { \l__zrefclever_label_type_a_tl }
3974     { \l__zrefclever_ref_language_tl }
3975     \l__zrefclever_refbounds_last_seq
3976     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3977     { \l__zrefclever_label_type_a_tl }
3978     { \l__zrefclever_ref_language_tl }
3979     \l__zrefclever_refbounds_last_pe_seq
3980     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3981     { \l__zrefclever_label_type_a_tl }
3982     { \l__zrefclever_ref_language_tl }
3983     \l__zrefclever_refbounds_last_re_seq
3984 }
3985
3986 % Here we send this to a couple of auxiliary functions.
3987 \bool_if:NTF \l__zrefclever_last_of_type_bool
3988 % There exists no next label of the same type as the current.
3989 { \__zrefclever_typeset_refs_last_of_type: }
3990 % There exists a next label of the same type as the current.
3991 { \__zrefclever_typeset_refs_not_last_of_type: }
3992 }
3993 }

```

(End of definition for __zrefclever_typeset_refs:.)

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

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

```

3994 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3995 {
3996     % Process the current label to the current queue.
3997     \int_case:nnF { \l__zrefclever_label_count_int }
3998     {
3999         % It is the last label of its type, but also the first one, and that's
4000         % what matters here: just store it.
4001         % Test: 'zc-typeset01.lvt': "Last of type: single"
4002         { 0 }
4003         {
4004             \tl_set:NV \l__zrefclever_type_first_label_tl
4005             \l__zrefclever_label_a_tl
4006             \tl_set:NV \l__zrefclever_type_first_label_type_tl
4007             \l__zrefclever_label_type_a_tl
4008             \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4009             \l__zrefclever_refbounds_first_sg_seq

```

```

4010     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4011 }
4012
4013 % The last is the second: we have a pair (if not repeated).
4014 % Test: 'zc-typeset01.lvt': "Last of type: pair"
4015 { 1 }
4016 {
4017     \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4018     {
4019         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4020         \l__zrefclever_refbounds_first_sg_seq
4021         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4022     }
4023     {
4024         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4025         {
4026             \exp_not:V \l__zrefclever_pairsep_tl
4027             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4028             \l__zrefclever_refbounds_last_pe_seq
4029         }
4030         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4031         \l__zrefclever_refbounds_first_pb_seq
4032         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4033     }
4034 }
4035 }
4036 % Last is third or more of its type: without repetition, we'd have the
4037 % last element on a list, but control for possible repetition.
4038 {
4039     \int_case:nnF { \l__zrefclever_range_count_int }
4040     {
4041         % There was no range going on.
4042         % Test: 'zc-typeset01.lvt': "Last of type: not range"
4043         { 0 }
4044         {
4045             \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4046             {
4047                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4048                 {
4049                     \exp_not:V \l__zrefclever_pairsep_tl
4050                     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4051                     \l__zrefclever_refbounds_last_pe_seq
4052                 }
4053             }
4054             {
4055                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4056                 {
4057                     \exp_not:V \l__zrefclever_lastsep_tl
4058                     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4059                     \l__zrefclever_refbounds_last_seq
4060                 }
4061             }
4062         }
4063         % Last in the range is also the second in it.

```

```

4064 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4065 { 1 }
4066 {
4067   \int_compare:nNnTF
4068     { \l__zrefclever_range_same_count_int } = { 1 }
4069     {
4070       % We know 'range_beg_is_first_bool' is false, since this is
4071       % the second element in the range, but the third or more in
4072       % the type list.
4073       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4074       {
4075         \exp_not:V \l__zrefclever_pairsep_tl
4076         \__zrefclever_get_ref:VN
4077         \l__zrefclever_range_beg_label_tl
4078         \l__zrefclever_refbounds_last_pe_seq
4079       }
4080       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4081       \l__zrefclever_refbounds_first_pb_seq
4082       \bool_set_true:N
4083       \l__zrefclever_type_first_refbounds_set_bool
4084     }
4085     {
4086       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4087       {
4088         \exp_not:V \l__zrefclever_listsep_tl
4089         \__zrefclever_get_ref:VN
4090         \l__zrefclever_range_beg_label_tl
4091         \l__zrefclever_refbounds_mid_seq
4092         \exp_not:V \l__zrefclever_lastsep_tl
4093         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4094         \l__zrefclever_refbounds_last_seq
4095       }
4096     }
4097   }
4098 }
4099 % Last in the range is third or more in it.
4100 {
4101   \int_case:nnF
4102     {
4103       \l__zrefclever_range_count_int -
4104       \l__zrefclever_range_same_count_int
4105     }
4106     {
4107       % Repetition, not a range.
4108       % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4109       { 0 }
4110       {
4111         % If 'range_beg_is_first_bool' is true, it means it was also
4112         % the first of the type, and hence its typesetting was
4113         % already handled, and we just have to set refbounds.
4114         \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4115         {
4116           \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4117           \l__zrefclever_refbounds_first_sg_seq

```

```

4118         \bool_set_true:N
4119         \l__zrefclever_type_first_refbounds_set_bool
4120     }
4121     {
4122         \int_compare:nNnTF
4123         { \l__zrefclever_ref_count_int } < { 2 }
4124         {
4125             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4126             {
4127                 \exp_not:V \l__zrefclever_pairsep_tl
4128                 \__zrefclever_get_ref:VN
4129                 \l__zrefclever_range_beg_label_tl
4130                 \l__zrefclever_refbounds_last_pe_seq
4131             }
4132         }
4133         {
4134             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4135             {
4136                 \exp_not:V \l__zrefclever_lastsep_tl
4137                 \__zrefclever_get_ref:VN
4138                 \l__zrefclever_range_beg_label_tl
4139                 \l__zrefclever_refbounds_last_seq
4140             }
4141         }
4142     }
4143 }
4144 % A 'range', but with no skipped value, treat as pair if range
4145 % started with first of type, otherwise as list.
4146 % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4147 { 1 }
4148 {
4149     % Ditto.
4150     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4151     {
4152         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4153         \l__zrefclever_refbounds_first_pb_seq
4154         \bool_set_true:N
4155         \l__zrefclever_type_first_refbounds_set_bool
4156         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4157         {
4158             \exp_not:V \l__zrefclever_pairsep_tl
4159             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4160             \l__zrefclever_refbounds_last_pe_seq
4161         }
4162     }
4163     {
4164         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4165         {
4166             \exp_not:V \l__zrefclever_listsep_tl
4167             \__zrefclever_get_ref:VN
4168             \l__zrefclever_range_beg_label_tl
4169             \l__zrefclever_refbounds_mid_seq
4170         }
4171         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

4172         {
4173             \exp_not:V \l__zrefclever_lastsep_tl
4174             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4175             \l__zrefclever_refbounds_last_seq
4176         }
4177     }
4178 }
4179 }
4180 {
4181     % An actual range.
4182     % Test: 'zc-typeset01.lvt': "Last of type: range"
4183     % Ditto.
4184     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4185     {
4186         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4187         \l__zrefclever_refbounds_first_rb_seq
4188         \bool_set_true:N
4189         \l__zrefclever_type_first_refbounds_set_bool
4190     }
4191     {
4192         \int_compare:nNnTF
4193         { \l__zrefclever_ref_count_int } < { 2 }
4194         {
4195             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4196             {
4197                 \exp_not:V \l__zrefclever_pairsep_tl
4198                 \__zrefclever_get_ref:VN
4199                 \l__zrefclever_range_beg_label_tl
4200                 \l__zrefclever_refbounds_mid_rb_seq
4201             }
4202             \seq_set_eq:NN
4203             \l__zrefclever_type_first_refbounds_seq
4204             \l__zrefclever_refbounds_first_pb_seq
4205             \bool_set_true:N
4206             \l__zrefclever_type_first_refbounds_set_bool
4207         }
4208         {
4209             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4210             {
4211                 \exp_not:V \l__zrefclever_lastsep_tl
4212                 \__zrefclever_get_ref:VN
4213                 \l__zrefclever_range_beg_label_tl
4214                 \l__zrefclever_refbounds_mid_rb_seq
4215             }
4216         }
4217     }
4218     \bool_lazy_and:nnTF
4219     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4220     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4221     {
4222         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4223         \l__zrefclever_range_beg_label_tl
4224         \l__zrefclever_label_a_tl
4225         \l__zrefclever_range_end_ref_tl

```

```

4226         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4227         {
4228             \exp_not:V \l__zrefclever_rangeseq_tl
4229             \__zrefclever_get_ref_endrange:VVN
4230             \l__zrefclever_label_a_tl
4231             \l__zrefclever_range_end_ref_tl
4232             \l__zrefclever_refbounds_last_re_seq
4233         }
4234     }
4235     {
4236         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4237         {
4238             \exp_not:V \l__zrefclever_rangeseq_tl
4239             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4240             \l__zrefclever_refbounds_last_re_seq
4241         }
4242     }
4243 }
4244 }
4245 }
4246
4247 % Handle "range" option. The idea is simple: if the queue is not empty,
4248 % we replace it with the end of the range (or pair). We can still
4249 % retrieve the end of the range from 'label_a' since we know to be
4250 % processing the last label of its type at this point.
4251 \bool_if:NT \l__zrefclever_typeset_range_bool
4252 {
4253     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4254     {
4255         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4256         { }
4257         {
4258             \msg_warning:nxx { zref-clever } { single-element-range }
4259             { \l__zrefclever_type_first_label_type_tl }
4260         }
4261     }
4262     {
4263         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4264         \bool_if:NT \l__zrefclever_rangetopair_bool
4265         {
4266             \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4267             { }
4268             {
4269                 \__zrefclever_labels_in_sequence:nn
4270                 { \l__zrefclever_type_first_label_tl }
4271                 { \l__zrefclever_label_a_tl }
4272             }
4273         }
4274         % Test: 'zc-typeset01.lvt': "Last of type: option range"
4275         % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4276         \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4277         {
4278             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4279             {

```



```

4280         \exp_not:V \l__zrefclever_pairsep_tl
4281         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4282         \l__zrefclever_refbounds_last_pe_seq
4283     }
4284     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4285     \l__zrefclever_refbounds_first_pb_seq
4286     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4287 }
4288 {
4289     \bool_lazy_and:nnTF
4290     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4291     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4292     {
4293         % We must get 'type_first_label_tl' instead of
4294         % 'range_beg_label_tl' here, since it is not necessary
4295         % that the first of type was actually starting a range for
4296         % the 'range' option to be used.
4297         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4298         \l__zrefclever_type_first_label_tl
4299         \l__zrefclever_label_a_tl
4300         \l__zrefclever_range_end_ref_tl
4301         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4302         {
4303             \exp_not:V \l__zrefclever_rangeseq_tl
4304             \__zrefclever_get_ref_endrange:VVN
4305             \l__zrefclever_label_a_tl
4306             \l__zrefclever_range_end_ref_tl
4307             \l__zrefclever_refbounds_last_re_seq
4308         }
4309     }
4310     {
4311         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4312         {
4313             \exp_not:V \l__zrefclever_rangeseq_tl
4314             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4315             \l__zrefclever_refbounds_last_re_seq
4316         }
4317     }
4318     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4319     \l__zrefclever_refbounds_first_rb_seq
4320     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4321 }
4322 }
4323 }
4324
4325 % If none of the special cases for the first of type refbounds have been
4326 % set, do it.
4327 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4328 {
4329     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4330     \l__zrefclever_refbounds_first_seq
4331 }
4332
4333 % Now that the type block is finished, we can add the name and the first

```

```

4334 % ref to the queue. Also, if "typeset" option is not "both", handle it
4335 % here as well.
4336 \__zrefclever_type_name_setup:
4337 \bool_if:nTF
4338 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4339 {
4340   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4341   { \__zrefclever_get_ref_first: }
4342 }
4343 {
4344   \bool_if:NTF \l__zrefclever_typeset_ref_bool
4345   {
4346     % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4347     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4348     {
4349       \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4350       \l__zrefclever_type_first_refbounds_seq
4351     }
4352   }
4353   {
4354     \bool_if:NTF \l__zrefclever_typeset_name_bool
4355     {
4356       % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4357       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4358       {
4359         \bool_if:NTF \l__zrefclever_name_in_link_bool
4360         {
4361           \exp_not:N \group_begin:
4362           \exp_not:V \l__zrefclever_namefont_tl
4363           \__zrefclever_hyperlink:nnn
4364           {
4365             \__zrefclever_extract_url_unexp:V
4366             \l__zrefclever_type_first_label_tl
4367           }
4368           {
4369             \__zrefclever_extract_unexp:Vnn
4370             \l__zrefclever_type_first_label_tl
4371             { anchor } { }
4372           }
4373           { \exp_not:V \l__zrefclever_type_name_tl }
4374           \exp_not:N \group_end:
4375         }
4376         {
4377           \exp_not:N \group_begin:
4378           \exp_not:V \l__zrefclever_namefont_tl
4379           \exp_not:V \l__zrefclever_type_name_tl
4380           \exp_not:N \group_end:
4381         }
4382       }
4383     }
4384     {
4385       % Logically, this case would correspond to "typeset=none", but
4386       % it should not occur, given that the options are set up to
4387       % typeset either "ref" or "name". Still, leave here a

```

```

4388         % sensible fallback, equal to the behavior of "both".
4389         % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4390         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4391         { \__zrefclever_get_ref_first: }
4392     }
4393 }
4394 }
4395
4396 % Typeset the previous type block, if there is one.
4397 \int_compare:nNt { \l__zrefclever_type_count_int } > { 0 }
4398 {
4399     \int_compare:nNt { \l__zrefclever_type_count_int } > { 1 }
4400     { \l__zrefclever_tlistsep_tl }
4401     \l__zrefclever_typeset_queue_prev_tl
4402 }
4403
4404 % Extra log for testing.
4405 \bool_if:NT \l__zrefclever_verbose_testing_bool
4406 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4407
4408 % Wrap up loop, or prepare for next iteration.
4409 \bool_if:NTF \l__zrefclever_typeset_last_bool
4410 {
4411     % We are finishing, typeset the current queue.
4412     \int_case:nnF { \l__zrefclever_type_count_int }
4413     {
4414         % Single type.
4415         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4416         { 0 }
4417         { \l__zrefclever_typeset_queue_curr_tl }
4418         % Pair of types.
4419         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4420         { 1 }
4421         {
4422             \l__zrefclever_tpairsep_tl
4423             \l__zrefclever_typeset_queue_curr_tl
4424         }
4425     }
4426     {
4427         % Last in list of types.
4428         % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4429         \l__zrefclever_tlastsep_tl
4430         \l__zrefclever_typeset_queue_curr_tl
4431     }
4432     % And nudge in case of multitype reference.
4433     \bool_lazy_all:nT
4434     {
4435         { \l__zrefclever_nudge_enabled_bool }
4436         { \l__zrefclever_nudge_multitype_bool }
4437         { \int_compare:nNt { \l__zrefclever_type_count_int } > { 0 } }
4438     }
4439     { \msg_warning:nn { zref-clever } { nudge-multitype } }
4440 }
4441 {

```

```

4442      % There are further labels, set variables for next iteration.
4443      \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4444      \l__zrefclever_typeset_queue_curr_tl
4445      \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4446      \tl_clear:N \l__zrefclever_type_first_label_tl
4447      \tl_clear:N \l__zrefclever_type_first_label_type_tl
4448      \tl_clear:N \l__zrefclever_range_beg_label_tl
4449      \tl_clear:N \l__zrefclever_range_end_ref_tl
4450      \int_zero:N \l__zrefclever_label_count_int
4451      \int_zero:N \l__zrefclever_ref_count_int
4452      \int_incr:N \l__zrefclever_type_count_int
4453      \int_zero:N \l__zrefclever_range_count_int
4454      \int_zero:N \l__zrefclever_range_same_count_int
4455      \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4456      \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4457    }
4458  }

```

(End of definition for __zrefclever_typeset_refs_last_of_type:.)

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

```

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

```

```

4492 \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4493 \tl_set:NV \l__zrefclever_range_beg_label_tl
4494 \l__zrefclever_label_a_tl
4495 \tl_clear:N \l__zrefclever_range_end_ref_tl
4496 \int_incr:N \l__zrefclever_range_count_int
4497 \bool_if:NT \l__zrefclever_next_is_same_bool
4498 { \int_incr:N \l__zrefclever_range_same_count_int }
4499 }
4500 }
4501 {
4502 % Current label is neither the first (nor the last) of its type.
4503 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4504 {
4505 % Starting, or continuing a range.
4506 \int_compare:nNnTF
4507 { \l__zrefclever_range_count_int } = { 0 }
4508 {
4509 % There was no range going, we are starting one.
4510 \tl_set:NV \l__zrefclever_range_beg_label_tl
4511 \l__zrefclever_label_a_tl
4512 \tl_clear:N \l__zrefclever_range_end_ref_tl
4513 \int_incr:N \l__zrefclever_range_count_int
4514 \bool_if:NT \l__zrefclever_next_is_same_bool
4515 { \int_incr:N \l__zrefclever_range_same_count_int }
4516 }
4517 {
4518 % Second or more in the range, but not the last.
4519 \int_incr:N \l__zrefclever_range_count_int
4520 \bool_if:NT \l__zrefclever_next_is_same_bool
4521 { \int_incr:N \l__zrefclever_range_same_count_int }
4522 }
4523 }
4524 {
4525 % Next element is not in sequence: there was no range, or we are
4526 % closing one.
4527 \int_case:nnF { \l__zrefclever_range_count_int }
4528 {
4529 % There was no range going on.
4530 % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4531 { 0 }
4532 {
4533 \int_incr:N \l__zrefclever_ref_count_int
4534 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4535 {
4536 \exp_not:V \l__zrefclever_listsep_tl
4537 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4538 \l__zrefclever_refbounds_mid_seq
4539 }
4540 }
4541 % Last is second in the range: if 'range_same_count' is also
4542 % '1', it's a repetition (drop it), otherwise, it's a "pair
4543 % within a list", treat as list.
4544 % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4545 % Test: 'zc-typeset01.lvt': "Not last of type: range pair"

```

```

4546 { 1 }
4547 {
4548     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4549     {
4550         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4551         \l__zrefclever_refbounds_first_seq
4552         \bool_set_true:N
4553         \l__zrefclever_type_first_refbounds_set_bool
4554     }
4555     {
4556         \int_incr:N \l__zrefclever_ref_count_int
4557         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4558         {
4559             \exp_not:V \l__zrefclever_listsep_tl
4560             \__zrefclever_get_ref:VN
4561             \l__zrefclever_range_beg_label_tl
4562             \l__zrefclever_refbounds_mid_seq
4563         }
4564     }
4565     \int_compare:nNnF
4566     { \l__zrefclever_range_same_count_int } = { 1 }
4567     {
4568         \int_incr:N \l__zrefclever_ref_count_int
4569         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4570         {
4571             \exp_not:V \l__zrefclever_listsep_tl
4572             \__zrefclever_get_ref:VN
4573             \l__zrefclever_label_a_tl
4574             \l__zrefclever_refbounds_mid_seq
4575         }
4576     }
4577 }
4578 }
4579 {
4580     % Last is third or more in the range: if 'range_count' and
4581     % 'range_same_count' are the same, its a repetition (drop it),
4582     % if they differ by '1', its a list, if they differ by more,
4583     % it is a real range.
4584     \int_case:nnF
4585     {
4586         \l__zrefclever_range_count_int -
4587         \l__zrefclever_range_same_count_int
4588     }
4589     {
4590         % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4591         { 0 }
4592         {
4593             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4594             {
4595                 \seq_set_eq:NN
4596                 \l__zrefclever_type_first_refbounds_seq
4597                 \l__zrefclever_refbounds_first_seq
4598                 \bool_set_true:N
4599                 \l__zrefclever_type_first_refbounds_set_bool

```

```

4600     }
4601     {
4602         \int_incr:N \l__zrefclever_ref_count_int
4603         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4604         {
4605             \exp_not:V \l__zrefclever_listsep_tl
4606             \__zrefclever_get_ref:VN
4607             \l__zrefclever_range_beg_label_tl
4608             \l__zrefclever_refbounds_mid_seq
4609         }
4610     }
4611 }
4612 % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4613 { 1 }
4614 {
4615     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4616     {
4617         \seq_set_eq:NN
4618         \l__zrefclever_type_first_refbounds_seq
4619         \l__zrefclever_refbounds_first_seq
4620         \bool_set_true:N
4621         \l__zrefclever_type_first_refbounds_set_bool
4622     }
4623     {
4624         \int_incr:N \l__zrefclever_ref_count_int
4625         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4626         {
4627             \exp_not:V \l__zrefclever_listsep_tl
4628             \__zrefclever_get_ref:VN
4629             \l__zrefclever_range_beg_label_tl
4630             \l__zrefclever_refbounds_mid_seq
4631         }
4632     }
4633     \int_incr:N \l__zrefclever_ref_count_int
4634     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4635     {
4636         \exp_not:V \l__zrefclever_listsep_tl
4637         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4638         \l__zrefclever_refbounds_mid_seq
4639     }
4640 }
4641 }
4642 {
4643     % Test: 'zc-typeset01.lvt': "Not last of type: range"
4644     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4645     {
4646         \seq_set_eq:NN
4647         \l__zrefclever_type_first_refbounds_seq
4648         \l__zrefclever_refbounds_first_rb_seq
4649         \bool_set_true:N
4650         \l__zrefclever_type_first_refbounds_set_bool
4651     }
4652     {
4653         \int_incr:N \l__zrefclever_ref_count_int

```

```

4654         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4655         {
4656             \exp_not:V \l__zrefclever_listsep_tl
4657             \__zrefclever_get_ref:VN
4658             \l__zrefclever_range_beg_label_tl
4659             \l__zrefclever_refbounds_mid_rb_seq
4660         }
4661     }
4662     % For the purposes of the serial comma, and thus for the
4663     % distinction of 'lastsep' and 'pairsep', a "range" counts
4664     % as one. Since 'range_beg' has already been counted
4665     % (here or with the first of type), we refrain from
4666     % incrementing 'ref_count_int'.
4667     \bool_lazy_and:nnTF
4668     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4669     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4670     {
4671         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4672         \l__zrefclever_range_beg_label_tl
4673         \l__zrefclever_label_a_tl
4674         \l__zrefclever_range_end_ref_tl
4675         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4676         {
4677             \exp_not:V \l__zrefclever_rangesep_tl
4678             \__zrefclever_get_ref_endrange:VVN
4679             \l__zrefclever_label_a_tl
4680             \l__zrefclever_range_end_ref_tl
4681             \l__zrefclever_refbounds_mid_re_seq
4682         }
4683     }
4684     {
4685         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4686         {
4687             \exp_not:V \l__zrefclever_rangesep_tl
4688             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4689             \l__zrefclever_refbounds_mid_re_seq
4690         }
4691     }
4692 }
4693 }
4694 % We just closed a range, reset 'range_beg_is_first' in case a
4695 % second range for the same type occurs, in which case its
4696 % 'range_beg' will no longer be 'first'.
4697 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4698 % Reset counters.
4699 \int_zero:N \l__zrefclever_range_count_int
4700 \int_zero:N \l__zrefclever_range_same_count_int
4701 }
4702 }
4703 % Step label counter for next iteration.
4704 \int_incr:N \l__zrefclever_label_count_int
4705 }

```

(End of definition for __zrefclever_typeset_refs_not_last_of_type:.)

Auxiliary functions

`__zrefclever_get_ref:nN` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:nN` handles all references but the first of its type, and `__zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `__zrefclever_typeset_refs_last_of_type:` and `__zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX 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 \mathfrak{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 \mathfrak{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.

