

The zref-clever package*

Code documentation

Gustavo Barros[†]

2023-08-14

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	30
4.8	Options	43
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 v0.4.2, released 2023-08-14.

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

9	Compatibility	127
9.1	appendix	127
9.2	appendices	128
9.3	memoir	129
9.4	amsmath	130
9.5	mathtools	132
9.6	breqn	132
9.7	listings	133
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 (ltxcmdhooks), 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 this latter kernel release.

- 3 `\def\zrefclever@required@kernel{2023-06-01}`
- 4 `\NeedsTeXFormat{LaTeX2e}[\zrefclever@required@kernel]`

```

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

Identify the package.

14 \ProvidesExplPackage {zref-clever} {2023-08-14} {0.4.2}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

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

```

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

```

3 zref setup

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

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

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

```

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

```

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

```

22 \zref@newprop { thecounter }
23   {

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

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

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

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

tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

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

```

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

```

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

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

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

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

```

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

```

```

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

```

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

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

```

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

```

Another piece of information we need is the page numbering format being used by `\thepage`, so that we know when we can (or not) group a set of page references in a range. Unfortunately, `page` is not a typical counter in ways which complicates things. First, it does commonly get reset along the document, not necessarily by the usual counter reset chains, but rather with `\pagenumbering` or variations thereof. Second, the format of the page number commonly changes in the document (roman, arabic, etc.), not necessarily, though usually, together with a reset. Trying to “parse” `\thepage` to retrieve such information is bound to go wrong: we don’t know, and can’t know, what is within that macro, and that’s the business of the user, or of the documentclass, or of the loaded packages. The technique used by `cleveref`, is simple and smart: store with the label what `\thepage` would return, if the counter `\c@page` was “1”. That would not allow us to *sort* the references, luckily however, we have `abspage` which solves this problem. But we can decide whether two labels can be compressed into a range or not based on this format: if they are identical, we can compress them, otherwise, we can’t. However, `x` expanding `\thepage` can lead to errors for some `babel` packages which redefine `\roman` containing non-expandable material (see <https://chat.stackexchange.com/transcript/message/63810027#63810027>, <https://chat.stackexchange.com/transcript/message/63810318#63810318>, <https://chat.stackexchange.com/transcript/message/63810720#63810720> and discussion). So I went for something a little different. As mentioned, we want to know if `\thepage` is the same for different labels, or if it has changed. We can thus test this directly, by comparing `\thepage` with a stored value of it, `\g__zrefclever_prev_page_format_tl`, and stepping a counter every time they differ. Of course, this cannot be done at label setting time, since it is not expandable. But we can do that comparison before shipout and then define the label property as starred (`\zref@newprop*{zc@pgfmt}`), so that the label comes after the counter, and we can get the correct value of the counter.

```

106 \int_new:N \g__zrefclever_page_format_int
107 \tl_new:N \g__zrefclever_prev_page_format_tl
108 \AddToHook { shipout / before }

```

```

109 {
110   \tl_if_eq:NNF \g__zrefclever_prev_page_format_tl \thepage
111   {
112     \int_gincr:N \g__zrefclever_page_format_int
113     \tl_gset_eq:NN \g__zrefclever_prev_page_format_tl \thepage
114   }
115 }
116 \zref@newprop* { zc@pgfmt } { \int_use:N \g__zrefclever_page_format_int }
117 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

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

4 Plumbing

4.1 Auxiliary

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

```

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

```

(End of definition for `_zrefclever_if_package_loaded:n` and `_zrefclever_if_class_loaded:n`.)

4.2 Messages

```

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

```



```

144 {
145   Language~'#1'~is-unknown~\msg_line_context:..~Can't-set-it-up.~
146   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
147   '\iow_char:N\zcDeclareLanguageAlias'.
148 }
149 \msg_new:nnn { zref-clever } { unknown-language-opt }
150 {
151   Language~'#1'~is-unknown~\msg_line_context:..~
152   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
153   '\iow_char:N\zcDeclareLanguageAlias'.
154 }
155 \msg_new:nnn { zref-clever } { unknown-language-decl }
156 {
157   Can't-set-declension~'#1'~for-unknown-language~'#2'~\msg_line_context:..~
158   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
159   '\iow_char:N\zcDeclareLanguageAlias'.
160 }
161 \msg_new:nnn { zref-clever } { language-no-decl-ref }
162 {
163   Language~'#1'~has-no-declared-declension-cases~\msg_line_context:..~
164   Nothing-to-do-with-option~'d=#2'.
165 }
166 \msg_new:nnn { zref-clever } { language-no-gender }
167 {
168   Language~'#1'~has-no-declared-gender~\msg_line_context:..~
169   Nothing-to-do-with-option~'#2=#3'.
170 }
171 \msg_new:nnn { zref-clever } { language-no-decl-setup }
172 {
173   Language~'#1'~has-no-declared-declension-cases~\msg_line_context:..~
174   Nothing-to-do-with-option~'case=#2'.
175 }
176 \msg_new:nnn { zref-clever } { unknown-decl-case }
177 {
178   Declension-case~'#1'~unknown-for-language~'#2'~\msg_line_context:..~
179   Using-default-declension-case.
180 }
181 \msg_new:nnn { zref-clever } { nudge-multitype }
182 {
183   Reference-with-multiple-types~\msg_line_context:..~
184   You-may-wish-to-separate-them-or-review-language-around-it.
185 }
186 \msg_new:nnn { zref-clever } { nudge-comptosing }
187 {
188   Multiple-labels-have-been-compressed-into-singular-type-name~
189   for-type~'#1'~\msg_line_context:..
190 }
191 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
192 {
193   Option~'sg'~signals-that-a-singular-type-name-was-expected~
194   \msg_line_context:..~But-type~'#1'~has-plural-type-name.
195 }
196 \msg_new:nnn { zref-clever } { gender-not-declared }
197 { Language~'#1'~has-no~'#2'~gender-declared~\msg_line_context:.. }

```

```

198 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
199 {
200   Gender~mismatch~for~type~'#1'~\msg_line_context:..
201   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
202 }
203 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
204 {
205   You've~specified~'g=#1'~\msg_line_context:..
206   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
207 }
208 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
209 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
210 \msg_new:nnn { zref-clever } { option-document-only }
211 { Option~'#1'~is~only~available~after~\iow_char:N\begin\{document\}. }
212 \msg_new:nnn { zref-clever } { langfile-loaded }
213 { Loaded~'#1'~language~file. }
214 \msg_new:nnn { zref-clever } { zref-property-undefined }
215 {
216   Option~'ref=#1'~requested~\msg_line_context:..
217   But~the~property~'#1'~is~not~declared,~falling~back~to~'default'.
218 }
219 \msg_new:nnn { zref-clever } { endrange-property-undefined }
220 {
221   Option~'endrange=#1'~requested~\msg_line_context:..
222   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
223 }
224 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
225 {
226   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:..
227   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
228   '\iow_char:N\zcref'.
229 }
230 \msg_new:nnn { zref-clever } { missing-hyperref }
231 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
232 \msg_new:nnn { zref-clever } { option-preamble-only }
233 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
234 \msg_new:nnn { zref-clever } { unknown-compat-module }
235 {
236   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
237   Nothing~to~do.
238 }
239 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
240 {
241   The~value~of~option~'#1'~must~be~a~comma~separated~list~
242   of~four~items.~We~received~'#2'~items~\msg_line_context:..
243   Option~not~set.
244 }
245 \msg_new:nnn { zref-clever } { missing-zref-check }
246 {
247   Option~'check'~requested~\msg_line_context:..
248   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
249 }
250 \msg_new:nnn { zref-clever } { zref-check-too-old }
251 {

```

```

252 Option~'check'~requested~\msg_line_context:.~
253 But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
254 }
255 \msg_new:nnn { zref-clever } { missing-type }
256 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
257 \msg_new:nnn { zref-clever } { missing-property }
258 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
259 \msg_new:nnn { zref-clever } { missing-name }
260 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
261 \msg_new:nnn { zref-clever } { single-element-range }
262 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
263 \msg_new:nnn { zref-clever } { compat-package }
264 { Loaded~support~for~'#1'~package. }
265 \msg_new:nnn { zref-clever } { compat-class }
266 { Loaded~support~for~'#1'~documentclass. }
267 \msg_new:nnn { zref-clever } { option-deprecated }
268 {
269 Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
270 Use~'#2'~instead.
271 }
272 \msg_new:nnn { zref-clever } { load-time-options }
273 {
274 'zref-clever'~does~not~accept~load~time~options.~
275 To~configure~package~options,~use~'\iow_char:N\zcsetup'.
276 }

```

4.3 Data extraction

`_zrefclever_extract_default:Nmn`

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