```
4706 \cs_new_protected:Npn \__zrefclever_ref_default:
4707   { \zref@default }
4708 \cs_new_protected:Npn \__zrefclever_name_default:
4709   { \zref@default }
```

(End of definition for `__zrefclever_ref_default:` and `__zrefclever_name_default:`.)

`__zrefclever_get_ref:nN` Handles a complete reference block to be accumulated in the “queue”, including ref-bounds, and hyperlinking. For use with all labels, except the first of its type, which is done by `__zrefclever_get_ref_first:`, and the last of a range, which is done by `__zrefclever_get_ref_endrange:nnN`.

```
\__zrefclever_get_ref:nN {<label>} {<refbounds>}

4710 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4711   {
4712     \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4713     {
4714       \bool_if:nTF
4715         {
4716           \l__zrefclever_hyperlink_bool &&
4717           ! \l__zrefclever_link_star_bool
4718         }
4719         {
4720           \seq_item:Nn #2 { 1 }
4721           \__zrefclever_hyperlink:nnn
```

```

4722 { \_zrefclever_extract_url_unexp:n {#1} }
4723 { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4724 {
4725   \seq_item:Nn #2 { 2 }
4726   \exp_not:N \group_begin:
4727   \exp_not:V \l__zrefclever_reffont_tl
4728   \_zrefclever_extract_unexp:nvn {#1}
4729   { \l__zrefclever_ref_property_tl } { }
4730   \exp_not:N \group_end:
4731   \seq_item:Nn #2 { 3 }
4732 }
4733 \seq_item:Nn #2 { 4 }
4734 }
4735 {
4736   \seq_item:Nn #2 { 1 }
4737   \seq_item:Nn #2 { 2 }
4738   \exp_not:N \group_begin:
4739   \exp_not:V \l__zrefclever_reffont_tl
4740   \_zrefclever_extract_unexp:nvn {#1}
4741   { \l__zrefclever_ref_property_tl } { }
4742   \exp_not:N \group_end:
4743   \seq_item:Nn #2 { 3 }
4744   \seq_item:Nn #2 { 4 }
4745 }
4746 }
4747 { \_zrefclever_ref_default: }
4748 }
4749 \cs_generate_variant:Nn \_zrefclever_get_ref:nN { VN }

```

(End of definition for _zrefclever_get_ref:nN.)

```

\_zrefclever_get_ref_endrange:nnN      \_zrefclever_get_ref_endrange:nnN {<label>} {<reference>} {<refbounds>}
4750 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3
4751 {
4752   \str_if_eq:nnTF {#2} { zc@missingproperty }
4753   { \_zrefclever_ref_default: }
4754   {
4755     \bool_if:nTF
4756     {
4757       \l__zrefclever_hyperlink_bool &&
4758       ! \l__zrefclever_link_star_bool
4759     }
4760     {
4761       \seq_item:Nn #3 { 1 }
4762       \_zrefclever_hyperlink:nnn
4763       { \_zrefclever_extract_url_unexp:n {#1} }
4764       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4765       {
4766         \seq_item:Nn #3 { 2 }
4767         \exp_not:N \group_begin:
4768         \exp_not:V \l__zrefclever_reffont_tl
4769         \exp_not:n {#2}
4770         \exp_not:N \group_end:
4771         \seq_item:Nn #3 { 3 }

```

```

4772     }
4773     \seq_item:Nn #3 { 4 }
4774   }
4775   {
4776     \seq_item:Nn #3 { 1 }
4777     \seq_item:Nn #3 { 2 }
4778     \exp_not:N \group_begin:
4779     \exp_not:V \l__zrefclever_reffont_tl
4780     \exp_not:n {#2}
4781     \exp_not:N \group_end:
4782     \seq_item:Nn #3 { 3 }
4783     \seq_item:Nn #3 { 4 }
4784   }
4785 }
4786 }
4787 \cs_generate_variant:Nn \__zrefclever_get_ref_endrange:nnN { VVN }

```

(End of definition for __zrefclever_get_ref_endrange:nnN.)

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

```

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

```

```

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

```

```

4869     }
4870     {
4871         \seq_item:Nn
4872         \l__zrefclever_type_first_refbounds_seq { 2 }
4873         \exp_not:N \group_begin:
4874         \exp_not:V \l__zrefclever_reffont_tl
4875         \__zrefclever_extract_unexp:Vvn
4876         \l__zrefclever_type_first_label_tl
4877         { \l__zrefclever_ref_property_tl } { }
4878         \exp_not:N \group_end:
4879         \seq_item:Nn
4880         \l__zrefclever_type_first_refbounds_seq { 3 }
4881     }
4882     \seq_item:Nn
4883     \l__zrefclever_type_first_refbounds_seq { 4 }
4884 }
4885 {
4886     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4887     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4888     \exp_not:N \group_begin:
4889     \exp_not:V \l__zrefclever_reffont_tl
4890     \__zrefclever_extract_unexp:Vvn
4891     \l__zrefclever_type_first_label_tl
4892     { \l__zrefclever_ref_property_tl } { }
4893     \exp_not:N \group_end:
4894     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4895     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4896 }
4897 }
4898 { \__zrefclever_ref_default: }
4899 }
4900 }
4901 }

```

(End of definition for __zrefclever_get_ref_first:.)

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

```

4902 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4903 {
4904     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4905     {
4906         \tl_clear:N \l__zrefclever_type_name_tl
4907         \bool_set_true:N \l__zrefclever_type_name_missing_bool

```

```

4908 }
4909 {
4910   \tl_if_eq:NnTF
4911     \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4912     {
4913       \tl_clear:N \l__zrefclever_type_name_tl
4914       \bool_set_true:N \l__zrefclever_type_name_missing_bool
4915     }
4916     {
4917       % Determine whether we should use capitalization, abbreviation,
4918       % and plural.
4919       \bool_lazy_or:nnTF
4920         { \l__zrefclever_cap_bool }
4921         {
4922           \l__zrefclever_capfirst_bool &&
4923           \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4924         }
4925         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4926         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4927       % If the queue is empty, we have a singular, otherwise, plural.
4928       \tl_if_empty:NNTF \l__zrefclever_typeset_queue_curr_tl
4929         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4930         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4931       \bool_lazy_and:nnTF
4932         { \l__zrefclever_abbrev_bool }
4933         {
4934           ! \int_compare_p:nNn
4935             { \l__zrefclever_type_count_int } = { 0 } ||
4936           ! \l__zrefclever_noabbrev_first_bool
4937         }
4938         {
4939           \tl_set:NV \l__zrefclever_name_format_fallback_tl
4940             \l__zrefclever_name_format_tl
4941           \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4942         }
4943       { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4944
4945       % Handle number and gender nudges.
4946       \bool_if:NT \l__zrefclever_nudge_enabled_bool
4947       {
4948         \bool_if:NNTF \l__zrefclever_nudge_singular_bool
4949         {
4950           \tl_if_empty:NNTF \l__zrefclever_typeset_queue_curr_tl
4951           {
4952             \msg_warning:nnx { zref-clever }
4953               { nudge-plural-when-sg }
4954             { \l__zrefclever_type_first_label_type_tl }
4955           }
4956         }
4957         {
4958           \bool_lazy_all:nT
4959           {
4960             { \l__zrefclever_nudge_comptosing_bool }
4961             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }

```

```

4962         {
4963             \int_compare_p:nNn
4964             { \l__zrefclever_label_count_int } > { 0 }
4965         }
4966     }
4967     {
4968         \msg_warning:nxx { zref-clever }
4969         { nudge-comptosing }
4970         { \l__zrefclever_type_first_label_type_tl }
4971     }
4972 }
4973 \bool_lazy_and:nnT
4974 { \l__zrefclever_nudge_gender_bool }
4975 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4976 {
4977     \__zrefclever_get_rf_opt_seq:nxxN { gender }
4978     { \l__zrefclever_type_first_label_type_tl }
4979     { \l__zrefclever_ref_language_tl }
4980     \l__zrefclever_type_name_gender_seq
4981     \seq_if_in:NVF
4982     \l__zrefclever_type_name_gender_seq
4983     \l__zrefclever_ref_gender_tl
4984     {
4985         \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4986         {
4987             \msg_warning:nxxxx { zref-clever }
4988             { nudge-gender-not-declared-for-type }
4989             { \l__zrefclever_ref_gender_tl }
4990             { \l__zrefclever_type_first_label_type_tl }
4991             { \l__zrefclever_ref_language_tl }
4992         }
4993         {
4994             \msg_warning:nxxxxx { zref-clever }
4995             { nudge-gender-mismatch }
4996             { \l__zrefclever_type_first_label_type_tl }
4997             { \l__zrefclever_ref_gender_tl }
4998             {
4999                 \seq_use:Nn
5000                 \l__zrefclever_type_name_gender_seq { ,~ }
5001             }
5002             { \l__zrefclever_ref_language_tl }
5003         }
5004     }
5005 }
5006 }
5007
5008 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5009 {
5010     \__zrefclever_opt_tl_get:cNF
5011     {
5012         \__zrefclever_opt_varname_type:een
5013         { \l__zrefclever_type_first_label_type_tl }
5014         { \l__zrefclever_name_format_tl }
5015         { tl }

```

```

5016 }
5017 \l__zrefclever_type_name_tl
5018 {
5019   \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5020   {
5021     \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5022     \tl_put_left:NV \l__zrefclever_name_format_tl
5023       \l__zrefclever_ref_decl_case_tl
5024   }
5025   \__zrefclever_opt_tl_get:cNF
5026   {
5027     \__zrefclever_opt_varname_lang_type:een
5028     { \l__zrefclever_ref_language_tl }
5029     { \l__zrefclever_type_first_label_type_tl }
5030     { \l__zrefclever_name_format_tl }
5031     { tl }
5032   }
5033   \l__zrefclever_type_name_tl
5034   {
5035     \tl_clear:N \l__zrefclever_type_name_tl
5036     \bool_set_true:N \l__zrefclever_type_name_missing_bool
5037     \msg_warning:nnxx { zref-clever } { missing-name }
5038     { \l__zrefclever_name_format_tl }
5039     { \l__zrefclever_type_first_label_type_tl }
5040   }
5041 }
5042 }
5043 {
5044   \__zrefclever_opt_tl_get:cNF
5045   {
5046     \__zrefclever_opt_varname_type:een
5047     { \l__zrefclever_type_first_label_type_tl }
5048     { \l__zrefclever_name_format_tl }
5049     { tl }
5050   }
5051   \l__zrefclever_type_name_tl
5052   {
5053     \__zrefclever_opt_tl_get:cNF
5054     {
5055       \__zrefclever_opt_varname_type:een
5056       { \l__zrefclever_type_first_label_type_tl }
5057       { \l__zrefclever_name_format_fallback_tl }
5058       { tl }
5059     }
5060     \l__zrefclever_type_name_tl
5061     {
5062       \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5063       {
5064         \tl_put_left:Nn
5065           \l__zrefclever_name_format_tl { - }
5066         \tl_put_left:NV \l__zrefclever_name_format_tl
5067           \l__zrefclever_ref_decl_case_tl
5068         \tl_put_left:Nn
5069           \l__zrefclever_name_format_fallback_tl { - }

```



```

5070         \tl_put_left:NV
5071         \l__zrefclever_name_format_fallback_tl
5072         \l__zrefclever_ref_decl_case_tl
5073     }
5074 \__zrefclever_opt_tl_get:cNF
5075 {
5076     \__zrefclever_opt_varname_lang_type:eeen
5077     { \l__zrefclever_ref_language_tl }
5078     { \l__zrefclever_type_first_label_type_tl }
5079     { \l__zrefclever_name_format_tl }
5080     { tl }
5081 }
5082 \l__zrefclever_type_name_tl
5083 {
5084     \__zrefclever_opt_tl_get:cNF
5085     {
5086         \__zrefclever_opt_varname_lang_type:eeen
5087         { \l__zrefclever_ref_language_tl }
5088         { \l__zrefclever_type_first_label_type_tl }
5089         { \l__zrefclever_name_format_fallback_tl }
5090         { tl }
5091     }
5092     \l__zrefclever_type_name_tl
5093     {
5094         \tl_clear:N \l__zrefclever_type_name_tl
5095         \bool_set_true:N
5096             \l__zrefclever_type_name_missing_bool
5097         \msg_warning:nxxx { zref-clever }
5098             { missing-name }
5099             { \l__zrefclever_name_format_tl }
5100             { \l__zrefclever_type_first_label_type_tl }
5101     }
5102 }
5103 }
5104 }
5105 }
5106 }
5107 }
5108
5109 % Signal whether the type name is to be included in the hyperlink or not.
5110 \bool_lazy_any:nTF
5111 {
5112     { ! \l__zrefclever_hyperlink_bool }
5113     { \l__zrefclever_link_star_bool }
5114     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5115     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5116 }
5117 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5118 {
5119     \bool_lazy_any:nTF
5120     {
5121         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5122         {
5123             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&

```

```

5124         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5125     }
5126     {
5127         \str_if_eq_p:Nn \l__zrefclever_nameinlink_str { single } &&
5128         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5129         \l__zrefclever_typeset_last_bool &&
5130         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5131     }
5132 }
5133 { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5134 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5135 }
5136 }

```

(End of definition for `__zrefclever_type_name_setup:`.)

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

```

        \__zrefclever_hyperlink:nnn {<url/file>} {<anchor>} {<text>}
5137 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5138 {
5139     \tl_if_empty:nTF {#1}
5140     { \hyperlink {#2} {#3} }
5141     { \hyper@linkfile {#3} {#1} {#2} }
5142 }

```

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

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

```

5143 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5144 {
5145     \zref@ifpropundefined { urluse }
5146     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5147     {
5148         \zref@ifrefcontainsprop {#1} { urluse }
5149         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5150         { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5151     }
5152 }
5153 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

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

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

```

5154 \__zrefclever_labels_in_sequence:nn {\label a}\{\label b\}
5155 {
5156   \exp_args:Nxx \tl_if_eq:nnT
5157   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5158   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5159   {
5160     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5161     {
5162       \exp_args:Nxx \tl_if_eq:nnT
5163       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5164       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5165       {
5166         \int_compare:nNnTF
5167         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5168         =
5169         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5170         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5171         {
5172           \int_compare:nNnT
5173           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5174           =
5175           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5176           {
5177             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5178             \bool_set_true:N \l__zrefclever_next_is_same_bool
5179           }
5180         }
5181       }
5182     }
5183   }
5184   \exp_args:Nxx \tl_if_eq:nnT
5185   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5186   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5187   {
5188     \exp_args:Nxx \tl_if_eq:nnT
5189     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5190     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5191     {
5192       \int_compare:nNnTF
5193       { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5194       =
5195       { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5196       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5197       {
5198         \int_compare:nNnT
5199         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5200         =
5201         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5202         {

```

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.

```

5203 \exp_args:Nxx \tl_if_eq:nnT
5204 {
5205   \__zrefclever_extract_unexp:nvn {#1}
5206   { l__zrefclever_ref_property_tl } { }
5207 }
5208 {
5209   \__zrefclever_extract_unexp:nvn {#2}
5210   { l__zrefclever_ref_property_tl } { }
5211 }
5212 {
5213   \bool_set_true:N
5214   \l__zrefclever_next_maybe_range_bool
5215   \bool_set_true:N
5216   \l__zrefclever_next_is_same_bool
5217 }
5218 }
5219 }
5220 }
5221 }
5222 }
5223 }
5224 }

```

(End of definition for `__zrefclever_labels_in_sequence:nn`.)

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

```

\__zrefclever_get_rf_opt_tl:nnnN \__zrefclever_get_rf_opt_tl:nnnN {<option>}
    {<ref type>} {<language>} {<tl variable>}
5225 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5226 {
5227   % First attempt: general options.
5228   \__zrefclever_opt_tl_get:cNF
5229   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5230   #4
5231   {
5232     % If not found, try type specific options.
5233     \__zrefclever_opt_tl_get:cNF
5234     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5235     #4
5236     {
5237       % If not found, try type- and language-specific.
5238       \__zrefclever_opt_tl_get:cNF
5239       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5240       #4
5241       {
5242         % If not found, try language-specific default.
5243         \__zrefclever_opt_tl_get:cNF
5244         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }

```

```

5245         #4
5246     {
5247         % If not found, try fallback.
5248         \_zrefclever_opt_tl_get:cNF
5249         { \_zrefclever_opt_varname_fallback:nn {#1} { tl } }
5250         #4
5251         { \tl_clear:N #4 }
5252     }
5253 }
5254 }
5255 }
5256 }
5257 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_tl:nnnN { nxxN }

```

(End of definition for _zrefclever_get_rf_opt_tl:nnnN.)

```

\_zrefclever_get_rf_opt_seq:nnnN      \_zrefclever_get_rf_opt_seq:nnnN {<option>}
                                     {<ref type>} {<language>} {<seq variable>}
5258 \cs_new_protected:Npn \_zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5259 {
5260     % First attempt: general options.
5261     \_zrefclever_opt_seq_get:cNF
5262     { \_zrefclever_opt_varname_general:nn {#1} { seq } }
5263     #4
5264     {
5265         % If not found, try type specific options.
5266         \_zrefclever_opt_seq_get:cNF
5267         { \_zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5268         #4
5269         {
5270             % If not found, try type- and language-specific.
5271             \_zrefclever_opt_seq_get:cNF
5272             { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5273             #4
5274             {
5275                 % If not found, try language-specific default.
5276                 \_zrefclever_opt_seq_get:cNF
5277                 { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5278                 #4
5279                 {
5280                     % If not found, try fallback.
5281                     \_zrefclever_opt_seq_get:cNF
5282                     { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5283                     #4
5284                     { \seq_clear:N #4 }
5285                 }
5286             }
5287         }
5288     }
5289 }
5290 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_seq:nnnN { nxxN }

```

(End of definition for _zrefclever_get_rf_opt_seq:nnnN.)

```

\__zrefclever_get_rf_opt_bool:nnnnN
\__zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
  {<ref type>} {<language>} {<bool variable>}
5291 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5292 {
5293   % First attempt: general options.
5294   \__zrefclever_opt_bool_get:cNF
5295   { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5296   #5
5297   {
5298     % If not found, try type specific options.
5299     \__zrefclever_opt_bool_get:cNF
5300     { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5301     #5
5302     {
5303       % If not found, try type- and language-specific.
5304       \__zrefclever_opt_bool_get:cNF
5305       { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5306       #5
5307       {
5308         % If not found, try language-specific default.
5309         \__zrefclever_opt_bool_get:cNF
5310         { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5311         #5
5312         {
5313           % If not found, try fallback.
5314           \__zrefclever_opt_bool_get:cNF
5315           { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5316           #5
5317           { \use:c { bool_set_ #2 :N } #5 }
5318         }
5319       }
5320     }
5321   }
5322 }
5323 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nnnxN }

```

(End of definition for __zrefclever_get_rf_opt_bool:nnnnN.)

9 Compatibility

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

9.1 appendix

One relevant case of different reference types sharing the same counter is the `\appendix` which in some document classes, including the standard ones, change the sectioning commands looks but, of course, keep using the same counter. `book.cls` and `report.cls` reset counters `chapter` and `section` to 0, change `\@chapapp` to use `\appendixname` and use `\@Alph` for `\thechapter`. `article.cls` resets counters `section` and `subsection` to 0, and uses `\@Alph` for `\thesection`. `memoir.cls`, `scrbook.cls` and `scrarticle.cls`

do the same as their corresponding standard classes, and sometimes a little more, but what interests us here is pretty much the same. See also the `appendix` package.