```

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

```

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

`_zrefclever_extract_unexp:nmn`

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

```

    \_zrefclever_extract_unexp:nnn{<label>}{<prop>}{<default>}
283 \cs_new:Npn \_zrefclever_extract_unexp:nnn #1#2#3
284 {
285   \exp_args:NNNo \exp_args:No
286   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
287 }
288 \cs_generate_variant:Nn \_zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

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

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

```
\_zrefclever_extract:nnn{<label>}{<prop>}{<default>}
```

```
289 \cs_new:Npn \_zrefclever_extract:nnn #1#2#3
290 { \zref@extractdefault {#1} {#2} {#3} }
```

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

4.4 Option *infra*

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

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

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

```
\_zrefclever_opt_varname_general:nn {<option>} {<data type>}
```

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

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

`_zrefclever_opt_varname_type:nnn` Defines, and leaves in the input stream, the csname of the variable used to store the type-specific *<option>* for *<ref type>*.

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

```
293 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
294 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
295 \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 {<lang>} {<option>} {<data type>}
296 \cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
297 {
298   \__zrefclever_language_if_declared:nTF {#1}
299   {
300     g__zrefclever_opt_language_
301     \tl_use:c { \__zrefclever_language_varname:n {#1} }
302     _ #2 _ #3
303   }
304   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
305 }
306 \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>}
307 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
308 {
309   \__zrefclever_language_if_declared:nTF {#1}
310   {
311     g__zrefclever_opt_lang_
312     \tl_use:c { \__zrefclever_language_varname:n {#1} }
313     _default_ #2 _ #3
314   }
315   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
316 }
317 \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>}
318 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
319 {
320   \__zrefclever_language_if_declared:nTF {#1}
321   {
322     g__zrefclever_opt_lang_
323     \tl_use:c { \__zrefclever_language_varname:n {#1} }
324     _type_ #2 _ #3 _ #4
325   }

```

```

326     { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
327   }
328 \cs_generate_variant:Nn
329   \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }

```

(End of definition for __zrefclever_opt_varname_lang_type:nnnn.)

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

```

    \__zrefclever_opt_varname_fallback:nn {<option>} {<data type>}
330 \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
331   { 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>}
332 \cs_new:Npn \__zrefclever_opt_var_set_bool:n #1
333   { \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>}
334 \cs_new_protected:Npn \__zrefclever_opt_tl_set:Nn #1#2
335   {
336     \tl_if_exist:NF #1
337     { \tl_new:N #1 }
338     \tl_set:Nn #1 {#2}
339     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
340     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
341     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
342   }
343 \cs_generate_variant:Nn \__zrefclever_opt_tl_set:Nn { cn }
344 \cs_new_protected:Npn \__zrefclever_opt_tl_clear:N #1
345   {
346     \tl_if_exist:NF #1
347     { \tl_new:N #1 }
348     \tl_clear:N #1
349     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }

```

```

350     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
351     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
352   }
353 \cs_generate_variant:Nn \__zrefclever_opt_tl_clear:N { c }
354 \cs_new_protected:Npn \__zrefclever_opt_tl_gset:Nn #1#2
355   {
356     \tl_if_exist:NF #1
357     { \tl_new:N #1 }
358     \tl_gset:Nn #1 {#2}
359   }
360 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset:Nn { cn }
361 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear:N #1
362   {
363     \tl_if_exist:NF #1
364     { \tl_new:N #1 }
365     \tl_gclear:N #1
366   }
367 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear:N { c }

```

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

__zrefclever_opt_tl_unset:N Unset *<option tl>*.

```

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

```

(End of definition for __zrefclever_opt_tl_unset:N.)

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

```

    \__zrefclever_opt_tl_if_set:N(TF) {<option tl>} {<true>} {<false>}
379 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
380   {
381     \tl_if_exist:NTF #1
382     {
383       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
384       {
385         \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
386         { \prg_return_true: }
387         { \prg_return_false: }

```

```

388     }
389     { \prg_return_true: }
390   }
391   { \prg_return_false: }
392 }

```

(End of definition for `_zrefclever_opt_tl_if_set:NTF`.)

```

\_zrefclever_opt_tl_gset_if_new:Nn    \_zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\_zrefclever_opt_tl_gclear_if_new:N   \_zrefclever_opt_tl_gclear_if_new:N {<option tl>}
393 \cs_new_protected:Npn \_zrefclever_opt_tl_gset_if_new:Nn #1#2
394 {
395   \_zrefclever_opt_tl_if_set:NF #1
396   {
397     \tl_if_exist:NF #1
398     { \tl_new:N #1 }
399     \tl_gset:Nn #1 {#2}
400   }
401 }
402 \cs_generate_variant:Nn \_zrefclever_opt_tl_gset_if_new:Nn { cn }
403 \cs_new_protected:Npn \_zrefclever_opt_tl_gclear_if_new:N #1
404 {
405   \_zrefclever_opt_tl_if_set:NF #1
406   {
407     \tl_if_exist:NF #1
408     { \tl_new:N #1 }
409     \tl_gclear:N #1
410   }
411 }
412 \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>}
413 \prg_new_protected_conditional:Npnn \_zrefclever_opt_tl_get:NN #1#2 { F }
414 {
415   \_zrefclever_opt_tl_if_set:NTF #1
416   {
417     \tl_set_eq:NN #2 #1
418     \prg_return_true:
419   }
420   { \prg_return_false: }
421 }
422 \prg_generate_conditional_variant:Nnn \_zrefclever_opt_tl_get:NN { cN } { F }
423

```

(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>}

```



```

424 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
425   { \seq_set_split:Nnn #1 { , } {#2} }
426 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
427   { \seq_gset_split:Nnn #1 { , } {#2} }
428 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
429   {
430     \seq_if_exist:NF #1
431     { \seq_new:N #1 }
432     \seq_set_eq:NN #1 #2
433     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
434     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
435     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
436   }
437 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
438 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
439   {
440     \seq_if_exist:NF #1
441     { \seq_new:N #1 }
442     \seq_gset_eq:NN #1 #2
443   }
444 \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 $\rangle$ }
445 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
446   {
447     \seq_if_exist:NT #1
448     {
449       \seq_clear:N #1 % ?
450       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
451       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
452       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
453     }
454   }
455 \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 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 $\rangle$ }
456 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
457   {
458     \seq_if_exist:N $\mathit{TF}$  #1
459     {
460       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
461       {
462         \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
463         { \prg_return_true: }
464         { \prg_return_false: }

```

```

465     }
466     { \prg_return_true: }
467   }
468   { \prg_return_false: }
469 }
470 \prg_generate_conditional_variant:Nnn
471 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End of definition for __zrefclever_opt_seq_if_set:NTF.)

```

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

```

(End of definition for __zrefclever_opt_seq_get:NNTF.)

__zrefclever_opt_bool_unset:N Unset <option bool>.

```

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

```

(End of definition for __zrefclever_opt_bool_unset:N.)

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

```

\__zrefclever_opt_bool_if_set:N(TF) {<option bool>} {<true>} {<false>}
494 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
495 {
496   \bool_if_exist:NTF #1
497   {
498     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
499     {
500       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }

```

```

501         { \prg_return_true: }
502         { \prg_return_false: }
503     }
504     { \prg_return_true: }
505 }
506 { \prg_return_false: }
507 }
508 \prg_generate_conditional_variant:Nnn
509 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End of definition for __zrefclever_opt_bool_if_set:NTF.)

```

\__zrefclever_opt_bool_set_true:N {<option bool>}
\__zrefclever_opt_bool_set_false:N {<option bool>}
\__zrefclever_opt_bool_gset_true:N {<option bool>}
\__zrefclever_opt_bool_gset_false:N {<option bool>}
510 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
511 {
512     \bool_if_exist:NF #1
513     { \bool_new:N #1 }
514     \bool_set_true:N #1
515     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
516     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
517     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
518 }
519 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
520 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
521 {
522     \bool_if_exist:NF #1
523     { \bool_new:N #1 }
524     \bool_set_false:N #1
525     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
526     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
527     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
528 }
529 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
530 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
531 {
532     \bool_if_exist:NF #1
533     { \bool_new:N #1 }
534     \bool_gset_true:N #1
535 }
536 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
537 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
538 {
539     \bool_if_exist:NF #1
540     { \bool_new:N #1 }
541     \bool_gset_false:N #1
542 }
543 \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>}

```

```

544 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
545 {
546   \__zrefclever_opt_bool_if_set:NTF #1
547   {
548     \bool_set_eq:NN #2 #1
549     \prg_return_true:
550   }
551   { \prg_return_false: }
552 }
553 \prg_generate_conditional_variant:Nnn
554 \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End of definition for __zrefclever_opt_bool_get:NNTF.)

```

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

```

(End of definition for __zrefclever_opt_bool_if:NTF.)