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

```

5324 \__zrefclever_compat_module:nn { appendix }
5325 {
5326   \AddToHook { cmd / appendix / before }
5327   {
5328     \__zrefclever_zcsetup:n
5329     {
5330       countertype =
5331       {
5332         chapter      = appendix ,
5333         section      = appendix ,
5334         subsection   = appendix ,
5335         subsubsection = appendix ,
5336         paragraph    = appendix ,
5337         subparagraph = appendix ,
5338       }
5339     }
5340   }
5341 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmds` (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (`##`) the patch to add the hook, if it needs to be done with the `\scantokens` method, may fail noisily (see <https://tex.stackexchange.com/q/617905>, with a detailed explanation and possible workaround by Phelype Oleinik). The 2021-11-15 kernel release already handles this gracefully, thanks to fix by Phelype Oleinik at <https://github.com/latex3/latex2e/pull/699>.

9.2 appendices

This module applies both to the `appendix` package, and to the `memoir` class, since it “emulates” the package.

```

5342 \__zrefclever_compat_module:nn { appendices }
5343 {
5344   \__zrefclever_if_package_loaded:nT { appendix }
5345   {
5346     \newcounter { zc@appendix }
5347     \newcounter { zc@save@appendix }
5348     \setcounter { zc@appendix } { 0 }
5349     \setcounter { zc@save@appendix } { 0 }
5350     \cs_if_exist:cTF { chapter }
5351     {

```

```

5352         \__zrefclever_zcsetup:n
5353         { counterresetby = { chapter = zc@appendix } }
5354     }
5355     {
5356         \cs_if_exist:cT { section }
5357         {
5358             \__zrefclever_zcsetup:n
5359             { counterresetby = { section = zc@appendix } }
5360         }
5361     }
5362 \AddToHook { env / appendices / begin }
5363 {
5364     \stepcounter { zc@save@appendix }
5365     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5366     \__zrefclever_zcsetup:n
5367     {
5368         countertype =
5369         {
5370             chapter      = appendix ,
5371             section      = appendix ,
5372             subsection   = appendix ,
5373             subsubsection = appendix ,
5374             paragraph    = appendix ,
5375             subparagraph = appendix ,
5376         }
5377     }
5378 }
5379 \AddToHook { env / appendices / end }
5380 { \setcounter { zc@appendix } { 0 } }
5381 \AddToHook { cmd / appendix / before }
5382 {
5383     \stepcounter { zc@save@appendix }
5384     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5385 }
5386 \AddToHook { env / subappendices / begin }
5387 {
5388     \__zrefclever_zcsetup:n
5389     {
5390         countertype =
5391         {
5392             section      = appendix ,
5393             subsection   = appendix ,
5394             subsubsection = appendix ,
5395             paragraph    = appendix ,
5396             subparagraph = appendix ,
5397         } ,
5398     }
5399 }
5400 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5401 }
5402 }

```


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. Fortunately, however, the kernel's new `label` hook, and the `labelhook` option, make things somewhat easier.

1. The `\subcaptionref` command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside the `\subcaption` argument of the subcaptioning commands, namely: `\subcaption`, `\contsubcaption`, `\subbottom`, `\contsubbottom`, `\subtop`. This functionality is implemented by `memoir` by setting a *second label* with prefix `sub@{label}`, and storing there just that part of interest. With `zref` this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find a place to hook into to add the property to the `main` list, since `memoir` does not really consider the possibility of some other command setting labels. `\@memsubcaption` is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.
2. `memoir`'s `\footnote`, `\verbfootnote`, `\sidefootnote` and `\pagenote`, just as the regular `\footnote` until recently in the kernel, do not set `\@currentcounter` alongside `\@currentlabel`, proper referencing to them requires setting the type for it.
3. Note that `memoir`'s appendix features “emulates” the `appendix` package, hence the corresponding compatibility module is loaded for `memoir` even if that package is not itself loaded. The same is true for the `\appendix` command module, since it is also defined.

```

5403 \__zrefclever_compat_module:nn { memoir }
5404 {
5405   \__zrefclever_if_class_loaded:nT { memoir }
5406   {

```

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

```

5407   \__zrefclever_zcsetup:n
5408   {
5409     countertype =
5410     {
5411       subfigure = figure ,
5412       subtable  = table  ,
5413       poemline  = line   ,
5414     } ,
5415     counterresetby =
5416     {
5417       subfigure = figure ,
5418       subtable  = table  ,
5419     } ,
5420   }

```

Support for subcaption reference.

```

5421 \zref@newprop { subcaption }
5422 { \cs_if_exist_use:c { @@thesub \@capytype } }
5423 \AddToHook{ cmd / @memsubcaption / before }
5424 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5425 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5426 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5427 \AddToHook { env / minipage / begin }
5428 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5429 \AddToHook { cmd / @makefntext / before }
5430 {
5431   \__zrefclever_zcsetup:x
5432   { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5433 }
5434 \AddToHook { cmd / @makesidefntext / before }
5435 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5436 \__zrefclever_zcsetup:n
5437 {
5438   countertype =
5439   {
5440     sidefootnote = footnote ,
5441     pagenote = endnote ,
5442   } ,
5443 }
5444 \AddToHook { file / \jobname.ent / before }
5445 { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5446 \msg_info:nnn { zref-clever } { compat-class } { memoir }
5447 }
5448 }

```

9.4 amsmath

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

```

5449 \__zrefclever_compat_module:nn { amsmath }
5450 {
5451   \__zrefclever_if_package_loaded:nT { amsmath }
5452   {

```

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).

```

5453   \bool_new:N \l__zrefclever_amsmath_subequations_bool

```

```

5454 \AddToHook { env / subequations / begin }
5455 {
5456   \__zrefclever_zcsetup:x
5457   {
5458     counterresetby =
5459     {
5460       parentequation =
5461       \__zrefclever_counter_reset_by:n { equation } ,
5462       equation = parentequation ,
5463     } ,
5464     currentcounter = parentequation ,
5465     countertype = { parentequation = equation } ,
5466   }
5467   \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5468 }

```

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.

```

5469 \zref@newprop { subeq } { \alph { equation } }
5470 \clist_map_inline:nn
5471 {
5472   equation ,
5473   equation* ,
5474   align ,
5475   align* ,
5476   alignat ,
5477   alignat* ,
5478   flalign ,
5479   flalign* ,
5480   xalignat ,
5481   xalignat* ,
5482   gather ,
5483   gather* ,
5484   multline ,
5485   multline* ,
5486 }
5487 {
5488   \AddToHook { env / #1 / begin }
5489   {
5490     \__zrefclever_zcsetup:n { currentcounter = equation }
5491     \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5492     { \zref@localaddprop \ZREF@mainlist { subeq } }
5493   }
5494 }

```

```

5495         \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5496     }
5497 }

```

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.

```

5498 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5499 \__zrefclever_compat_module:nn { mathtools }
5500 {
5501     \__zrefclever_if_package_loaded:nT { mathtools }
5502     {
5503         \MH_if_boolean:nT { show_only_refs }
5504         {
5505             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5506             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5507             {
5508                 \@bsphack
5509                 \seq_map_inline:Nn #1
5510                 {
5511                     \exp_args:Nx \tl_if_eq:nnTF
5512                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5513                     { equation }
5514                     {
5515                         \protected@write \@auxout { }
5516                         { \string \MT@newlabel {##1} }
5517                     }
5518                 }
5519                 \exp_args:Nx \tl_if_eq:nnT
5520                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5521                 { parentequation }
5522                 {
5523                     \protected@write \@auxout { }
5524                     { \string \MT@newlabel {##1} }
5525                 }
5526             }
5527         }
5528         \@esphack
5529     }
5530     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5531 }
5532 }
5533 }

```

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.

```

5534 \__zrefclever_compat_module:nn { breqn }
5535 {
5536   \__zrefclever_if_package_loaded:NT { breqn }
5537   {

```

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>).

```

5538   \bool_new:N \l__zrefclever_breqn_dgroup_bool
5539   \AddToHook { env / dgroup / begin }
5540   {
5541     \__zrefclever_zcsetup:x
5542     {
5543       counterresetby =
5544       {
5545         parentequation =
5546         \__zrefclever_counter_reset_by:n { equation } ,
5547         equation = parentequation ,
5548       } ,
5549       currentcounter = parentequation ,
5550       countertype = { parentequation = equation } ,
5551     }
5552     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5553   }
5554   \zref@ifpropundefined { subeq }
5555   { \zref@newprop { subeq } { \alph { equation } } }
5556   { }
5557   \clist_map_inline:nn
5558   {
5559     dmath ,
5560     dseries ,
5561     darray ,
5562   }
5563   {
5564     \AddToHook { env / #1 / begin }
5565     {
5566       \__zrefclever_zcsetup:n { currentcounter = equation }
5567       \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5568       { \zref@localaddprop \ZREF@mainlist { subeq } }
5569     }
5570   }
5571   \msg_info:nnn { zref-clever } { compat-package } { breqn }
5572 }
5573 }

```

9.7 listings

```

5574 \__zrefclever_compat_module:nn { listings }
5575 {
5576   \__zrefclever_if_package_loaded:nT { listings }
5577   {
5578     \__zrefclever_zcsetup:n
5579     {
5580       countertype =
5581       {
5582         lstlisting = listing ,
5583         lstnumber = line ,
5584       } ,
5585       counterresetby = { lstnumber = lstlisting } ,
5586     }

```

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.

```

5587   \lst@AddToHook { Init }
5588   { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5589   \msg_info:nnn { zref-clever } { compat-package } { listings }
5590 }
5591 }

```

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.

```

5592 \__zrefclever_compat_module:nn { enumitem }
5593 {
5594   \__zrefclever_if_package_loaded:nT { enumitem }
5595   {
5596     \int_set:Nn \l_tmpa_int { 5 }
5597     \bool_while_do:nn
5598     {
5599       \cs_if_exist_p:c
5600       { c@ enum \int_to_roman:n { \l_tmpa_int } }

```

```

5601     }
5602     {
5603         \__zrefclever_zcsetup:x
5604         {
5605             counterresetby =
5606             {
5607                 enum \int_to_roman:n { \l_tmpa_int } =
5608                 enum \int_to_roman:n { \l_tmpa_int - 1 }
5609             } ,
5610             countertype =
5611             { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5612         }
5613         \int_incr:N \l_tmpa_int
5614     }
5615     \int_compare:nNnT { \l_tmpa_int } > { 5 }
5616     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5617 }
5618 }

```

9.9 subcaption

```

5619 \__zrefclever_compat_module:nn { subcaption }
5620 {
5621     \__zrefclever_if_package_loaded:nT { subcaption }
5622     {
5623         \__zrefclever_zcsetup:n
5624         {
5625             countertype =
5626             {
5627                 subfigure = figure ,
5628                 subtable = table ,
5629             } ,
5630             counterresetby =
5631             {
5632                 subfigure = figure ,
5633                 subtable = table ,
5634             } ,
5635         }

```

Support for subref reference.

```

5636     \zref@newprop { subref }
5637     { \cs_if_exist_use:c { thesub \@captype } }
5638     \tl_put_right:Nn \caption@subtypehook
5639     { \zref@localaddprop \ZREF@mainlist { subref } }
5640 }
5641 }

```

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.

```

5642 \__zrefclever_compat_module:nn { subfig }
5643 {
5644     \__zrefclever_if_package_loaded:nT { subfig }
5645     {

```

```

5646 \__zrefclever_zcsetup:n
5647 {
5648     countertype =
5649     {
5650         subfigure = figure ,
5651         subtable = table ,
5652     } ,
5653     counterresetby =
5654     {
5655         subfigure = figure ,
5656         subtable = table ,
5657     } ,
5658 }
5659 }
5660 }
5661 </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” `refbounds` options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that’s the point where we know from `babel` or `polyglossia` the `\language` name. But we also don’t want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

zref-vario: If you are interested in the localization of `zref-clever` to your language, and willing to contribute to it, you may also want to consider doing the same for the

companion package `zref-vario`. It is actually a much simpler task than localizing `zref-clever`.

10.2 English

English language file has been initially provided by the author.

```

5662 <*package>
5663 \zcDeclareLanguage { english }
5664 \zcDeclareLanguageAlias { american } { english }
5665 \zcDeclareLanguageAlias { australian } { english }
5666 \zcDeclareLanguageAlias { british } { english }
5667 \zcDeclareLanguageAlias { canadian } { english }
5668 \zcDeclareLanguageAlias { newzealand } { english }
5669 \zcDeclareLanguageAlias { UKenglish } { english }
5670 \zcDeclareLanguageAlias { USenglish } { english }
5671 </package>

5672 <*lang-english>

5673 namesep = {\nobreakspace} ,
5674 pairsep = {\~and\nobreakspace} ,
5675 listsep = {,~} ,
5676 lastsep = {\~and\nobreakspace} ,
5677 tpairsep = {\~and\nobreakspace} ,
5678 tlistsep = {,~} ,
5679 tlastsep = {\~and\nobreakspace} ,
5680 notesep = {\~} ,
5681 rangesep = {\~to\nobreakspace} ,
5682
5683 type = book ,
5684   Name-sg = Book ,
5685   name-sg = book ,
5686   Name-pl = Books ,
5687   name-pl = books ,
5688
5689 type = part ,
5690   Name-sg = Part ,
5691   name-sg = part ,
5692   Name-pl = Parts ,
5693   name-pl = parts ,
5694
5695 type = chapter ,
5696   Name-sg = Chapter ,
5697   name-sg = chapter ,
5698   Name-pl = Chapters ,
5699   name-pl = chapters ,
5700
5701 type = section ,
5702   Name-sg = Section ,
5703   name-sg = section ,
5704   Name-pl = Sections ,
5705   name-pl = sections ,
5706
5707 type = paragraph ,
5708   Name-sg = Paragraph ,

```

```

5709     name-sg = paragraph ,
5710     Name-pl = Paragraphs ,
5711     name-pl = paragraphs ,
5712     Name-sg-ab = Par. ,
5713     name-sg-ab = par. ,
5714     Name-pl-ab = Par. ,
5715     name-pl-ab = par. ,
5716
5717     type = appendix ,
5718     Name-sg = Appendix ,
5719     name-sg = appendix ,
5720     Name-pl = Appendices ,
5721     name-pl = appendices ,
5722
5723     type = page ,
5724     Name-sg = Page ,
5725     name-sg = page ,
5726     Name-pl = Pages ,
5727     name-pl = pages ,
5728     rangesep = {\textendash} ,
5729     rangetopair = false ,
5730
5731     type = line ,
5732     Name-sg = Line ,
5733     name-sg = line ,
5734     Name-pl = Lines ,
5735     name-pl = lines ,
5736
5737     type = figure ,
5738     Name-sg = Figure ,
5739     name-sg = figure ,
5740     Name-pl = Figures ,
5741     name-pl = figures ,
5742     Name-sg-ab = Fig. ,
5743     name-sg-ab = fig. ,
5744     Name-pl-ab = Figs. ,
5745     name-pl-ab = figs. ,
5746
5747     type = table ,
5748     Name-sg = Table ,
5749     name-sg = table ,
5750     Name-pl = Tables ,
5751     name-pl = tables ,
5752
5753     type = item ,
5754     Name-sg = Item ,
5755     name-sg = item ,
5756     Name-pl = Items ,
5757     name-pl = items ,
5758
5759     type = footnote ,
5760     Name-sg = Footnote ,
5761     name-sg = footnote ,
5762     Name-pl = Footnotes ,

```

```

5763     name-pl = footnotes ,
5764
5765 type = endnote ,
5766     Name-sg = Note ,
5767     name-sg = note ,
5768     Name-pl = Notes ,
5769     name-pl = notes ,
5770
5771 type = note ,
5772     Name-sg = Note ,
5773     name-sg = note ,
5774     Name-pl = Notes ,
5775     name-pl = notes ,
5776
5777 type = equation ,
5778     Name-sg = Equation ,
5779     name-sg = equation ,
5780     Name-pl = Equations ,
5781     name-pl = equations ,
5782     Name-sg-ab = Eq. ,
5783     name-sg-ab = eq. ,
5784     Name-pl-ab = Eqs. ,
5785     name-pl-ab = eqs. ,
5786     refbounds-first-sg = {,(,),} ,
5787     refbounds = {(,,)} ,
5788
5789 type = theorem ,
5790     Name-sg = Theorem ,
5791     name-sg = theorem ,
5792     Name-pl = Theorems ,
5793     name-pl = theorems ,
5794
5795 type = lemma ,
5796     Name-sg = Lemma ,
5797     name-sg = lemma ,
5798     Name-pl = Lemmas ,
5799     name-pl = lemmas ,
5800
5801 type = corollary ,
5802     Name-sg = Corollary ,
5803     name-sg = corollary ,
5804     Name-pl = Corollaries ,
5805     name-pl = corollaries ,
5806
5807 type = proposition ,
5808     Name-sg = Proposition ,
5809     name-sg = proposition ,
5810     Name-pl = Propositions ,
5811     name-pl = propositions ,
5812
5813 type = definition ,
5814     Name-sg = Definition ,
5815     name-sg = definition ,
5816     Name-pl = Definitions ,

```

```

5817   name-pl = definitions ,
5818
5819   type = proof ,
5820     Name-sg = Proof ,
5821     name-sg = proof ,
5822     Name-pl = Proofs ,
5823     name-pl = proofs ,
5824
5825   type = result ,
5826     Name-sg = Result ,
5827     name-sg = result ,
5828     Name-pl = Results ,
5829     name-pl = results ,
5830
5831   type = remark ,
5832     Name-sg = Remark ,
5833     name-sg = remark ,
5834     Name-pl = Remarks ,
5835     name-pl = remarks ,
5836
5837   type = example ,
5838     Name-sg = Example ,
5839     name-sg = example ,
5840     Name-pl = Examples ,
5841     name-pl = examples ,
5842
5843   type = algorithm ,
5844     Name-sg = Algorithm ,
5845     name-sg = algorithm ,
5846     Name-pl = Algorithms ,
5847     name-pl = algorithms ,
5848
5849   type = listing ,
5850     Name-sg = Listing ,
5851     name-sg = listing ,
5852     Name-pl = Listings ,
5853     name-pl = listings ,
5854
5855   type = exercise ,
5856     Name-sg = Exercise ,
5857     name-sg = exercise ,
5858     Name-pl = Exercises ,
5859     name-pl = exercises ,
5860
5861   type = solution ,
5862     Name-sg = Solution ,
5863     name-sg = solution ,
5864     Name-pl = Solutions ,
5865     name-pl = solutions ,
5866 </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`.

```

5867 <*package>
5868 \zcDeclareLanguage
5869 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5870 { german }
5871 \zcDeclareLanguageAlias { ngerman      } { german }
5872 \zcDeclareLanguageAlias { austrian     } { german }
5873 \zcDeclareLanguageAlias { naustrian    } { german }
5874 \zcDeclareLanguageAlias { swissgerman  } { german }
5875 \zcDeclareLanguageAlias { nswissgerman } { german }
5876 </package>
5877 <*lang-german>
5878 namesep = {\nobreakspace} ,
5879 pairsep = {\und\nobreakspace} ,
5880 listsep = {,~} ,
5881 lastsep = {\und\nobreakspace} ,
5882 tpairsep = {\und\nobreakspace} ,
5883 tlistsep = {,~} ,
5884 tlastsep = {\und\nobreakspace} ,
5885 notesep = {~} ,
5886 rangesep = {\bis\nobreakspace} ,
5887
5888 type = book ,
5889 gender = n ,
5890 case = N ,
5891   Name-sg = Buch ,
5892   Name-pl = Bücher ,
5893 case = A ,
5894   Name-sg = Buch ,
5895   Name-pl = Bücher ,
5896 case = D ,
5897   Name-sg = Buch ,
5898   Name-pl = Büchern ,
5899 case = G ,
5900   Name-sg = Buches ,
5901   Name-pl = Bücher ,
5902
5903 type = part ,
5904 gender = m ,
5905 case = N ,
5906   Name-sg = Teil ,
5907   Name-pl = Teile ,
5908 case = A ,
5909   Name-sg = Teil ,
5910   Name-pl = Teile ,
5911 case = D ,
5912   Name-sg = Teil ,
5913   Name-pl = Teilen ,
5914 case = G ,
5915   Name-sg = Teiles ,
5916   Name-pl = Teile ,
5917

```

```

5918 type = chapter ,
5919     gender = n ,
5920     case = N ,
5921         Name-sg = Kapitel ,
5922         Name-pl = Kapitel ,
5923     case = A ,
5924         Name-sg = Kapitel ,
5925         Name-pl = Kapitel ,
5926     case = D ,
5927         Name-sg = Kapitel ,
5928         Name-pl = Kapiteln ,
5929     case = G ,
5930         Name-sg = Kapitals ,
5931         Name-pl = Kapitel ,
5932
5933 type = section ,
5934     gender = m ,
5935     case = N ,
5936         Name-sg = Abschnitt ,
5937         Name-pl = Abschnitte ,
5938     case = A ,
5939         Name-sg = Abschnitt ,
5940         Name-pl = Abschnitte ,
5941     case = D ,
5942         Name-sg = Abschnitt ,
5943         Name-pl = Abschnitten ,
5944     case = G ,
5945         Name-sg = Abschnitts ,
5946         Name-pl = Abschnitte ,
5947
5948 type = paragraph ,
5949     gender = m ,
5950     case = N ,
5951         Name-sg = Absatz ,
5952         Name-pl = Absätze ,
5953     case = A ,
5954         Name-sg = Absatz ,
5955         Name-pl = Absätze ,
5956     case = D ,
5957         Name-sg = Absatz ,
5958         Name-pl = Absätzen ,
5959     case = G ,
5960         Name-sg = Absatzes ,
5961         Name-pl = Absätze ,
5962
5963 type = appendix ,
5964     gender = m ,
5965     case = N ,
5966         Name-sg = Anhang ,
5967         Name-pl = Anhänge ,
5968     case = A ,
5969         Name-sg = Anhang ,
5970         Name-pl = Anhänge ,
5971     case = D ,

```



```

5972     Name-sg = Anhang ,
5973     Name-pl = Anhängen ,
5974     case = G ,
5975     Name-sg = Anhangs ,
5976     Name-pl = Anhänge ,
5977
5978 type = page ,
5979     gender = f ,
5980     case = N ,
5981     Name-sg = Seite ,
5982     Name-pl = Seiten ,
5983     case = A ,
5984     Name-sg = Seite ,
5985     Name-pl = Seiten ,
5986     case = D ,
5987     Name-sg = Seite ,
5988     Name-pl = Seiten ,
5989     case = G ,
5990     Name-sg = Seite ,
5991     Name-pl = Seiten ,
5992     rangesep = {\textendash} ,
5993     rangetopair = false ,
5994
5995 type = line ,
5996     gender = f ,
5997     case = N ,
5998     Name-sg = Zeile ,
5999     Name-pl = Zeilen ,
6000     case = A ,
6001     Name-sg = Zeile ,
6002     Name-pl = Zeilen ,
6003     case = D ,
6004     Name-sg = Zeile ,
6005     Name-pl = Zeilen ,
6006     case = G ,
6007     Name-sg = Zeile ,
6008     Name-pl = Zeilen ,
6009
6010 type = figure ,
6011     gender = f ,
6012     case = N ,
6013     Name-sg = Abbildung ,
6014     Name-pl = Abbildungen ,
6015     Name-sg-ab = Abb. ,
6016     Name-pl-ab = Abb. ,
6017     case = A ,
6018     Name-sg = Abbildung ,
6019     Name-pl = Abbildungen ,
6020     Name-sg-ab = Abb. ,
6021     Name-pl-ab = Abb. ,
6022     case = D ,
6023     Name-sg = Abbildung ,
6024     Name-pl = Abbildungen ,
6025     Name-sg-ab = Abb. ,

```

```

6026     Name-pl-ab = Abb. ,
6027     case = G ,
6028     Name-sg = Abbildung ,
6029     Name-pl = Abbildungen ,
6030     Name-sg-ab = Abb. ,
6031     Name-pl-ab = Abb. ,
6032
6033 type = table ,
6034     gender = f ,
6035     case = N ,
6036     Name-sg = Tabelle ,
6037     Name-pl = Tabellen ,
6038     case = A ,
6039     Name-sg = Tabelle ,
6040     Name-pl = Tabellen ,
6041     case = D ,
6042     Name-sg = Tabelle ,
6043     Name-pl = Tabellen ,
6044     case = G ,
6045     Name-sg = Tabelle ,
6046     Name-pl = Tabellen ,
6047
6048 type = item ,
6049     gender = m ,
6050     case = N ,
6051     Name-sg = Punkt ,
6052     Name-pl = Punkte ,
6053     case = A ,
6054     Name-sg = Punkt ,
6055     Name-pl = Punkte ,
6056     case = D ,
6057     Name-sg = Punkt ,
6058     Name-pl = Punkten ,
6059     case = G ,
6060     Name-sg = Punktes ,
6061     Name-pl = Punkte ,
6062
6063 type = footnote ,
6064     gender = f ,
6065     case = N ,
6066     Name-sg = Fußnote ,
6067     Name-pl = Fußnoten ,
6068     case = A ,
6069     Name-sg = Fußnote ,
6070     Name-pl = Fußnoten ,
6071     case = D ,
6072     Name-sg = Fußnote ,
6073     Name-pl = Fußnoten ,
6074     case = G ,
6075     Name-sg = Fußnote ,
6076     Name-pl = Fußnoten ,
6077
6078 type = endnote ,
6079     gender = f ,

```

```

6080 case = N ,
6081     Name-sg = Endnote ,
6082     Name-pl = Endnoten ,
6083 case = A ,
6084     Name-sg = Endnote ,
6085     Name-pl = Endnoten ,
6086 case = D ,
6087     Name-sg = Endnote ,
6088     Name-pl = Endnoten ,
6089 case = G ,
6090     Name-sg = Endnote ,
6091     Name-pl = Endnoten ,
6092
6093 type = note ,
6094     gender = f ,
6095     case = N ,
6096         Name-sg = Anmerkung ,
6097         Name-pl = Anmerkungen ,
6098     case = A ,
6099         Name-sg = Anmerkung ,
6100         Name-pl = Anmerkungen ,
6101     case = D ,
6102         Name-sg = Anmerkung ,
6103         Name-pl = Anmerkungen ,
6104     case = G ,
6105         Name-sg = Anmerkung ,
6106         Name-pl = Anmerkungen ,
6107
6108 type = equation ,
6109     gender = f ,
6110     case = N ,
6111         Name-sg = Gleichung ,
6112         Name-pl = Gleichungen ,
6113     case = A ,
6114         Name-sg = Gleichung ,
6115         Name-pl = Gleichungen ,
6116     case = D ,
6117         Name-sg = Gleichung ,
6118         Name-pl = Gleichungen ,
6119     case = G ,
6120         Name-sg = Gleichung ,
6121         Name-pl = Gleichungen ,
6122     refbounds-first-sg = {,(,),} ,
6123     refbounds = {(,,)} ,
6124
6125 type = theorem ,
6126     gender = n ,
6127     case = N ,
6128         Name-sg = Theorem ,
6129         Name-pl = Theoreme ,
6130     case = A ,
6131         Name-sg = Theorem ,
6132         Name-pl = Theoreme ,
6133     case = D ,

```

```

6134     Name-sg = Theorem ,
6135     Name-pl = Theoremen ,
6136     case = G ,
6137     Name-sg = Theorems ,
6138     Name-pl = Theoreme ,
6139
6140 type = lemma ,
6141     gender = n ,
6142     case = N ,
6143     Name-sg = Lemma ,
6144     Name-pl = Lemmata ,
6145     case = A ,
6146     Name-sg = Lemma ,
6147     Name-pl = Lemmata ,
6148     case = D ,
6149     Name-sg = Lemma ,
6150     Name-pl = Lemmata ,
6151     case = G ,
6152     Name-sg = Lemmas ,
6153     Name-pl = Lemmata ,
6154
6155 type = corollary ,
6156     gender = n ,
6157     case = N ,
6158     Name-sg = Korollar ,
6159     Name-pl = Korollare ,
6160     case = A ,
6161     Name-sg = Korollar ,
6162     Name-pl = Korollare ,
6163     case = D ,
6164     Name-sg = Korollar ,
6165     Name-pl = Korollaren ,
6166     case = G ,
6167     Name-sg = Korollars ,
6168     Name-pl = Korollare ,
6169
6170 type = proposition ,
6171     gender = m ,
6172     case = N ,
6173     Name-sg = Satz ,
6174     Name-pl = Sätze ,
6175     case = A ,
6176     Name-sg = Satz ,
6177     Name-pl = Sätze ,
6178     case = D ,
6179     Name-sg = Satz ,
6180     Name-pl = Sätzen ,
6181     case = G ,
6182     Name-sg = Satzes ,
6183     Name-pl = Sätze ,
6184
6185 type = definition ,
6186     gender = f ,
6187     case = N ,

```

```

6188     Name-sg = Definition ,
6189     Name-pl = Definitionen ,
6190     case = A ,
6191     Name-sg = Definition ,
6192     Name-pl = Definitionen ,
6193     case = D ,
6194     Name-sg = Definition ,
6195     Name-pl = Definitionen ,
6196     case = G ,
6197     Name-sg = Definition ,
6198     Name-pl = Definitionen ,
6199
6200 type = proof ,
6201     gender = m ,
6202     case = N ,
6203     Name-sg = Beweis ,
6204     Name-pl = Beweise ,
6205     case = A ,
6206     Name-sg = Beweis ,
6207     Name-pl = Beweise ,
6208     case = D ,
6209     Name-sg = Beweis ,
6210     Name-pl = Beweisen ,
6211     case = G ,
6212     Name-sg = Beweises ,
6213     Name-pl = Beweise ,
6214
6215 type = result ,
6216     gender = n ,
6217     case = N ,
6218     Name-sg = Ergebnis ,
6219     Name-pl = Ergebnisse ,
6220     case = A ,
6221     Name-sg = Ergebnis ,
6222     Name-pl = Ergebnisse ,
6223     case = D ,
6224     Name-sg = Ergebnis ,
6225     Name-pl = Ergebnissen ,
6226     case = G ,
6227     Name-sg = Ergebnisses ,
6228     Name-pl = Ergebnisse ,
6229
6230 type = remark ,
6231     gender = f ,
6232     case = N ,
6233     Name-sg = Bemerkung ,
6234     Name-pl = Bemerkungen ,
6235     case = A ,
6236     Name-sg = Bemerkung ,
6237     Name-pl = Bemerkungen ,
6238     case = D ,
6239     Name-sg = Bemerkung ,
6240     Name-pl = Bemerkungen ,
6241     case = G ,

```

```

6242     Name-sg = Bemerkung ,
6243     Name-pl = Bemerkungen ,
6244
6245 type = example ,
6246     gender = n ,
6247     case = N ,
6248     Name-sg = Beispiel ,
6249     Name-pl = Beispiele ,
6250     case = A ,
6251     Name-sg = Beispiel ,
6252     Name-pl = Beispiele ,
6253     case = D ,
6254     Name-sg = Beispiel ,
6255     Name-pl = Beispielen ,
6256     case = G ,
6257     Name-sg = Beispiels ,
6258     Name-pl = Beispiele ,
6259
6260 type = algorithm ,
6261     gender = m ,
6262     case = N ,
6263     Name-sg = Algorithmus ,
6264     Name-pl = Algorithmen ,
6265     case = A ,
6266     Name-sg = Algorithmus ,
6267     Name-pl = Algorithmen ,
6268     case = D ,
6269     Name-sg = Algorithmus ,
6270     Name-pl = Algorithmen ,
6271     case = G ,
6272     Name-sg = Algorithmus ,
6273     Name-pl = Algorithmen ,
6274
6275 type = listing ,
6276     gender = n ,
6277     case = N ,
6278     Name-sg = Listing ,
6279     Name-pl = Listings ,
6280     case = A ,
6281     Name-sg = Listing ,
6282     Name-pl = Listings ,
6283     case = D ,
6284     Name-sg = Listing ,
6285     Name-pl = Listings ,
6286     case = G ,
6287     Name-sg = Listings ,
6288     Name-pl = Listings ,
6289
6290 type = exercise ,
6291     gender = f ,
6292     case = N ,
6293     Name-sg = Übungsaufgabe ,
6294     Name-pl = Übungsaufgaben ,
6295     case = A ,

```

```

6296     Name-sg = Übungsaufgabe ,
6297     Name-pl = Übungsaufgaben ,
6298     case = D ,
6299     Name-sg = Übungsaufgabe ,
6300     Name-pl = Übungsaufgaben ,
6301     case = G ,
6302     Name-sg = Übungsaufgabe ,
6303     Name-pl = Übungsaufgaben ,
6304
6305     type = solution ,
6306     gender = f ,
6307     case = N ,
6308     Name-sg = Lösung ,
6309     Name-pl = Lösungen ,
6310     case = A ,
6311     Name-sg = Lösung ,
6312     Name-pl = Lösungen ,
6313     case = D ,
6314     Name-sg = Lösung ,
6315     Name-pl = Lösungen ,
6316     case = G ,
6317     Name-sg = Lösung ,
6318     Name-pl = Lösungen ,
6319 </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 `francais`, `frenchb`, and `canadien`, but they are deprecated as options and, if used, they fall back to either `french` or `acadian`.

```

6320 <*package>
6321 \zcDeclareLanguage [ gender = { f , m } ] { french }
6322 \zcDeclareLanguageAlias { acadian } { french }
6323 </package>
6324 <*lang-french>
6325 namesep = {\nobreakspace} ,
6326 pairsep = {\~et\nobreakspace} ,
6327 listsep = { , ~ } ,
6328 lastsep = {\~et\nobreakspace} ,
6329 tpairsep = {\~et\nobreakspace} ,
6330 tlistsep = { , ~ } ,
6331 tlastsep = {\~et\nobreakspace} ,
6332 notesep = { ~ } ,
6333 rangesep = {\~à\nobreakspace} ,
6334
6335 type = book ,
6336 gender = m ,
6337 Name-sg = Livre ,

```

```

6338     name-sg = livre ,
6339     Name-pl = Livres ,
6340     name-pl = livres ,
6341
6342     type = part ,
6343     gender = f ,
6344     Name-sg = Partie ,
6345     name-sg = partie ,
6346     Name-pl = Parties ,
6347     name-pl = parties ,
6348
6349     type = chapter ,
6350     gender = m ,
6351     Name-sg = Chapitre ,
6352     name-sg = chapitre ,
6353     Name-pl = Chapitres ,
6354     name-pl = chapitres ,
6355
6356     type = section ,
6357     gender = f ,
6358     Name-sg = Section ,
6359     name-sg = section ,
6360     Name-pl = Sections ,
6361     name-pl = sections ,
6362
6363     type = paragraph ,
6364     gender = m ,
6365     Name-sg = Paragraphe ,
6366     name-sg = paragraphe ,
6367     Name-pl = Paragraphes ,
6368     name-pl = paragraphes ,
6369
6370     type = appendix ,
6371     gender = f ,
6372     Name-sg = Annexe ,
6373     name-sg = annexe ,
6374     Name-pl = Annexes ,
6375     name-pl = annexes ,
6376
6377     type = page ,
6378     gender = f ,
6379     Name-sg = Page ,
6380     name-sg = page ,
6381     Name-pl = Pages ,
6382     name-pl = pages ,
6383     rangesep = {-} ,
6384     rangetopair = false ,
6385
6386     type = line ,
6387     gender = f ,
6388     Name-sg = Ligne ,
6389     name-sg = ligne ,
6390     Name-pl = Lignes ,
6391     name-pl = lignes ,

```



```

6392
6393 type = figure ,
6394     gender = f ,
6395     Name-sg = Figure ,
6396     name-sg = figure ,
6397     Name-pl = Figures ,
6398     name-pl = figures ,
6399
6400 type = table ,
6401     gender = f ,
6402     Name-sg = Table ,
6403     name-sg = table ,
6404     Name-pl = Tables ,
6405     name-pl = tables ,
6406
6407 type = item ,
6408     gender = m ,
6409     Name-sg = Point ,
6410     name-sg = point ,
6411     Name-pl = Points ,
6412     name-pl = points ,
6413
6414 type = footnote ,
6415     gender = f ,
6416     Name-sg = Note ,
6417     name-sg = note ,
6418     Name-pl = Notes ,
6419     name-pl = notes ,
6420
6421 type = endnote ,
6422     gender = f ,
6423     Name-sg = Note ,
6424     name-sg = note ,
6425     Name-pl = Notes ,
6426     name-pl = notes ,
6427
6428 type = note ,
6429     gender = f ,
6430     Name-sg = Note ,
6431     name-sg = note ,
6432     Name-pl = Notes ,
6433     name-pl = notes ,
6434
6435 type = equation ,
6436     gender = f ,
6437     Name-sg = Équation ,
6438     name-sg = équation ,
6439     Name-pl = Équations ,
6440     name-pl = équations ,
6441     refbounds-first-sg = {(,),(,)} ,
6442     refbounds = {(,,)} ,
6443
6444 type = theorem ,
6445     gender = m ,

```

```

6446     Name-sg = Théorème ,
6447     name-sg = théorème ,
6448     Name-pl = Théorèmes ,
6449     name-pl = théorèmes ,
6450
6451     type = lemma ,
6452     gender = m ,
6453     Name-sg = Lemme ,
6454     name-sg = lemme ,
6455     Name-pl = Lemmes ,
6456     name-pl = lemmes ,
6457
6458     type = corollary ,
6459     gender = m ,
6460     Name-sg = Corollaire ,
6461     name-sg = corollaire ,
6462     Name-pl = Corollaires ,
6463     name-pl = corollaires ,
6464
6465     type = proposition ,
6466     gender = f ,
6467     Name-sg = Proposition ,
6468     name-sg = proposition ,
6469     Name-pl = Propositions ,
6470     name-pl = propositions ,
6471
6472     type = definition ,
6473     gender = f ,
6474     Name-sg = Définition ,
6475     name-sg = définition ,
6476     Name-pl = Définitions ,
6477     name-pl = définitions ,
6478
6479     type = proof ,
6480     gender = f ,
6481     Name-sg = Démonstration ,
6482     name-sg = démonstration ,
6483     Name-pl = Démonstrations ,
6484     name-pl = démonstrations ,
6485
6486     type = result ,
6487     gender = m ,
6488     Name-sg = Résultat ,
6489     name-sg = résultat ,
6490     Name-pl = Résultats ,
6491     name-pl = résultats ,
6492
6493     type = remark ,
6494     gender = f ,
6495     Name-sg = Remarque ,
6496     name-sg = remarque ,
6497     Name-pl = Remarques ,
6498     name-pl = remarques ,
6499

```

```

6500 type = example ,
6501   gender = m ,
6502   Name-sg = Exemple ,
6503   name-sg = exemple ,
6504   Name-pl = Exemples ,
6505   name-pl = exemples ,
6506
6507 type = algorithm ,
6508   gender = m ,
6509   Name-sg = Algorithme ,
6510   name-sg = algorithme ,
6511   Name-pl = Algorithmes ,
6512   name-pl = algorithmes ,
6513
6514 type = listing ,
6515   gender = m ,
6516   Name-sg = Listing ,
6517   name-sg = listing ,
6518   Name-pl = Listings ,
6519   name-pl = listings ,
6520
6521 type = exercise ,
6522   gender = m ,
6523   Name-sg = Exercice ,
6524   name-sg = exercice ,
6525   Name-pl = Exercices ,
6526   name-pl = exercices ,
6527
6528 type = solution ,
6529   gender = f ,
6530   Name-sg = Solution ,
6531   name-sg = solution ,
6532   Name-pl = Solutions ,
6533   name-pl = solutions ,
6534 </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.

```

6535 <*package>
6536 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6537 \zcDeclareLanguageAlias { brazilian } { portuguese }
6538 \zcDeclareLanguageAlias { brazil   } { portuguese }
6539 \zcDeclareLanguageAlias { portuges  } { portuguese }
6540 </package>
6541 <*lang-portuguese>
6542 namesep = {\nobreakspace} ,
6543 pairsep  = {\sim\nobreakspace} ,
6544 listsep  = { ,\sim } ,
6545 lastsep  = {\sim\nobreakspace} ,

```

```

6546 tpairsep = {\~e\nobreakspace} ,
6547 tlistsep = {,~} ,
6548 tlastsep = {\~e\nobreakspace} ,
6549 notesep = {\~} ,
6550 rangesep = {\~a\nobreakspace} ,
6551
6552 type = book ,
6553     gender = m ,
6554     Name-sg = Livro ,
6555     name-sg = livro ,
6556     Name-pl = Livros ,
6557     name-pl = livros ,
6558
6559 type = part ,
6560     gender = f ,
6561     Name-sg = Parte ,
6562     name-sg = parte ,
6563     Name-pl = Partes ,
6564     name-pl = partes ,
6565
6566 type = chapter ,
6567     gender = m ,
6568     Name-sg = Capítulo ,
6569     name-sg = capítulo ,
6570     Name-pl = Capítulos ,
6571     name-pl = capítulos ,
6572
6573 type = section ,
6574     gender = f ,
6575     Name-sg = Seção ,
6576     name-sg = seção ,
6577     Name-pl = Seções ,
6578     name-pl = seções ,
6579
6580 type = paragraph ,
6581     gender = m ,
6582     Name-sg = Parágrafo ,
6583     name-sg = parágrafo ,
6584     Name-pl = Parágrafos ,
6585     name-pl = parágrafos ,
6586     Name-sg-ab = Par. ,
6587     name-sg-ab = par. ,
6588     Name-pl-ab = Par. ,
6589     name-pl-ab = par. ,
6590
6591 type = appendix ,
6592     gender = m ,
6593     Name-sg = Apêndice ,
6594     name-sg = apêndice ,
6595     Name-pl = Apêndices ,
6596     name-pl = apêndices ,
6597
6598 type = page ,
6599     gender = f ,

```

```

6600 Name-sg = Página ,
6601 name-sg = página ,
6602 Name-pl = Páginas ,
6603 name-pl = páginas ,
6604 rangesep = {\textendash} ,
6605 rangetopair = false ,
6606
6607 type = line ,
6608 gender = f ,
6609 Name-sg = Linha ,
6610 name-sg = linha ,
6611 Name-pl = Linhas ,
6612 name-pl = linhas ,
6613
6614 type = figure ,
6615 gender = f ,
6616 Name-sg = Figura ,
6617 name-sg = figura ,
6618 Name-pl = Figuras ,
6619 name-pl = figuras ,
6620 Name-sg-ab = Fig. ,
6621 name-sg-ab = fig. ,
6622 Name-pl-ab = Figs. ,
6623 name-pl-ab = figs. ,
6624
6625 type = table ,
6626 gender = f ,
6627 Name-sg = Tabela ,
6628 name-sg = tabela ,
6629 Name-pl = Tabelas ,
6630 name-pl = tabelas ,
6631
6632 type = item ,
6633 gender = m ,
6634 Name-sg = Item ,
6635 name-sg = item ,
6636 Name-pl = Itens ,
6637 name-pl = itens ,
6638
6639 type = footnote ,
6640 gender = f ,
6641 Name-sg = Nota ,
6642 name-sg = nota ,
6643 Name-pl = Notas ,
6644 name-pl = notas ,
6645
6646 type = endnote ,
6647 gender = f ,
6648 Name-sg = Nota ,
6649 name-sg = nota ,
6650 Name-pl = Notas ,
6651 name-pl = notas ,
6652
6653 type = note ,

```

```

6654 gender = f ,
6655 Name-sg = Nota ,
6656 name-sg = nota ,
6657 Name-pl = Notas ,
6658 name-pl = notas ,
6659
6660 type = equation ,
6661 gender = f ,
6662 Name-sg = Equação ,
6663 name-sg = equação ,
6664 Name-pl = Equações ,
6665 name-pl = equações ,
6666 Name-sg-ab = Eq. ,
6667 name-sg-ab = eq. ,
6668 Name-pl-ab = Eqs. ,
6669 name-pl-ab = eqs. ,
6670 refbounds-first-sg = {(,),(,)} ,
6671 refbounds = {(,),(,)} ,
6672
6673 type = theorem ,
6674 gender = m ,
6675 Name-sg = Teorema ,
6676 name-sg = teorema ,
6677 Name-pl = Teoremas ,
6678 name-pl = teoremas ,
6679
6680 type = lemma ,
6681 gender = m ,
6682 Name-sg = Lema ,
6683 name-sg = lema ,
6684 Name-pl = Lemas ,
6685 name-pl = lemas ,
6686
6687 type = corollary ,
6688 gender = m ,
6689 Name-sg = Corolário ,
6690 name-sg = corolário ,
6691 Name-pl = Corolários ,
6692 name-pl = corolários ,
6693
6694 type = proposition ,
6695 gender = f ,
6696 Name-sg = Proposição ,
6697 name-sg = proposição ,
6698 Name-pl = Proposições ,
6699 name-pl = proposições ,
6700
6701 type = definition ,
6702 gender = f ,
6703 Name-sg = Definição ,
6704 name-sg = definição ,
6705 Name-pl = Definições ,
6706 name-pl = definições ,
6707

```

```

6708 type = proof ,
6709     gender = f ,
6710     Name-sg = Demonstração ,
6711     name-sg = demonstração ,
6712     Name-pl = Demonstrações ,
6713     name-pl = demonstrações ,
6714
6715 type = result ,
6716     gender = m ,
6717     Name-sg = Resultado ,
6718     name-sg = resultado ,
6719     Name-pl = Resultados ,
6720     name-pl = resultados ,
6721
6722 type = remark ,
6723     gender = f ,
6724     Name-sg = Observação ,
6725     name-sg = observação ,
6726     Name-pl = Observações ,
6727     name-pl = observações ,
6728
6729 type = example ,
6730     gender = m ,
6731     Name-sg = Exemplo ,
6732     name-sg = exemplo ,
6733     Name-pl = Exemplos ,
6734     name-pl = exemplos ,
6735
6736 type = algorithm ,
6737     gender = m ,
6738     Name-sg = Algoritmo ,
6739     name-sg = algoritmo ,
6740     Name-pl = Algoritmos ,
6741     name-pl = algoritmos ,
6742
6743 type = listing ,
6744     gender = f ,
6745     Name-sg = Listagem ,
6746     name-sg = listagem ,
6747     Name-pl = Listagens ,
6748     name-pl = listagens ,
6749
6750 type = exercise ,
6751     gender = m ,
6752     Name-sg = Exercício ,
6753     name-sg = exercício ,
6754     Name-pl = Exercícios ,
6755     name-pl = exercícios ,
6756
6757 type = solution ,
6758     gender = f ,
6759     Name-sg = Solução ,
6760     name-sg = solução ,
6761     Name-pl = Soluções ,

```

```

6762   name-pl = soluções ,
6763 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```

6764 <*package>
6765 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6766 </package>

6767 <*lang-spanish>

6768 namesep = {\nobreakspace} ,
6769 pairsep = {\~y\nobreakspace} ,
6770 listsep = {,~} ,
6771 lastsep = {\~y\nobreakspace} ,
6772 tpairsep = {\~y\nobreakspace} ,
6773 tlistsep = {,~} ,
6774 tlastsep = {\~y\nobreakspace} ,
6775 notesep = {\~} ,
6776 rangesep = {\~a\nobreakspace} ,
6777
6778 type = book ,
6779   gender = m ,
6780   Name-sg = Libro ,
6781   name-sg = libro ,
6782   Name-pl = Libros ,
6783   name-pl = libros ,
6784
6785 type = part ,
6786   gender = f ,
6787   Name-sg = Parte ,
6788   name-sg = parte ,
6789   Name-pl = Partes ,
6790   name-pl = partes ,
6791
6792 type = chapter ,
6793   gender = m ,
6794   Name-sg = Capítulo ,
6795   name-sg = capítulo ,
6796   Name-pl = Capítulos ,
6797   name-pl = capítulos ,
6798
6799 type = section ,
6800   gender = f ,
6801   Name-sg = Sección ,
6802   name-sg = sección ,
6803   Name-pl = Secciones ,
6804   name-pl = secciones ,
6805
6806 type = paragraph ,
6807   gender = m ,
6808   Name-sg = Párrafo ,
6809   name-sg = párrafo ,

```



```

6810     Name-pl = Párrafos ,
6811     name-pl = párrafos ,
6812
6813     type = appendix ,
6814     gender = m ,
6815     Name-sg = Apéndice ,
6816     name-sg = apéndice ,
6817     Name-pl = Apéndices ,
6818     name-pl = apéndices ,
6819
6820     type = page ,
6821     gender = f ,
6822     Name-sg = Página ,
6823     name-sg = página ,
6824     Name-pl = Páginas ,
6825     name-pl = páginas ,
6826     rangsep = {\textendash} ,
6827     rangetopair = false ,
6828
6829     type = line ,
6830     gender = f ,
6831     Name-sg = Línea ,
6832     name-sg = línea ,
6833     Name-pl = Líneas ,
6834     name-pl = líneas ,
6835
6836     type = figure ,
6837     gender = f ,
6838     Name-sg = Figura ,
6839     name-sg = figura ,
6840     Name-pl = Figuras ,
6841     name-pl = figuras ,
6842
6843     type = table ,
6844     gender = m ,
6845     Name-sg = Cuadro ,
6846     name-sg = cuadro ,
6847     Name-pl = Cuadros ,
6848     name-pl = cuadros ,
6849
6850     type = item ,
6851     gender = m ,
6852     Name-sg = Punto ,
6853     name-sg = punto ,
6854     Name-pl = Puntos ,
6855     name-pl = puntos ,
6856
6857     type = footnote ,
6858     gender = f ,
6859     Name-sg = Nota ,
6860     name-sg = nota ,
6861     Name-pl = Notas ,
6862     name-pl = notas ,
6863

```

```

6864 type = endnote ,
6865     gender = f ,
6866     Name-sg = Nota ,
6867     name-sg = nota ,
6868     Name-pl = Notas ,
6869     name-pl = notas ,
6870
6871 type = note ,
6872     gender = f ,
6873     Name-sg = Nota ,
6874     name-sg = nota ,
6875     Name-pl = Notas ,
6876     name-pl = notas ,
6877
6878 type = equation ,
6879     gender = f ,
6880     Name-sg = Ecuación ,
6881     name-sg = ecuación ,
6882     Name-pl = Ecuaciones ,
6883     name-pl = ecuaciones ,
6884     refbounds-first-sg = {(,),(,)},
6885     refbounds = {(,,)},
6886
6887 type = theorem ,
6888     gender = m ,
6889     Name-sg = Teorema ,
6890     name-sg = teorema ,
6891     Name-pl = Teoremas ,
6892     name-pl = teoremas ,
6893
6894 type = lemma ,
6895     gender = m ,
6896     Name-sg = Lema ,
6897     name-sg = lema ,
6898     Name-pl = Lemas ,
6899     name-pl = lemas ,
6900
6901 type = corollary ,
6902     gender = m ,
6903     Name-sg = Corolario ,
6904     name-sg = corolario ,
6905     Name-pl = Corolarios ,
6906     name-pl = corolarios ,
6907
6908 type = proposition ,
6909     gender = f ,
6910     Name-sg = Proposición ,
6911     name-sg = proposición ,
6912     Name-pl = Proposiciones ,
6913     name-pl = proposiciones ,
6914
6915 type = definition ,
6916     gender = f ,
6917     Name-sg = Definición ,

```

```

6918     name-sg = definición ,
6919     Name-pl = Definiciones ,
6920     name-pl = definiciones ,
6921
6922 type = proof ,
6923     gender = f ,
6924     Name-sg = Demostración ,
6925     name-sg = demostración ,
6926     Name-pl = Demostraciones ,
6927     name-pl = demostraciones ,
6928
6929 type = result ,
6930     gender = m ,
6931     Name-sg = Resultado ,
6932     name-sg = resultado ,
6933     Name-pl = Resultados ,
6934     name-pl = resultados ,
6935
6936 type = remark ,
6937     gender = f ,
6938     Name-sg = Observación ,
6939     name-sg = observación ,
6940     Name-pl = Observaciones ,
6941     name-pl = observaciones ,
6942
6943 type = example ,
6944     gender = m ,
6945     Name-sg = Ejemplo ,
6946     name-sg = ejemplo ,
6947     Name-pl = Ejemplos ,
6948     name-pl = ejemplos ,
6949
6950 type = algorithm ,
6951     gender = m ,
6952     Name-sg = Algoritmo ,
6953     name-sg = algoritmo ,
6954     Name-pl = Algoritmos ,
6955     name-pl = algoritmos ,
6956
6957 type = listing ,
6958     gender = m ,
6959     Name-sg = Listado ,
6960     name-sg = listado ,
6961     Name-pl = Listados ,
6962     name-pl = listados ,
6963
6964 type = exercise ,
6965     gender = m ,
6966     Name-sg = Ejercicio ,
6967     name-sg = ejercicio ,
6968     Name-pl = Ejercicios ,
6969     name-pl = ejercicios ,
6970
6971 type = solution ,

```

```

6972 gender = f ,
6973 Name-sg = Solución ,
6974 name-sg = solución ,
6975 Name-pl = Soluciones ,
6976 name-pl = soluciones ,
6977 </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.

```

6978 <*package>
6979 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6980 </package>
6981 <*lang-dutch>
6982 namesep = {\nobreakspace} ,
6983 pairsep = {\sim\nobreakspace} ,
6984 listsep = {,~} ,
6985 lastsep = {\sim\nobreakspace} ,
6986 tpairsep = {\sim\nobreakspace} ,
6987 tlistsep = {,~} ,
6988 tlastsep = {,~\nobreakspace} ,
6989 notesep = {\sim} ,
6990 rangesep = {\sim/m\nobreakspace} ,
6991
6992 type = book ,
6993 gender = n ,
6994 Name-sg = Boek ,
6995 name-sg = boek ,
6996 Name-pl = Boeken ,
6997 name-pl = boeken ,
6998
6999 type = part ,
7000 gender = n ,
7001 Name-sg = Deel ,
7002 name-sg = deel ,
7003 Name-pl = Delen ,
7004 name-pl = delen ,
7005
7006 type = chapter ,
7007 gender = n ,
7008 Name-sg = Hoofdstuk ,
7009 name-sg = hoofdstuk ,
7010 Name-pl = Hoofdstukken ,
7011 name-pl = hoofdstukken ,
7012
7013 type = section ,
7014 gender = m ,
7015 Name-sg = Paragraaf ,
7016 name-sg = paragraaf ,
7017 Name-pl = Paragrafen ,
7018 name-pl = paragrafen ,

```

```

7019
7020 type = paragraph ,
7021     gender = f ,
7022     Name-sg = Alinea ,
7023     name-sg = alinea ,
7024     Name-pl = Alinea's ,
7025     name-pl = alinea's ,
7026

```

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.

```

7027 type = appendix ,
7028     gender = { f , m } ,
7029     Name-sg = Blage ,
7030     name-sg = blage ,
7031     Name-pl = Blagen ,
7032     name-pl = blagen ,
7033
7034 type = page ,
7035     gender = { f , m } ,
7036     Name-sg = Pagina ,
7037     name-sg = pagina ,
7038     Name-pl = Pagina's ,
7039     name-pl = pagina's ,
7040     rangesep = {\textendash} ,
7041     rangetopair = false ,
7042
7043 type = line ,
7044     gender = m ,
7045     Name-sg = Regel ,
7046     name-sg = regel ,
7047     Name-pl = Regels ,
7048     name-pl = regels ,
7049
7050 type = figure ,
7051     gender = { n , f , m } ,
7052     Name-sg = Figuur ,
7053     name-sg = figuur ,
7054     Name-pl = Figuren ,
7055     name-pl = figuren ,
7056
7057 type = table ,
7058     gender = { f , m } ,
7059     Name-sg = Tabel ,
7060     name-sg = tabel ,
7061     Name-pl = Tabellen ,
7062     name-pl = tabellen ,
7063
7064 type = item ,
7065     gender = n ,
7066     Name-sg = Punt ,
7067     name-sg = punt ,
7068     Name-pl = Punten ,

```

```

7069     name-pl = punten ,
7070
7071 type = footnote ,
7072     gender = { f , m } ,
7073     Name-sg = Voetnoot ,
7074     name-sg = voetnoot ,
7075     Name-pl = Voetnoten ,
7076     name-pl = voetnoten ,
7077
7078 type = endnote ,
7079     gender = { f , m } ,
7080     Name-sg = Eindnoot ,
7081     name-sg = eindnoot ,
7082     Name-pl = Eindnoten ,
7083     name-pl = eindnoten ,
7084
7085 type = note ,
7086     gender = f ,
7087     Name-sg = Opmerking ,
7088     name-sg = opmerking ,
7089     Name-pl = Opmerkingen ,
7090     name-pl = opmerkingen ,
7091
7092 type = equation ,
7093     gender = f ,
7094     Name-sg = Vergelking ,
7095     name-sg = vergelking ,
7096     Name-pl = Vergelkingen ,
7097     name-pl = vergelkingen ,
7098     Name-sg-ab = Vgl. ,
7099     name-sg-ab = vgl. ,
7100     Name-pl-ab = Vgl.'s ,
7101     name-pl-ab = vgl.'s ,
7102     refbounds-first-sg = {,(,),} ,
7103     refbounds = {(,,)} ,
7104
7105 type = theorem ,
7106     gender = f ,
7107     Name-sg = Stelling ,
7108     name-sg = stelling ,
7109     Name-pl = Stellingen ,
7110     name-pl = stellingen ,
7111

```

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”.

```

7112 type = lemma ,
7113     gender = n ,
7114     Name-sg = Lemma ,
7115     name-sg = lemma ,
7116     Name-pl = Lemma’s ,
7117     name-pl = lemma’s ,
7118

```

```

7119 type = corollary ,
7120     gender = n ,
7121     Name-sg = Gevolg ,
7122     name-sg = gevolg ,
7123     Name-pl = Gevolgen ,
7124     name-pl = gevolgen ,
7125
7126 type = proposition ,
7127     gender = f ,
7128     Name-sg = Propositie ,
7129     name-sg = propositie ,
7130     Name-pl = Propositiones ,
7131     name-pl = proposities ,
7132
7133 type = definition ,
7134     gender = f ,
7135     Name-sg = Definitie ,
7136     name-sg = definitie ,
7137     Name-pl = Definities ,
7138     name-pl = definities ,
7139
7140 type = proof ,
7141     gender = n ,
7142     Name-sg = Bews ,
7143     name-sg = bews ,
7144     Name-pl = Bewzen ,
7145     name-pl = bewzen ,
7146
7147 type = result ,
7148     gender = n ,
7149     Name-sg = Resultaat ,
7150     name-sg = resultaat ,
7151     Name-pl = Resultaten ,
7152     name-pl = resultaten ,
7153
7154 type = remark ,
7155     gender = f ,
7156     Name-sg = Opmerking ,
7157     name-sg = opmerking ,
7158     Name-pl = Opmerkingen ,
7159     name-pl = opmerkingen ,
7160
7161 type = example ,
7162     gender = n ,
7163     Name-sg = Voorbeeld ,
7164     name-sg = voorbeeld ,
7165     Name-pl = Voorbeelden ,
7166     name-pl = voorbeelden ,
7167

```

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”.

```

7168 type = algorithm ,
7169     gender = { n , f , m } ,

```

```

7170 Name-sg = Algoritme ,
7171 name-sg = algoritme ,
7172 Name-pl = Algoritmen ,
7173 name-pl = algoritmen ,
7174

```

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

```

7175 type = listing ,
7176 gender = m ,
7177 Name-sg = Listing ,
7178 name-sg = listing ,
7179 Name-pl = Listings ,
7180 name-pl = listings ,
7181
7182 type = exercise ,
7183 gender = { f , m } ,
7184 Name-sg = Opgave ,
7185 name-sg = opgave ,
7186 Name-pl = Opgaven ,
7187 name-pl = opgaven ,
7188
7189 type = solution ,
7190 gender = f ,
7191 Name-sg = Oplossing ,
7192 name-sg = oplossing ,
7193 Name-pl = Oplossingen ,
7194 name-pl = oplossingen ,
7195 </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->

```

7196 <*package>
7197 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7198 </package>
7199 <*lang-italian>
7200 namesep = {\nobreakspace} ,
7201 pairsep = {\~e\nobreakspace} ,
7202 listsep = {,~} ,
7203 lastsep = {\~e\nobreakspace} ,
7204 tpairsep = {\~e\nobreakspace} ,
7205 tlistsep = {,~} ,
7206 tlastsep = {,~e\nobreakspace} ,
7207 notesep = {\~} ,
7208 rangesep = {\~a\nobreakspace} ,
7209 +refbounds-rb = {da\nobreakspace,,,} ,
7210
7211 type = book ,
7212 gender = m ,

```



```

7213   Name-sg = Libro ,
7214   name-sg = libro ,
7215   Name-pl = Libri ,
7216   name-pl = libri ,
7217
7218   type = part ,
7219   gender = f ,
7220   Name-sg = Parte ,
7221   name-sg = parte ,
7222   Name-pl = Parti ,
7223   name-pl = parti ,
7224
7225   type = chapter ,
7226   gender = m ,
7227   Name-sg = Capitolo ,
7228   name-sg = capitolo ,
7229   Name-pl = Capitoli ,
7230   name-pl = capitoli ,
7231
7232   type = section ,
7233   gender = m ,
7234   Name-sg = Paragrafo ,
7235   name-sg = paragrafo ,
7236   Name-pl = Paragrafi ,
7237   name-pl = paragrafi ,
7238
7239   type = paragraph ,
7240   gender = m ,
7241   Name-sg = Capoverso ,
7242   name-sg = capoverso ,
7243   Name-pl = Capoversi ,
7244   name-pl = capoversi ,
7245
7246   type = appendix ,
7247   gender = f ,
7248   Name-sg = Appendice ,
7249   name-sg = appendice ,
7250   Name-pl = Appendici ,
7251   name-pl = appendici ,
7252
7253   type = page ,
7254   gender = f ,
7255   Name-sg = Pagina ,
7256   name-sg = pagina ,
7257   Name-pl = Pagine ,
7258   name-pl = pagine ,
7259   Name-sg-ab = Pag. ,
7260   name-sg-ab = pag. ,
7261   Name-pl-ab = Pag. ,
7262   name-pl-ab = pag. ,
7263   rangesep = {\textendash} ,
7264   rangetopair = false ,
7265   +refbounds-rb = {,,} ,
7266

```

```

7267 type = line ,
7268     gender = f ,
7269     Name-sg = Riga ,
7270     name-sg = riga ,
7271     Name-pl = Righe ,
7272     name-pl = righe ,
7273
7274 type = figure ,
7275     gender = f ,
7276     Name-sg = Figura ,
7277     name-sg = figura ,
7278     Name-pl = Figure ,
7279     name-pl = figure ,
7280     Name-sg-ab = Fig. ,
7281     name-sg-ab = fig. ,
7282     Name-pl-ab = Fig. ,
7283     name-pl-ab = fig. ,
7284
7285 type = table ,
7286     gender = f ,
7287     Name-sg = Tabella ,
7288     name-sg = tabella ,
7289     Name-pl = Tabelle ,
7290     name-pl = tabelle ,
7291     Name-sg-ab = Tab. ,
7292     name-sg-ab = tab. ,
7293     Name-pl-ab = Tab. ,
7294     name-pl-ab = tab. ,
7295
7296 type = item ,
7297     gender = m ,
7298     Name-sg = Punto ,
7299     name-sg = punto ,
7300     Name-pl = Punti ,
7301     name-pl = punti ,
7302
7303 type = footnote ,
7304     gender = f ,
7305     Name-sg = Nota ,
7306     name-sg = nota ,
7307     Name-pl = Note ,
7308     name-pl = note ,
7309
7310 type = endnote ,
7311     gender = f ,
7312     Name-sg = Nota ,
7313     name-sg = nota ,
7314     Name-pl = Note ,
7315     name-pl = note ,
7316
7317 type = note ,
7318     gender = f ,
7319     Name-sg = Nota ,
7320     name-sg = nota ,

```

```

7321 Name-pl = Note ,
7322 name-pl = note ,
7323
7324 type = equation ,
7325 gender = f ,
7326 Name-sg = Equazione ,
7327 name-sg = equazione ,
7328 Name-pl = Equazioni ,
7329 name-pl = equazioni ,
7330 Name-sg-ab = Eq. ,
7331 name-sg-ab = eq. ,
7332 Name-pl-ab = Eq. ,
7333 name-pl-ab = eq. ,
7334 +refbounds-rb = {da\nobreakspace(,,)} ,
7335 refbounds-first-sg = {(,)}, ,
7336 refbounds = {(,,)} ,
7337
7338 type = theorem ,
7339 gender = m ,
7340 Name-sg = Teorema ,
7341 name-sg = teorema ,
7342 Name-pl = Teoremi ,
7343 name-pl = teoremi ,
7344
7345 type = lemma ,
7346 gender = m ,
7347 Name-sg = Lemma ,
7348 name-sg = lemma ,
7349 Name-pl = Lemmi ,
7350 name-pl = lemmi ,
7351
7352 type = corollary ,
7353 gender = m ,
7354 Name-sg = Corollario ,
7355 name-sg = corollario ,
7356 Name-pl = Corollari ,
7357 name-pl = corollari ,
7358
7359 type = proposition ,
7360 gender = f ,
7361 Name-sg = Proposizione ,
7362 name-sg = proposizione ,
7363 Name-pl = Proposizioni ,
7364 name-pl = proposizioni ,
7365
7366 type = definition ,
7367 gender = f ,
7368 Name-sg = Definizione ,
7369 name-sg = definizione ,
7370 Name-pl = Definizioni ,
7371 name-pl = definizioni ,
7372
7373 type = proof ,
7374 gender = f ,

```

```

7375   Name-sg = Dimostrazione ,
7376   name-sg = dimostrazione ,
7377   Name-pl = Dimostrazioni ,
7378   name-pl = dimostrazioni ,
7379
7380   type = result ,
7381   gender = m ,
7382   Name-sg = Risultato ,
7383   name-sg = risultato ,
7384   Name-pl = Risultati ,
7385   name-pl = risultati ,
7386
7387   type = remark ,
7388   gender = f ,
7389   Name-sg = Osservazione ,
7390   name-sg = osservazione ,
7391   Name-pl = Osservazioni ,
7392   name-pl = osservazioni ,
7393
7394   type = example ,
7395   gender = m ,
7396   Name-sg = Esempio ,
7397   name-sg = esempio ,
7398   Name-pl = Esempi ,
7399   name-pl = esempi ,
7400
7401   type = algorithm ,
7402   gender = m ,
7403   Name-sg = Algoritmo ,
7404   name-sg = algoritmo ,
7405   Name-pl = Algoritmi ,
7406   name-pl = algoritmi ,
7407
7408   type = listing ,
7409   gender = m ,
7410   Name-sg = Listato ,
7411   name-sg = listato ,
7412   Name-pl = Listati ,
7413   name-pl = listati ,
7414
7415   type = exercise ,
7416   gender = m ,
7417   Name-sg = Esercizio ,
7418   name-sg = esercizio ,
7419   Name-pl = Esercizi ,
7420   name-pl = esercizi ,
7421
7422   type = solution ,
7423   gender = f ,
7424   Name-sg = Soluzione ,
7425   name-sg = soluzione ,
7426   Name-pl = Soluzioni ,
7427   name-pl = soluzioni ,
7428 </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	1884
\AddToHook	108, 2030, 2074, 2097, 2128, 2130, 2168, 2241, 2283, 2446, 2459, 2467, 5326, 5362, 5379, 5381, 5386, 5423, 5427, 5429, 5434, 5444, 5454, 5488, 5539, 5564
\AddToHookWithArguments	2421
\alph	5469, 5555
\appendix	2, 126, 127, 129, 136
\appendixname	126, 138
\AtEndOfPackage	2457
B	
\babelname	2084
\babelprovide	30, 55
bool commands:	
\bool_gset_false:N	540
\bool_gset_true:N	533, 2433
\bool_if:NTF ...	384, 461, 499, 557, 1760, 1811, 2034, 2038, 2423, 2469, 3443, 3846, 3987, 4114, 4150, 4184, 4251, 4264, 4276, 4327, 4344, 4354, 4359, 4405, 4409, 4465, 4490, 4497, 4503, 4514, 4520, 4548, 4593, 4615, 4644, 4793, 4946, 4948, 5491, 5567
\bool_if:nTF	77, 3557, 3567, 3591, 3608, 3623, 3688, 3696, 4337, 4714, 4755, 4836, 4853
\bool_if_exist:NTF	338, 348, 372, 382, 432, 449, 459, 484, 487, 495, 497, 511, 514, 521, 524, 531, 538
\bool_lazy_all:nTF	4433, 4958
\bool_lazy_and:nnTF	50, 55, 2590, 3414, 3435, 4218, 4289, 4667, 4931, 4973
\bool_lazy_any:nTF	5110, 5119
\bool_lazy_or:nnTF	1377, 1410, 1960, 2539, 2803, 3302, 3327, 3418, 4919
\bool_new:N	339, 349, 374, 433, 451, 489, 512, 515, 522, 525, 532, 539, 799, 1574, 1575, 1602, 1626, 1978, 1985, 1992, 2005, 2006, 2176, 2177, 2178, 2179, 2180, 2276, 2277, 2414, 2426, 3451, 3466, 3728, 3729, 3740, 3741, 3748, 3750, 3751, 3764, 3765, 3766, 3778, 3779, 5453, 5498, 5538
\bool_set:Nn	3411
\bool_set_eq:NN	547
\bool_set_false:N ...	373, 450, 486, 488, 523, 1587, 1591, 1763, 1861, 1906, 1948, 2013, 2022, 2023, 2040, 2047, 2188, 2192, 2199, 2207, 2208, 2209, 2301, 2313, 3456, 3549, 3795, 3796, 3813, 3852, 3863, 4263, 4455, 4456, 4463, 4464, 4697, 5117, 5134
\bool_set_true:N	340, 350, 434, 513, 516, 526, 1581, 1582, 1586, 1592, 1785, 1807, 1870, 1872, 1910, 1912, 1921, 1923, 1952, 1954, 1967, 1969, 2012, 2017, 2018, 2186, 2193, 2198, 2215, 2217, 2219, 2222, 2223, 2224, 2289, 2294, 3563, 3573, 3577, 3599, 3614, 3629, 3653, 3821, 3847, 3853, 3857, 3864, 4010, 4021, 4032, 4082, 4118, 4154, 4188, 4205, 4286, 4320, 4492, 4552, 4598, 4620, 4649, 4907, 4914, 5036, 5095, 5133, 5170, 5177, 5178, 5196, 5213, 5215, 5467, 5505, 5552
\bool_until_do:Nn	1862, 1913, 1955, 3589, 3814
\bool_while_do:nn	5597
\l_tmpa_bool	1763, 1785, 1807, 1811, 1861, 1862, 1870, 1872, 1906, 1910, 1912, 1913, 1921, 1923, 1948, 1952, 1954, 1955, 1967, 1969
C	
\chaptername	138
clist commands:	
\clist_map_inline:nn	632, 680, 697, 1016, 2210, 2362, 2435, 2939, 5470, 5557
\contsubbottom	129
\contsubcaption	129
\counterwithin	5
\crefstripprefix	45
cs commands:	
\cs_generate_variant:Nn	74, 281, 287, 294, 305, 316, 327, 342, 352, 359, 366, 377, 401, 411, 436, 443, 454, 492, 518, 528, 535, 542, 970, 1844, 1880, 1931, 1977, 2598, 4749, 4787, 5153, 5257, 5290, 5323
\cs_if_exist:NTF	25, 28, 67, 87, 5350, 5356

\cs_if_exist_p:N		G	
..... 52, 57, 4220, 4291, 4669, 5599		group commands:	
\cs_if_exist_use:N	5422, 5637	\group_begin:	
\cs_new:Npn	65, 75, 85, 96, 110, 735, 916, 1762, 1849,	
282, 288, 290, 292, 295, 306, 317,		1894, 1936, 2863, 3408, 3422, 3454,	
329, 331, 719, 4710, 4750, 4788, 5143		4361, 4377, 4726, 4738, 4767, 4778,	
\cs_new_protected:Npn	276, 333, 343,	4809, 4816, 4827, 4842, 4873, 4888	
353, 360, 367, 392, 402, 423, 425,		\group_end: 113, 747, 968, 1826, 1875,	
427, 437, 444, 482, 509, 519, 529,		1926, 1972, 2891, 3425, 3448, 3458,	
536, 812, 914, 1523, 1542, 1746,		4374, 4380, 4730, 4742, 4770, 4781,	
1845, 1890, 1932, 2465, 2596, 3406,		4812, 4821, 4830, 4845, 4878, 4893	
3470, 3512, 3523, 3536, 3666, 3718,			
3780, 3994, 4459, 4706, 4708, 4902,			
5137, 5154, 5225, 5258, 5291, 5506			
\cs_set_eq:NN	721	H	
\cs_to_str:N	332	\hyperlink	5140
		I	
D		\IfBooleanT	3455
\d	1884	\IfClassLoadedTF	120
		\ifdraft	2191
E		\IfFormatAtLeastTF	4, 5
\endinput	13	\ifoptionfinal	2197
exp commands:		\IfPackageAtLeastTF	2287
\exp_args:NNe	37, 40	\IfPackageLoadedTF	118
\exp_args:NNNo	278	\input	30
\exp_args:NNNV	1825, 1874, 1925, 1971	int commands:	
\exp_args:NNo	278, 284	\int_case:nnTF	
\exp_args:No	284	... 3997, 4039, 4101, 4412, 4527, 4584	
\exp_args:Nx	924, 5511, 5519	\int_compare:nNnTF	
\exp_args:Nxx		3600, 3615, 3630, 3642, 3654, 3674,	
1764, 1776, 1788, 1798, 1864, 1915,		3676, 3720, 3894, 4017, 4045, 4067,	
1957, 5156, 5162, 5184, 5188, 5203		4122, 4192, 4397, 4399, 4476, 4506,	
\exp_not:N	113,	4565, 5166, 5172, 5192, 5198, 5615	
4361, 4374, 4377, 4380, 4726, 4730,		\int_compare_p:nNn 1380, 1413, 1961,	
4738, 4742, 4767, 4770, 4778, 4781,		1962, 2541, 2805, 3305, 3330, 3690,	
4809, 4812, 4816, 4821, 4827, 4830,		3698, 4437, 4923, 4934, 4963, 5130	
4842, 4845, 4873, 4878, 4888, 4893		\int_incr:N	
\exp_not:n	285, 4026, 4049,	... 4452, 4484, 4496, 4498, 4513,	
4057, 4075, 4088, 4092, 4127, 4136,		4515, 4519, 4521, 4533, 4556, 4568,	
4158, 4166, 4173, 4197, 4211, 4228,		4602, 4624, 4633, 4653, 4704, 5613	
4238, 4280, 4303, 4313, 4362, 4373,		\int_new:N	3467,
4378, 4379, 4536, 4559, 4571, 4605,		3468, 3730, 3731, 3732, 3745, 3746	
4627, 4636, 4656, 4677, 4687, 4727,		\int_rand:n	303, 314, 325
4739, 4768, 4769, 4779, 4780, 4810,		\int_set:Nn	
4811, 4813, 4817, 4828, 4829, 4831,		... 111, 3675, 3677, 3681, 3684, 5596	
4839, 4843, 4844, 4847, 4874, 4889		\int_to_roman:n 5600, 5607, 5608, 5611	
\ExplSyntaxOn	30, 930	\int_use:N	53, 58, 63, 69
		\int_zero:N	3668,
F		3669, 3790, 3791, 3792, 3793, 3794,	
\figurename	138	4450, 4451, 4453, 4454, 4699, 4700	
file commands:		\l_tmpa_int	5596,
\file_get:nnNTF	924	5600, 5607, 5608, 5611, 5613, 5615	
\fmtversion	4	low commands:	
\footnote	2, 129, 130	\iow_char:N	
		... 124, 139, 140, 145, 146, 151,	
		152, 157, 158, 210, 227, 274, 2258, 2267	

\iow_newline:	268	\msg_warning:nnn	739, 760, 1555, 1562, 1718, 1724, 2117, 2154, 2166, 2228, 2239, 2281, 2306, 2390, 2452, 2462, 2612, 2726, 2732, 2890, 2934, 2980, 3172, 3178, 3259, 3878, 4258, 4952, 4968
J			
\jobname	5444	\msg_warning:nnnn	828, 845, 879, 897, 2070, 2257, 2266, 2335, 2500, 2506, 2512, 2518, 2548, 2761, 2767, 2773, 2779, 2815, 2907, 2914, 2944, 3227, 3233, 3239, 3245, 3318, 3343, 3886, 5037, 5097
K			
keys commands:		\msg_warning:nnnnn 865, 904, 2928, 4987	
\l_keys_choice_tl	804	\msg_warning:nnnnnn	4994
\keys_define:nn	20, 640, 686, 703, 764, 971, 1060, 1085, 1113, 1322, 1361, 1439, 1549, 1576, 1603, 1612, 1627, 1636, 1979, 1986, 1993, 1999, 2007, 2042, 2051, 2065, 2093, 2132, 2163, 2170, 2182, 2243, 2250, 2252, 2261, 2271, 2278, 2290, 2302, 2314, 2322, 2329, 2358, 2384, 2408, 2415, 2428, 2448, 2477, 2495, 2525, 2559, 2582, 2608, 2620, 2644, 2756, 2786, 2826, 2894, 2965, 2989, 3016, 3222, 3252, 3292, 3354		
\keys_set:nn 27, 30, 57, 58, 84, 744, 892, 955, 2295, 2597, 2602, 2888, 3409			
keyval commands:			
\keyval_parse:nnn . . .	1528, 2333, 2388		
L			
\label	2, 60, 63, 133		
\labelformat	3		
\language name	24, 138, 2078		
M			
\mainbabelname	24, 2085		
\MessageBreak	11		
MH commands:			
\MH_if_boolean:nTF	5503		
msg commands:			
\msg_info:nnn	958, 1011, 1076, 1269, 1275, 1329, 5400, 5446, 5495, 5530, 5571, 5589, 5616		
\msg_info:nnnn	984, 991, 1021, 1393, 1427		
\msg_info:nnnnn	1005		
\msg_line_context:	123, 129, 133, 135, 138, 144, 150, 156, 162, 167, 172, 177, 182, 188, 193, 196, 199, 204, 208, 215, 220, 225, 232, 241, 246, 251, 255, 257, 259, 261, 268		
\msg_new:nnn	121, 127, 132, 134, 136, 142, 148, 154, 160, 165, 170, 175, 180, 185, 190, 195, 197, 202, 207, 209, 211, 213, 218, 223, 229, 231, 233, 238, 244, 249, 254, 256, 258, 260, 262, 264, 266, 271		
\msg_warning:nn	2039, 2045, 2317, 2593, 4439		
N			
\NeedsTeXFormat	3		
\newcounter	5, 5346, 5347		
\NewDocumentCommand	733, 750, 2594, 2599, 2861, 3404, 3452		
\newfloat	129		
\NewHook	1635		
\newsfloat	129		
\newtheorem	137		
\nobreakspace	1536, 5673, 5674, 5676, 5677, 5679, 5681, 5878, 5879, 5881, 5882, 5884, 5886, 6325, 6326, 6328, 6329, 6331, 6333, 6542, 6543, 6545, 6546, 6548, 6550, 6768, 6769, 6771, 6772, 6774, 6776, 6982, 6983, 6985, 6986, 6988, 6990, 7200, 7201, 7203, 7204, 7206, 7208, 7209, 7334		
\NumCheckSetup	45		
\NumsCheckSetup	45		
P			
\PackageError	8		
\pagename	138		
\pagenote	129, 130		
\pagenumbering	7		
\pageref	85		
\PagesCheckSetup	45		
\paragraph	61		
\part	137		
\partname	138		
prg commands:			
\prg_generate_conditional_- variant:Nnn	421, 469, 480, 507, 552, 560, 729, 1888		
\prg_new_conditional:Npnn	117, 119, 378, 455, 493, 554, 723		
\prg_new_protected_conditional:Npnn	412, 471, 543, 1881		

`\prg_return_false:`
. 118, 120, 386, 390, 419, 463, 467,
478, 501, 505, 550, 557, 558, 727, 1886
`\prg_return_true:`
.... 118, 120, 385, 388, 417, 462,
465, 476, 500, 503, 548, 557, 726, 1885
`\prg_set_eq_conditional:NNn` ... 731
prop commands:
`\prop_if_in:NnTF` 37
`\prop_if_in_p:Nn` 78
`\prop_item:Nn` 40, 79
`\prop_new:N` 2328, 2383
`\prop_put:Nnn` 1546
`\prop_remove:Nn` 1545
`\providecommand` 4
`\ProvidesExplPackage` 15
`\ProvidesFile` 30

R

`\refstepcounter` . 3, 4, 130, 131, 133, 134
regex commands:
`\regex_match:nnTF` 1884
`\renewlist` 134
`\RequirePackage` 17, 18, 19, 20, 2035

S

`\scantokens` 127
seq commands:
`\seq_clear:N` 448, 823,
860, 941, 954, 1015, 1623, 2536,
2800, 2874, 2887, 2938, 3472, 5284
`\seq_const_from_clist:Nn` 21
`\seq_count:N`
1381, 1395, 1414, 1429, 2541, 2550,
2805, 2817, 3306, 3320, 3331, 3345
`\seq_gclear:N` . 1374, 1407, 3299, 3324
`\seq_gconcat:NNN` 625, 629
`\seq_get_left:NN`
. 838, 849, 945, 993, 2878, 2916, 3824
`\seq_gput_right:Nn` ... 956, 962, 2439
`\seq_gremove_all:Nn` 2471
`\seq_gset_eq:NN` 441, 1043
`\seq_gset_from_clist:Nn`
.... 568, 577, 589, 604, 612, 773, 788
`\seq_gset_split:Nnn` 426
`\seq_if_empty:NTF` .. 824, 861, 942,
982, 1003, 2875, 2905, 2926, 3818, 4985
`\seq_if_exist:NTF` .. 429, 439, 446, 457
`\seq_if_in:NnTF`
.. 842, 876, 920, 988, 1018, 2364,
2437, 2470, 2911, 2941, 3516, 4981
`\seq_item:Nn` 4720, 4725, 4731, 4733,
4736, 4737, 4743, 4744, 4761, 4766,
4771, 4773, 4776, 4777, 4782, 4783,
4814, 4815, 4822, 4824, 4859, 4871,
4879, 4882, 4886, 4887, 4894, 4895
`\seq_map_break:n` 99, 3709, 3712
`\seq_map_function:NN` 3475
`\seq_map_indexed_inline:Nn` . 44, 3670
`\seq_map_inline:Nn` ... 1057, 1082,
1319, 1358, 1436, 2461, 2474, 2522,
2556, 2605, 2617, 2783, 2823, 2962,
2986, 3249, 3289, 3351, 3706, 5509
`\seq_map_tokens:Nn` 81
`\seq_new:N` 430,
440, 565, 566, 567, 576, 588, 603,
611, 624, 628, 768, 783, 913, 1035,
1611, 2357, 2427, 3450, 3469, 3727,
3744, 3767, 3768, 3769, 3770, 3771,
3772, 3773, 3774, 3775, 3776, 3777
`\seq_pop_left:NN` 3816
`\seq_put_right:Nn`
..... 1019, 2366, 2942, 3519
`\seq_reverse:N` 1617
`\seq_set_eq:NN`
. 431, 475, 3782, 4008, 4019, 4030,
4080, 4116, 4152, 4186, 4202, 4284,
4318, 4329, 4550, 4595, 4617, 4646
`\seq_set_from_clist:Nn` ... 1616, 3410
`\seq_set_split:Nnn` 424
`\seq_sort:Nn` 87, 3478
`\seq_use:Nn` 4999
`\g_tmpa_seq` 1374, 1376,
1381, 1390, 1395, 1407, 1409, 1414,
1424, 1429, 3299, 3301, 3306, 3315,
3320, 3324, 3326, 3331, 3340, 3345
`\l_tmpa_seq` 1015, 1019, 1051, 2536,
2538, 2541, 2545, 2550, 2800, 2802,
2805, 2812, 2817, 2938, 2942, 2957
`\setcounter` .. 5348, 5349, 5365, 5380, 5384
`\sidefootnote` 129, 130
sort commands:
`\sort_return_same:` 87,
91, 3485, 3490, 3564, 3584, 3605,
3620, 3634, 3659, 3694, 3709, 3725
`\sort_return_swapped:`
.. 87, 91, 3498, 3574, 3583, 3604,
3619, 3635, 3658, 3702, 3712, 3724
`\stepcounter` 133, 5364, 5383
str commands:
`\str_case:nn` 45
`\str_case:nnTF` 1118,
1640, 2099, 2136, 2212, 2648, 3021
`\str_compare:nNnTF` 3580
`\str_if_eq:nnTF` 98, 4752
`\str_if_eq_p:nn` 5115, 5121, 5123, 5127
`\str_new:N` 2050
`\str_set:Nn` ... 2055, 2057, 2059, 2061

<code>\string</code>	5516, 5524	<code>\zref@localaddprop</code>	5424, 5492, 5568, 5639
<code>\subbottom</code>	129	<code>\ZREF@mainlist</code>	22, 32, 47,
<code>\subcaption</code>	129		62, 64, 106, 116, 5424, 5492, 5568, 5639
<code>\subcaptionref</code>	129	<code>\zref@newprop</code>	6, 7, 21, 23, 33,
<code>\subparagraph</code>	136		48, 63, 101, 115, 5421, 5469, 5555, 5636
<code>\subref</code>	135	<code>\zref@refused</code>	3871
<code>\subsections</code>	136	<code>\zref@wrapper@babel</code>	63, 83, 2424, 3405
<code>\subsubsection</code>	61	<code>\textendash</code>	1540,
<code>\subsubsections</code>	136		5728, 5992, 6604, 6826, 7040, 7263
<code>\subsubsubsection</code>	61	<code>\thechapter</code>	126
<code>\subtop</code>	129	<code>\thelstnumber</code>	134
		<code>\thepage</code>	7, 112
		<code>\thesection</code>	126
		tl commands:	
		<code>\c_novalue_tl</code> ..	688, 689, 690, 691,
			692, 693, 694, 2479, 2527, 2622, 2788
		<code>\tl_clear:N</code>	347, 371, 832,
			870, 883, 900, 909, 934, 943, 976,
			2603, 2866, 2876, 2899, 3784, 3785,
			3786, 3787, 3788, 3789, 3820, 4445,
			4446, 4447, 4448, 4449, 4495, 4512,
			4906, 4913, 4943, 5035, 5094, 5251
		<code>\tl_const:Nn</code>	1525
		<code>\tl_gclear:N</code>	364, 408
		<code>\tl_gset:Nn</code> ...	112, 357, 398, 742, 757
		<code>\tl_head:N</code>	
			.. 3618, 3631, 3643, 3645, 3655, 3657
		<code>\tl_head:n</code>	1865, 1916, 1958, 1962
		<code>\tl_if_empty:NTF</code>	35, 89,
			826, 836, 863, 874, 895, 902, 1009,
			1065, 1090, 1122, 1157, 1194, 1231,
			1279, 1327, 1333, 1366, 1444, 1482,
			1748, 1869, 1920, 2932, 2970, 2994,
			3025, 3060, 3097, 3134, 3182, 3257,
			3263, 3297, 3359, 3380, 3426, 4253,
			4846, 4928, 4950, 5008, 5019, 5062
		<code>\tl_if_empty:nTF</code>	
			.. 736, 752, 975, 1267, 1544, 1553,
			1716, 2432, 2724, 2898, 3170, 5139
		<code>\tl_if_empty_p:N</code>	
			.. 51, 56, 2592, 4219, 4290,
			4668, 4961, 4975, 5114, 5124, 5128
		<code>\tl_if_empty_p:n</code>	1378, 1411,
			2540, 2804, 3303, 3328, 3559, 3560,
			3569, 3570, 3595, 3596, 3611, 3626
		<code>\tl_if_eq:NNTF</code>	3530, 3553, 3860
		<code>\tl_if_eq:NnTF</code>	
			.. 1774, 3473, 3505, 3680, 3683,
			3708, 3711, 3828, 3876, 4910, 5160
		<code>\tl_if_eq:nnTF</code> ..	1764, 1776, 1788,
			1798, 1864, 1915, 1957, 3672, 5156,
			5162, 5184, 5188, 5203, 5511, 5519
T			
<code>\tablename</code>	138		
<code>\tag</code>	123, 131, 133		
T _E X and L ^A T _E X 2 _ε commands:			
<code>\@Alph</code>	126		
<code>\@addtoreset</code>	5		
<code>\@auxout</code>	5515, 5523		
<code>\@bsphack</code>	917, 5508		
<code>\@capttype</code>	5422, 5637		
<code>\@chapapp</code>	126		
<code>\@currentcounter</code> ...	2–5, 63, 129–		
	131, 134, 28, 29, 56, 57, 58, 2411, 2412		
<code>\@currentlabel</code>	3, 123, 129, 134		
<code>\@elt</code>	5		
<code>\@esphack</code>	967, 5528		
<code>\@ifl@t@r</code>	4		
<code>\@memsubcaption</code>	129		
<code>\@onlypreamble</code>	749, 763, 2893		
<code>\bbl@loaded</code>	55		
<code>\bbl@main@language</code>	24, 2079		
<code>\c@lstnumber</code>	134		
<code>\c@page</code>	7, 111		
<code>\caption@subtyphook</code>	5638		
<code>\hyper@link</code>	113, 122		
<code>\hyper@linkfile</code>	5141		
<code>\lst@AddToHook</code>	5587		
<code>\ltx@gobble</code>	63		
<code>\MT@newlabel</code>	5516, 5524		
<code>\protected@write</code>	5515, 5523		
<code>\zref@addprop</code>	22, 32, 47, 62, 64, 106, 116		
<code>\zref@default</code>	113, 4707, 4709		
<code>\zref@extractdefault</code>			
	11, 122, 279, 285, 289		
<code>\zref@ifpropundefined</code> ..	43, 1273,		
	1560, 1722, 2730, 3176, 5145, 5554		
<code>\zref@ifrefcontainsprop</code> ..	43, 45,		
	1750, 1758, 1817, 1835, 1847, 1892,		
	1934, 3881, 4712, 4795, 4849, 5148		
<code>\zref@ifrefundefined</code>			
	3480, 3482, 3494, 3849, 3851, 3856,		
	3873, 4255, 4266, 4467, 4790, 4904		
<code>\zref@label</code>	63, 2424		

<code>\tl_if_exist:N</code>	NTF	335, 345, 355, 362, 369, 380, 396, 406, 725
<code>\tl_if_exist_p:N</code>		2591
<code>\tl_if_novalue:nTF</code>		2482, 2530, 2625, 2791
<code>\tl_map_break:n</code>		99
<code>\tl_map_tokens:Nn</code>		91
<code>\tl_new:N</code>		107, 336, 346, 356, 363, 397, 407, 562, 563, 564, 714, 715, 716, 717, 741, 756, 1548, 2162, 2181, 2249, 2270, 2321, 2407, 3460, 3461, 3462, 3463, 3464, 3465, 3733, 3734, 3735, 3736, 3737, 3738, 3739, 3742, 3743, 3747, 3749, 3752, 3753, 3754, 3755, 3756, 3757, 3758, 3759, 3760, 3761, 3762, 3763, 5425
<code>\tl_put_left:Nn</code>		4340, 4347, 4390, 5021, 5022, 5064, 5066, 5068, 5070
<code>\tl_put_right:Nn</code>		4024, 4047, 4055, 4073, 4086, 4125, 4134, 4156, 4164, 4171, 4195, 4209, 4226, 4236, 4534, 4557, 4569, 4603, 4625, 4634, 4654, 4675, 4685, 4929, 4930, 4941, 5638
<code>\tl_reverse:N</code>		3540, 3543
<code>\tl_set:Nn</code>		278, 337, 718, 743, 933, 977, 989, 1557, 1564, 1566, 1755, 1823, 1827, 1840, 1851, 1856, 1867, 1868, 1876, 1878, 1896, 1901, 1918, 1919, 1927, 1929, 1938, 1943, 1964, 1965, 1973, 1975, 2078, 2079, 2084, 2085, 2088, 2089, 2103, 2109, 2114, 2140, 2146, 2151, 2601, 2867, 2900, 2912, 3647, 3649, 3830, 3831, 4004, 4006, 4278, 4301, 4311, 4357, 4480, 4482, 4493, 4510, 4925, 4926, 4939, 5426, 5428
<code>\tl_set_eq:NN</code>		416, 4443
<code>\tl_show:N</code>		4406
<code>\tl_tail:N</code>		3648, 3650
<code>\tl_tail:n</code>		1867, 1868, 1918, 1919, 1964, 1965
<code>\tl_use:N</code>		300, 311, 322, 758, 922, 927, 957, 959, 963
<code>\l_tmpa_tl</code>		931, 955, 1851, 1865, 1867, 1896, 1907, 1916, 1918, 1938, 1949, 1958, 1964, 3431, 3432
<code>\l_tmpb_tl</code>		1813, 1820, 1823, 1827, 1856, 1865, 1868, 1869, 1876, 1901, 1909, 1916, 1919, 1920, 1927, 1943, 1951, 1958, 1961, 1962, 1965, 1973
U		
use commands:		
<code>\use:N</code>		26, 29, 804, 4222, 4297, 4671, 5317
<code>\UseHook</code>		1850, 1895, 1937
V		
<code>\value</code>		5365, 5384
<code>\verbfootnote</code>		129, 130
Z		
<code>\Z</code>		1884
<code>\zcDeclareLanguage</code>		12, 25, 733, 5663, 5868, 6321, 6536, 6765, 6979, 7197
<code>\zcDeclareLanguageAlias</code>		26, 750, 5664, 5665, 5666, 5667, 5668, 5669, 5670, 5871, 5872, 5873, 5874, 5875, 6322, 6537, 6538, 6539
<code>\zcLanguageSetup</code>		20, 21, 29, 30, 68, 73, 74, 138, 2861
<code>\zcpageref</code>		85, 3452
<code>\zceref</code>		20, 65, 67, 83, 85–87, 93, 95, 132, 3404, 3457
<code>\zcRefTypeSetup</code>		20, 21, 68, 2599
<code>\zcsetup</code>		20, 55, 65, 67, 2594
<code>\zlabel</code>		2, 63, 129, 130, 133
zrefcheck commands:		
<code>\zrefcheck_zceref_beg_label:</code>		3417
<code>\zrefcheck_zceref_end_label_</code>		maybe: 3439
<code>\zrefcheck_zceref_run_checks_on_</code>		labels:n 3440
zrefclever commands:		
<code>\zrefclever_language_if_declared:n</code>		731
<code>\zrefclever_language_if_declared:nTF</code>		731
<code>\zrefclever_language_varname:n</code>		721, 721
<code>\l_zrefclever_ref_language_tl</code>		717
zrefclever internal commands:		
<code>\l_zrefclever_abbrev_bool</code>		3752, 3939, 4932
<code>\l_zrefclever_amsmath_subequations_</code>		bool 5453, 5467, 5491
<code>\l_zrefclever_breqn_dgroup_bool</code>		5538, 5552, 5567
<code>\l_zrefclever_cap_bool</code>		3752, 3935, 4920
<code>\l_zrefclever_capfirst_bool</code>		1985, 1988, 4922
<code>_zrefclever_compat_module:nn</code>		64, 2465, 2465, 5324, 5342, 5403, 5449, 5499, 5534, 5574, 5592, 5619, 5642
<code>_zrefclever_counter_reset_by:n</code>		6, 61, 62, 67, 69, 71, 75, 75, 5461, 5546

_zrefclever_counter_reset_by_-	4089, 4093, 4128, 4137, 4159, 4167,
aux:nn	4174, 4198, 4212, 4239, 4281, 4314,
_zrefclever_counter_reset_by_-	4349, 4537, 4560, 4572, 4606, 4628,
aux:nnn	4637, 4657, 4688, <u>4710</u> , 4710, 4749
\l_zrefclever_counter_resetby_-	_zrefclever_get_ref_endrange:nnN
prop	45, <u>113</u> ,
\l_zrefclever_counter_resettters_-	<u>114</u> , 4229, 4304, 4678, <u>4750</u> , 4750, 4787
seq	_zrefclever_get_ref_first: . . .
\l_zrefclever_counter_type_prop	113, <u>117</u> , 4341, 4391, <u>4788</u> , 4788
.	_zrefclever_get_rf_opt_bool:nN <u>126</u>
\l_zrefclever_current_counter_-	_zrefclever_get_rf_opt_-
tl	bool:nnnnN <u>20</u> ,
38, 41, 43, 51, 52, 53, 104, 2407, 2410	3932, 3936, 3940, <u>5291</u> , 5291, 5323
\l_zrefclever_current_language_-	_zrefclever_get_rf_opt_-
tl	seq:nnnN . . <u>20</u> , <u>125</u> , 3944, 3948,
55, 715, 2078, 2084, 2088, 2104, 2141	3952, 3956, 3960, 3964, 3968, 3972,
\l_zrefclever_endrangefunc_tl . .	3976, 3980, 4977, <u>5258</u> , 5258, 5290
. . . <u>3752</u> , 3927, 4219, 4220, 4222,	_zrefclever_get_rf_opt_tl:nnnN
4290, 4291, 4297, 4668, 4669, 4671	20,
\l_zrefclever_endrangeprop_tl . .	<u>22</u> , <u>45</u> , <u>124</u> , 3428, 3799, 3803, 3807,
. <u>48</u> , 1748, 1758, <u>3752</u> , 3931	3896, 3900, 3904, 3908, 3912, 3916,
_zrefclever_extract:nnn	3920, 3924, 3928, <u>5225</u> , 5225, 5257
. <u>11</u> , <u>288</u> , 288, 1853, 1858,	_zrefclever_hyperlink:nnn
1898, 1903, 1940, 1945, 3601, 3603,	122, 4363,
3616, 3633, 3721, 3723, 5167, 5169,	4721, 4762, 4799, 4861, <u>5137</u> , 5137
5173, 5175, 5193, 5195, 5199, 5201	\l_zrefclever_hyperlink_bool
_zrefclever_extract_default:Nnnn	2005, 2012, 2017, 2022, 2034, 2040,
. . . <u>11</u> , <u>276</u> , 276, 281, 1752, 1813,	2047, 3456, 4716, 4757, 4855, 5112
1820, 1830, 1837, 3514, 3525, 3527,	\l_zrefclever_hyperref_warn_-
3538, 3541, 3544, 3546, 3834, 3837	bool . . . 2006, 2013, 2018, 2023, 2038
_zrefclever_extract_unexp:nnn .	_zrefclever_if_class_loaded:n .
. <u>11</u> , <u>122</u> ,	117, 119
<u>282</u> , 282, 287, 1766, 1770, 1778,	_zrefclever_if_class_loaded:nTF
1782, 1790, 1794, 1800, 1804, 4369,	5405
4723, 4728, 4740, 4764, 4805, 4818,	_zrefclever_if_package_-
4867, 4875, 4890, 5146, 5149, 5150,	loaded:n <u>117</u> , 117
5157, 5158, 5163, 5164, 5185, 5186,	_zrefclever_if_package_-
5189, 5190, 5205, 5209, 5512, 5520	loaded:nTF <u>2032</u> ,
_zrefclever_extract_url_-	<u>2076</u> , <u>2082</u> , <u>2285</u> , <u>5344</u> , <u>5451</u> ,
unexp:n 4365, 4722,	<u>5501</u> , <u>5536</u> , <u>5576</u> , <u>5594</u> , <u>5621</u> , <u>5644</u>
4763, 4801, 4863, <u>5143</u> , 5143, 5153	_zrefclever_is_integer_rgx:n . .
_zrefclever_get_enclosing_-	1881, 1882, 1889
counters_value:n	_zrefclever_is_integer_rgx:nTF
. <u>6</u> , <u>65</u> , 65, 70, 74, 103	1907, 1909, 1949, 1951
_zrefclever_get_endrange_-	\l_zrefclever_label_a_tl
pagecomp:nnN 1890, 1931	. <u>92</u> , <u>3733</u> , 3817, 3836, 3849, 3871,
_zrefclever_get_endrange_-	3873, 3879, 3882, 3888, 4005, 4027,
pagecomptwo:nnN 1932, 1977	4050, 4058, 4093, 4159, 4174, 4224,
_zrefclever_get_endrange_-	4230, 4239, 4271, 4281, 4299, 4305,
property:nnN <u>45</u> , 1746, 1844	4314, 4467, 4471, 4481, 4494, 4511,
_zrefclever_get_endrange_-	4537, 4573, 4637, 4673, 4679, 4688
stripprefix:nnN 1845, 1880	\l_zrefclever_label_b_tl
_zrefclever_get_ref:nN	92, <u>3733</u> ,
. <u>113</u> , 4027, 4050, 4058, 4076,	3820, 3825, 3839, 3851, 3856, 4471

<code>\l_zrefclever_label_count_int ..</code>	<code>\l_zrefclever_last_of_type_bool</code>
..... 93 , 3730 , 3790 , 92 , 3727 , 3847 ,
3894 , 3997 , 4450 , 4476 , 4704 , 4964	3852 , 3853 , 3857 , 3863 , 3864 , 3987
<code>\l_zrefclever_label_enclval_a_-</code>	<code>\l_zrefclever_lastsep_tl ..</code> 3752 ,
tl 3460 , 3538 , 3540 , 3595 ,	3911 , 4057 , 4092 , 4136 , 4173 , 4211
3611 , 3631 , 3643 , 3647 , 3648 , 3655	<code>\l_zrefclever_link_star_bool ...</code>
<code>\l_zrefclever_label_enclval_b_-</code>	.. 3411 , 3450 , 4717 , 4758 , 4856 , 5113
tl 3460 , 3541 , 3543 , 3596 ,	<code>\l_zrefclever_listsep_tl</code>
3618 , 3626 , 3645 , 3649 , 3650 , 3657	... 3752 , 3907 , 4088 , 4166 , 4536 ,
<code>\l_zrefclever_label_extdoc_a_tl</code>	4559 , 4571 , 4605 , 4627 , 4636 , 4656
.. 3460 , 3544 , 3554 , 3559 , 3569 , 3582	<code>\g_zrefclever_loaded_langfiles_-</code>
<code>\l_zrefclever_label_extdoc_b_tl</code>	seq 913 , 921 , 956 , 962
.. 3460 , 3546 , 3555 , 3560 , 3570 , 3581	<code>\l_zrefclever_main_language_tl ..</code>
<code>\l_zrefclever_label_type_a_tl ..</code> 24 ,
..... 3429 , 3460 , 3515 , 3517 ,	55 , 716 , 2079 , 2085 , 2089 , 2110 , 2147
3520 , 3526 , 3531 , 3680 , 3708 , 3800 ,	<code>_zrefclever_mathtools_showonlyrefs:n</code>
3804 , 3808 , 3830 , 3835 , 3861 , 3876 , 3445 , 5506
3897 , 3901 , 3905 , 3909 , 3913 , 3917 ,	<code>\l_zrefclever_mathtools_-</code>
3921 , 3925 , 3929 , 3933 , 3937 , 3941 ,	showonlyrefs_bool 3443 , 5498 , 5505
3945 , 3949 , 3953 , 3957 , 3961 , 3965 ,	<code>\l_zrefclever_memoir_footnote_-</code>
3969 , 3973 , 3977 , 3981 , 4007 , 4483	type_tl 5425 , 5426 , 5428 , 5432
<code>\l_zrefclever_label_type_b_tl ..</code>	<code>_zrefclever_name_default:</code>
..... 3460 , 3528 , 4706 , 4708 , 4838
3532 , 3683 , 3711 , 3831 , 3838 , 3862	<code>\l_zrefclever_name_format_-</code>
<code>_zrefclever_label_type_put_-</code>	fallback_tl 3739 , 4939 ,
new_right:n 86 , 87 , 3476 , 3512 , 3512	4943 , 5008 , 5057 , 5069 , 5071 , 5089
<code>\l_zrefclever_label_types_seq ..</code>	<code>\l_zrefclever_name_format_tl ...</code>
.... 87 , 3469 , 3472 , 3516 , 3519 , 3706	... 3739 , 4925 , 4926 , 4929 , 4930 ,
<code>\l_zrefclever_labelhook_bool ...</code>	4940 , 4941 , 5014 , 5021 , 5022 , 5030 ,
..... 2414 , 2417 , 2423	5038 , 5048 , 5065 , 5066 , 5079 , 5099
<code>_zrefclever_labels_in_sequence:nn</code>	<code>\l_zrefclever_name_in_link_bool</code>
.. 48 , 93 , 123 , 4269 , 4470 , 5154 , 5154 115 ,
<code>\l_zrefclever_lang_decl_case_tl</code>	117 , 3739 , 4359 , 4793 , 5117 , 5133 , 5134
562 , 943 , 946 , 989 , 994 , 1333 , 1350 ,	<code>\l_zrefclever_namefont_tl</code> 3752 ,
2876 , 2879 , 2912 , 2917 , 3263 , 3280	3919 , 4362 , 4378 , 4810 , 4828 , 4843
<code>\l_zrefclever_lang_declension_-</code>	<code>\l_zrefclever_nameinlink_str ...</code>
seq 562 , 2050 , 2055 , 2057 ,
822 , 823 , 824 , 838 , 842 , 849 , 940 ,	2059 , 2061 , 5115 , 5121 , 5123 , 5127
941 , 942 , 945 , 982 , 988 , 993 , 2873 ,	<code>\l_zrefclever_namesep_tl</code>
2874 , 2875 , 2878 , 2905 , 2911 , 2916	.. 3752 , 3899 , 4813 , 4831 , 4839 , 4847
<code>\l_zrefclever_lang_gender_seq ..</code>	<code>\l_zrefclever_next_is_same_bool</code>
.... 562 , 859 , 860 , 861 , 876 , 953 , 93 , 122 , 3745 ,
954 , 1003 , 1018 , 2886 , 2887 , 2926 , 2941	4464 , 4497 , 4514 , 4520 , 5178 , 5216
<code>_zrefclever_language_if_-</code>	<code>\l_zrefclever_next_maybe_range_-</code>
declared:n 25 , 723 , 730 , 732	bool
<code>_zrefclever_language_if_-</code>	. 93 , 122 , 3745 , 4263 , 4276 , 4463 ,
declared:n(TF) 25	4490 , 4503 , 5170 , 5177 , 5196 , 5214
<code>_zrefclever_language_if_-</code>	<code>\l_zrefclever_noabbrev_first_-</code>
declared:nTF 297 , 308 , 319 , 723 ,	bool 1992 , 1995 , 4936
738 , 754 , 814 , 918 , 2115 , 2152 , 2864	<code>\g_zrefclever_nocompat_bool ...</code>
<code>_zrefclever_language_varname:n</code> 2426 , 2433 , 2469
..... 24 , 25 , 300 , 311 ,	<code>\l_zrefclever_nocompat_bool ...</code> 64
322 , 719 , 719 , 722 , 725 , 741 , 742 ,	<code>\g_zrefclever_nocompat_modules_-</code>
756 , 757 , 758 , 922 , 927 , 957 , 959 , 963	seq 2427 , 2437 , 2440 , 2461 , 2470 , 2471

```

\l_zrefclever_nocompat_modules_-
  seq ..... 64
\l_zrefclever_nudge_comptosing_-
  bool ... 2178, 2208, 2217, 2223, 4960
\l_zrefclever_nudge_enabled_-
  bool ..... 2176, 2186, 2188,
  2192, 2193, 2198, 2199, 4435, 4946
\l_zrefclever_nudge_gender_bool
  ..... 2180, 2209, 2219, 2224, 4974
\l_zrefclever_nudge_multitype_-
  bool ... 2177, 2207, 2215, 2222, 4436
\l_zrefclever_nudge_singular_-
  bool ..... 2179, 2235, 4948
\_zrefclever_opt_bool_get:NN ...
  ..... 543, 553
\_zrefclever_opt_bool_get:NN(TF)
  ..... 19
\_zrefclever_opt_bool_get:NNTF .
  ... 543, 5294, 5299, 5304, 5309, 5314
\_zrefclever_opt_bool_gset_-
  false:N ..... 19,
  509, 536, 542, 1491, 1508, 3382, 3390
\_zrefclever_opt_bool_gset_-
  true:N ..... 19,
  509, 529, 535, 1453, 1470, 3361, 3369
\_zrefclever_opt_bool_if:N 554, 561
\_zrefclever_opt_bool_if:N(TF) . 20
\_zrefclever_opt_bool_if:NNTF ...
  ..... 554, 887
\_zrefclever_opt_bool_if_set:N .
  ..... 493, 508
\_zrefclever_opt_bool_if_-
  set:N(TF) ..... 18
\_zrefclever_opt_bool_if_-
  set:NNTF ..... 493,
  545, 556, 1446, 1462, 1484, 1500
\_zrefclever_opt_bool_set_-
  false:N 19, 509, 519, 528, 2569, 2840
\_zrefclever_opt_bool_set_-
  true:N 19, 509, 509, 518, 2564, 2831
\_zrefclever_opt_bool_unset:N ..
  ..... 18, 482, 482, 492, 2574, 2849
\_zrefclever_opt_seq_get:NN 471, 481
\_zrefclever_opt_seq_get:NN(TF) 18
\_zrefclever_opt_seq_get:NNTF ..
  ... 471, 817, 854, 935, 948, 2868,
  2881, 5261, 5266, 5271, 5276, 5281
\_zrefclever_opt_seq_gset_-
  clist_split:Nn ..... 16,
  423, 425, 1375, 1408, 3300, 3325
\_zrefclever_opt_seq_gset_eq:NN
  ..... 16, 423,
  437, 443, 1384, 1417, 2949, 3309, 3334
\_zrefclever_opt_seq_if_set:N ..
  ..... 455, 470
\_zrefclever_opt_seq_if_-
  set:N(TF) ..... 17
\_zrefclever_opt_seq_if_set:NNTF
  ..... 455, 473, 1026, 1368, 1400
\_zrefclever_opt_seq_set_clist_-
  split:Nn .. 16, 423, 423, 2537, 2801
\_zrefclever_opt_seq_set_eq:NN .
  ..... 16, 423, 427, 436, 2543, 2807
\_zrefclever_opt_seq_unset:N ...
  ..... 17, 444, 444, 454, 2532, 2793
\_zrefclever_opt_tl_clear:N ...
  ..... 14, 333,
  343, 352, 1644, 1649, 1664, 1679,
  1694, 2652, 2657, 2672, 2687, 2702
\_zrefclever_opt_tl_cset_-
  fallback:nm ..... 1523, 1530
\_zrefclever_opt_tl_gclear:N ...
  ..... 14, 333,
  360, 366, 3027, 3033, 3041, 3048,
  3069, 3085, 3106, 3122, 3143, 3159
\_zrefclever_opt_tl_gclear_if_-
  new:N ..... 16, 392,
  402, 411, 1124, 1130, 1138, 1145,
  1166, 1182, 1203, 1219, 1240, 1256
\_zrefclever_opt_tl_get:NN 412, 422
\_zrefclever_opt_tl_get:NN(TF) . 16
\_zrefclever_opt_tl_get:NNTF ...
  412, 5010, 5025, 5044, 5053, 5074,
  5084, 5228, 5233, 5238, 5243, 5248
\_zrefclever_opt_tl_gset:N .... 14
\_zrefclever_opt_tl_gset:Nn ...
  .. 333, 353, 359, 2972, 2996, 3004,
  3062, 3077, 3099, 3114, 3136, 3151,
  3184, 3191, 3200, 3208, 3265, 3275
\_zrefclever_opt_tl_gset_if_-
  new:Nn ..... 16,
  392, 392, 401, 1067, 1092, 1101,
  1159, 1174, 1196, 1211, 1233, 1248,
  1281, 1288, 1297, 1305, 1335, 1345
\_zrefclever_opt_tl_if_set:N .. 378
\_zrefclever_opt_tl_if_set:N(TF)
  ..... 15
\_zrefclever_opt_tl_if_set:NNTF .
  ..... 378, 394, 404, 414
\_zrefclever_opt_tl_set:N ..... 14
\_zrefclever_opt_tl_set:Nn ....
  ..... 333, 333, 342,
  1658, 1673, 1688, 1728, 1734, 2488,
  2634, 2666, 2681, 2696, 2736, 2743
\_zrefclever_opt_tl_unset:N ...
  ..... 15, 367, 367,
  377, 1703, 1708, 2484, 2627, 2711, 2716

```

```

\__zrefclever_opt_var_set_bool:n
..... 14, 331, 331, 338,
339, 340, 348, 349, 350, 372, 373,
374, 382, 384, 432, 433, 434, 449,
450, 451, 459, 461, 487, 488, 489,
497, 499, 514, 515, 516, 524, 525, 526
\__zrefclever_opt_varname_-
  fallback:nn ..... 14,
329, 329, 1526, 5249, 5282, 5315
\__zrefclever_opt_varname_-
  general:nn ..... 12,
290, 290, 1646, 1651, 1660, 1666,
1675, 1681, 1690, 1696, 1705, 1710,
1730, 1736, 2485, 2489, 2533, 2544,
2565, 2570, 2575, 5229, 5262, 5295
\__zrefclever_opt_varname_lang_-
  default:nnn .. 13, 306, 306, 316,
1069, 1094, 1126, 1132, 1161, 1168,
1198, 1205, 1235, 1242, 1283, 1290,
1370, 1386, 1448, 1455, 1486, 1493,
2974, 2998, 3029, 3035, 3064, 3071,
3101, 3108, 3138, 3145, 3186, 3193,
3311, 3363, 3384, 5244, 5277, 5310
\__zrefclever_opt_varname_lang_-
  type:nnnn ..... 13,
317, 317, 328, 1028, 1037, 1045,
1103, 1140, 1147, 1176, 1184, 1213,
1221, 1250, 1258, 1299, 1307, 1337,
1347, 1402, 1419, 1464, 1472, 1502,
1510, 2951, 3006, 3043, 3050, 3079,
3087, 3116, 3124, 3153, 3161, 3202,
3210, 3267, 3277, 3336, 3371, 3392,
5027, 5076, 5086, 5239, 5272, 5305
\__zrefclever_opt_varname_-
  language:nnn ..... 12, 295,
295, 305, 770, 775, 785, 790, 801,
806, 819, 856, 889, 937, 950, 2870, 2883
\__zrefclever_opt_varname_-
  type:nnn 12, 292, 292, 294, 2629,
2636, 2654, 2659, 2668, 2674, 2683,
2689, 2698, 2704, 2713, 2718, 2738,
2745, 2795, 2809, 2833, 2842, 2851,
5012, 5046, 5055, 5234, 5267, 5300
\g__zrefclever_page_format_tl ...
..... 7, 107, 112, 115
\l__zrefclever_pairsep_tl .....
..... 3752, 3903, 4026,
4049, 4075, 4127, 4158, 4197, 4280
\__zrefclever_process_language_-
  settings: .. 57, 58, 812, 812, 3413
\__zrefclever_prop_put_non_-
  empty:Nnn 42, 1542, 1542, 2339, 2394
\__zrefclever_provide_langfile:n
..... 21,
30, 31, 84, 914, 914, 970, 2121, 3412
\l__zrefclever_range_beg_is_-
  first_bool ..... 3745,
3795, 4114, 4150, 4184, 4455,
4492, 4548, 4593, 4615, 4644, 4697
\l__zrefclever_range_beg_label_-
  tl ..... 93,
3745, 3788, 4077, 4090, 4129, 4138,
4168, 4199, 4213, 4223, 4448, 4493,
4510, 4561, 4607, 4629, 4658, 4672
\l__zrefclever_range_count_int ..
..... 93,
3745, 3793, 4039, 4103, 4453, 4496,
4507, 4513, 4519, 4527, 4586, 4699
\l__zrefclever_range_end_ref_tl .
... 3745, 3789, 4225, 4231, 4300,
4306, 4449, 4495, 4512, 4674, 4680
\l__zrefclever_range_same_count_-
  int ..... 93,
3745, 3794, 4017, 4068, 4104, 4454,
4498, 4515, 4521, 4566, 4587, 4700
\l__zrefclever_rangeseptl .....
..... 3752, 3915,
4228, 4238, 4303, 4313, 4677, 4687
\l__zrefclever_rangetopair_bool .
..... 3752, 3943, 4264
\l__zrefclever_ref_count_int ...
..... 3730, 3792,
4045, 4123, 4193, 4451, 4484, 4533,
4556, 4568, 4602, 4624, 4633, 4653
\l__zrefclever_ref_decl_case_tl .
..... 27, 826, 831, 832, 836, 839,
843, 847, 850, 895, 898, 900, 2162,
2172, 5019, 5023, 5062, 5067, 5072
\__zrefclever_ref_default: 4706,
4706, 4747, 4753, 4791, 4832, 4898
\l__zrefclever_ref_gender_tl ...
..... 28, 863, 869,
870, 874, 877, 882, 883, 902, 908,
909, 2181, 2245, 4975, 4983, 4989, 4997
\l__zrefclever_ref_language_tl ..
..... 24, 27, 55, 714, 718, 815,
820, 830, 848, 857, 867, 881, 890,
899, 906, 2103, 2109, 2114, 2122,
2140, 2146, 2151, 3412, 3430, 3801,
3805, 3809, 3898, 3902, 3906, 3910,
3914, 3918, 3922, 3926, 3930, 3934,
3938, 3942, 3946, 3950, 3954, 3958,
3962, 3966, 3970, 3974, 3978, 3982,
4979, 4991, 5002, 5028, 5077, 5087
\l__zrefclever_ref_property_tl ..
..... 43, 48, 1548,
1557, 1564, 1566, 1750, 1774, 1818,
1835, 1847, 1854, 1859, 1892, 1899,

```

1904, 1934, 1941, 1946, 3505, 3828,
 3883, 3887, 4712, 4797, 4851, 5160
 \l_zrefclever_ref_propserity_tl 3473
 \l_zrefclever_ref_typeset_font_-
 tl 2249, 2251, 3423
 \l_zrefclever_refbounds_first_-
 pb_seq 3767,
 3955, 4031, 4081, 4153, 4204, 4285
 \l_zrefclever_refbounds_first_-
 rb_seq . 3767, 3959, 4187, 4319, 4648
 \l_zrefclever_refbounds_first_-
 seq 3767, 3947, 4330, 4551, 4597, 4619
 \l_zrefclever_refbounds_first_-
 sg_seq . 3767, 3951, 4009, 4020, 4117
 \l_zrefclever_refbounds_last_-
 pe_seq 3767, 3979,
 4028, 4051, 4078, 4130, 4160, 4282
 \l_zrefclever_refbounds_last_-
 re_seq
 .. 3767, 3983, 4232, 4240, 4307, 4315
 \l_zrefclever_refbounds_last_-
 seq 3767, 3975, 4059, 4094, 4139, 4175
 \l_zrefclever_refbounds_mid_rb_-
 seq ... 3767, 3967, 4200, 4214, 4659
 \l_zrefclever_refbounds_mid_re_-
 seq 3767, 3971, 4681, 4689
 \l_zrefclever_refbounds_mid_seq
 3767, 3963, 4091, 4169,
 4538, 4562, 4574, 4608, 4630, 4638
 \l_zrefclever_reffont_tl
 3752, 3923, 4727,
 4739, 4768, 4779, 4817, 4874, 4889
 \l_zrefclever_reftype_override_-
 tl 35, 45, 2321, 2324
 \g_zrefclever_rf_opts_bool_-
 maybe_type_specific_seq
 .. 52, 53, 567, 1437, 2557, 2824, 3352
 \g_zrefclever_rf_opts_seq_-
 refbounds_seq
 567, 1359, 2523, 2784, 3290
 \g_zrefclever_rf_opts_tl_maybe_-
 type_specific_seq 567, 1083, 2987
 \g_zrefclever_rf_opts_tl_not_-
 type_specific_seq
 567, 1058, 2606, 2963
 \g_zrefclever_rf_opts_tl_-
 reference_seq 567, 2475
 \g_zrefclever_rf_opts_tl_type_-
 names_seq 567, 1320, 3250
 \g_zrefclever_rf_opts_tl_-
 typesetup_seq 567, 2618
 \l_zrefclever_setup_language_tl
 562, 743, 771, 776, 786,
 791, 802, 807, 933, 985, 992, 1006,
 1023, 1029, 1038, 1046, 1070, 1095,
 1104, 1127, 1133, 1141, 1148, 1162,
 1169, 1177, 1185, 1199, 1206, 1214,
 1222, 1236, 1243, 1251, 1259, 1284,
 1291, 1300, 1308, 1338, 1348, 1371,
 1387, 1403, 1420, 1449, 1456, 1465,
 1473, 1487, 1494, 1503, 1511, 2867,
 2908, 2915, 2929, 2946, 2952, 2975,
 2999, 3007, 3030, 3036, 3044, 3051,
 3065, 3072, 3080, 3088, 3102, 3109,
 3117, 3125, 3139, 3146, 3154, 3162,
 3187, 3194, 3203, 3211, 3268, 3278,
 3312, 3337, 3364, 3372, 3385, 3393
 \l_zrefclever_setup_type_tl 562,
 934, 976, 977, 1009, 1030, 1039,
 1047, 1065, 1090, 1105, 1122, 1142,
 1149, 1157, 1178, 1186, 1194, 1215,
 1223, 1231, 1252, 1260, 1279, 1301,
 1309, 1327, 1339, 1349, 1366, 1404,
 1421, 1444, 1466, 1474, 1482, 1504,
 1512, 2601, 2603, 2630, 2637, 2655,
 2660, 2669, 2675, 2684, 2690, 2699,
 2705, 2714, 2719, 2739, 2746, 2796,
 2810, 2834, 2843, 2852, 2866, 2899,
 2900, 2932, 2953, 2970, 2994, 3008,
 3025, 3045, 3052, 3060, 3081, 3089,
 3097, 3118, 3126, 3134, 3155, 3163,
 3182, 3204, 3212, 3257, 3269, 3279,
 3297, 3338, 3359, 3373, 3380, 3394
 \l_zrefclever_sort_decided_bool
 3466, 3549, 3563, 3573,
 3577, 3589, 3599, 3614, 3629, 3653
 _zrefclever_sort_default:nn ...
 87, 3507, 3523, 3523
 _zrefclever_sort_default_-
 different_types:nn
 44, 86, 90, 3534, 3666, 3666
 _zrefclever_sort_default_same_-
 type:nn ... 85, 88, 3533, 3536, 3536
 _zrefclever_sort_labels:
 86, 87, 91, 3421, 3470, 3470
 _zrefclever_sort_page:nn
 91, 3506, 3718, 3718
 \l_zrefclever_sort_prior_a_int .
 3467,
 3668, 3674, 3675, 3681, 3691, 3699
 \l_zrefclever_sort_prior_b_int .
 3467,
 3669, 3676, 3677, 3684, 3692, 3700
 \l_zrefclever_tlastsep_tl
 3752, 3810, 4429
 \l_zrefclever_tlistsep_tl
 3752, 3806, 4400
 \l_zrefclever_tpairsep_tl


```

..... 3752, 3802, 4422
\l_zrefclever_type_count_int ...
. 93, 117, 3730, 3791, 4397, 4399,
4412, 4437, 4452, 4923, 4935, 5130
\l_zrefclever_type_first_label_-
tl ..... 92,
115, 3733, 3786, 4004, 4255, 4266,
4270, 4298, 4349, 4366, 4370, 4446,
4480, 4790, 4796, 4802, 4806, 4819,
4850, 4864, 4868, 4876, 4891, 4904
\l_zrefclever_type_first_label_-
type_tl ... 93, 117, 3733, 3787,
4006, 4259, 4447, 4482, 4911, 4954,
4970, 4978, 4990, 4996, 5013, 5029,
5039, 5047, 5056, 5078, 5088, 5100
\l_zrefclever_type_first_-
refbounds_seq .....
... 3767, 4008, 4019, 4030, 4080,
4116, 4152, 4186, 4203, 4284, 4318,
4329, 4350, 4550, 4596, 4618, 4647,
4814, 4815, 4822, 4824, 4860, 4872,
4880, 4883, 4886, 4887, 4894, 4895
\l_zrefclever_type_first_-
refbounds_set_bool ..... 3767,
3796, 4010, 4021, 4032, 4083, 4119,
4155, 4189, 4206, 4286, 4320,
4327, 4456, 4553, 4599, 4621, 4650
\l_zrefclever_type_name_gender_-
seq ... 3739, 4980, 4982, 4985, 5000
\l_zrefclever_type_name_-
missing_bool .....
.. 3739, 4836, 4907, 4914, 5036, 5096
\l_zrefclever_type_name_setup: ..
..... 20, 22, 115, 4336, 4902, 4902
\l_zrefclever_type_name_tl ....
..... 115, 117,
3739, 4373, 4379, 4811, 4829, 4844,
4846, 4906, 4913, 5017, 5033, 5035,
5051, 5060, 5082, 5092, 5094, 5114
\l_zrefclever_typeset_compress_-
bool ..... 1626, 1629, 4465
\l_zrefclever_typeset_labels_-
seq 92, 3727, 3782, 3816, 3818, 3824
\l_zrefclever_typeset_last_bool
..... 92, 3727,
3813, 3814, 3821, 3846, 4409, 5129
\l_zrefclever_typeset_name_bool
.. 1575, 1582, 1587, 1592, 4338, 4354

\l_zrefclever_typeset_queue_-
curr_tl ..... 93,
95, 113, 117, 3733, 3785, 4024,
4047, 4055, 4073, 4086, 4125,
4134, 4156, 4164, 4171, 4195, 4209,
4226, 4236, 4253, 4278, 4301, 4311,
4340, 4347, 4357, 4390, 4406, 4417,
4423, 4430, 4444, 4445, 4534, 4557,
4569, 4603, 4625, 4634, 4654, 4675,
4685, 4928, 4950, 4961, 5124, 5128
\l_zrefclever_typeset_queue_-
prev_tl . 93, 3733, 3784, 4401, 4443
\l_zrefclever_typeset_range_-
bool ... 1760, 1978, 1981, 3420, 4251
\l_zrefclever_typeset_ref_bool .
.. 1574, 1581, 1586, 1591, 4338, 4344
\l_zrefclever_typeset_refs: .....
..... 92-95, 3424, 3780, 3780
\l_zrefclever_typeset_refs_last_-
of_type: .....
.. 99, 113, 115, 117, 3989, 3994, 3994
\l_zrefclever_typeset_refs_not_-
last_of_type: .....
... 93, 99, 113, 122, 3991, 4459, 4459
\l_zrefclever_typeset_sort_bool
..... 1602, 1605, 3419
\l_zrefclever_typesort_seq ....
. 44, 90, 1611, 1616, 1617, 1623, 3670
\l_zrefclever_verbose_testing_-
bool ..... 3779, 4405
\l_zrefclever_zceref:nnn .....
..... 27, 56, 3405, 3406
\l_zrefclever_zceref:nnnn 83, 86, 3406
\l_zrefclever_zceref_labels_seq .
..... 86, 87, 3410,
3441, 3446, 3450, 3475, 3478, 3783
\l_zrefclever_zceref_note_tl ...
..... 2270, 2273, 3426, 3433
\l_zrefclever_zceref_with_check_-
bool ..... 2277, 2294, 3416, 3437
\l_zrefclever_zcsetup:n .....
. 67, 2595, 2596, 2596, 2598, 5328,
5352, 5358, 5366, 5388, 5407, 5431,
5435, 5436, 5445, 5456, 5490, 5541,
5566, 5578, 5588, 5603, 5623, 5646
\l_zrefclever_zrefcheck_-
available_bool .....
.. 2276, 2289, 2301, 2313, 3415, 3436

```