4.5 Reference format

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

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

somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the “key with no value” is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value “unset” for this purpose. And similarly for “choice” options.

However, “unsetting” options is only supported at the general and reference type levels, that is, at `\zcsetup`, at `\zcref`, and at `\zcRefTypeSetup`. For language-specific options – in the language files or at `\zcLanguageSetup` – there is no unsetting, an option which has been set can there only be changed to another value. This for two reasons. First, these are low precedence levels, so it is less meaningful to be able to unset these options. Second, these settings can only be done in the preamble (or the package itself). They are meant to be global. So, do it once, do it right, and if you need to locally change something along the document, use a higher precedence level.

`\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.

```

563 \tl_new:N \l__zrefclever_setup_type_tl
564 \tl_new:N \l__zrefclever_setup_language_tl
565 \tl_new:N \l__zrefclever_lang_decl_case_tl
566 \seq_new:N \l__zrefclever_lang_declension_seq
567 \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.

```

568 \seq_new:N \g__zrefclever_rf_opts_tl_not_type_specific_seq
569 \seq_gset_from_clist:Nn
570   \g__zrefclever_rf_opts_tl_not_type_specific_seq
571   {
572     tpairsep ,
573     tlistsep ,
574     tlastsep ,
575     notesep ,
576   }
577 \seq_new:N \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
578 \seq_gset_from_clist:Nn
579   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
580   {
581     namesep ,
582     pairsep ,
583     listsep ,
584     lastsep ,
585     rangesep ,
586     namefont ,
587     reffont ,

```

```

588 }
589 \seq_new:N \g__zrefclever_rf_opts_seq_refbounds_seq
590 \seq_gset_from_clist:Nn
591   \g__zrefclever_rf_opts_seq_refbounds_seq
592 {
593   refbounds-first ,
594   refbounds-first-sg ,
595   refbounds-first-pb ,
596   refbounds-first-rb ,
597   refbounds-mid ,
598   refbounds-mid-rb ,
599   refbounds-mid-re ,
600   refbounds-last ,
601   refbounds-last-pe ,
602   refbounds-last-re ,
603 }
604 \seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
605 \seq_gset_from_clist:Nn
606   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
607 {
608   cap ,
609   abbrev ,
610   rangetopair ,
611 }

```

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:`.

```

612 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
613 \seq_gset_from_clist:Nn
614   \g__zrefclever_rf_opts_tl_type_names_seq
615 {
616   Name-sg ,
617   name-sg ,
618   Name-pl ,
619   name-pl ,
620   Name-sg-ab ,
621   name-sg-ab ,
622   Name-pl-ab ,
623   name-pl-ab ,
624 }

```

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

```

625 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
626 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
627   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
628   \g__zrefclever_rf_opts_tl_type_names_seq
629 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
630 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
631   \g__zrefclever_rf_opts_tl_not_type_specific_seq
632   \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.

```

633 \clist_map_inline:nn
634 {
635   reference ,
636   typesetup ,
637   langsetup ,
638   langfile ,
639 }
640 {
641   \keys_define:nn { zref-clever/ #1 }
642   {
643     +refbounds-first .meta:n =
644     {
645       refbounds-first = {##1} ,
646       refbounds-first-sg = {##1} ,
647       refbounds-first-pb = {##1} ,
648       refbounds-first-rb = {##1} ,
649     } ,
650     +refbounds-mid .meta:n =
651     {
652       refbounds-mid = {##1} ,
653       refbounds-mid-rb = {##1} ,
654       refbounds-mid-re = {##1} ,
655     } ,
656     +refbounds-last .meta:n =
657     {
658       refbounds-last = {##1} ,
659       refbounds-last-pe = {##1} ,
660       refbounds-last-re = {##1} ,
661     } ,
662     +refbounds-rb .meta:n =
663     {
664       refbounds-first-rb = {##1} ,
665       refbounds-mid-rb = {##1} ,
666     } ,
667     +refbounds-re .meta:n =
668     {
669       refbounds-mid-re = {##1} ,
670       refbounds-last-re = {##1} ,
671     } ,
672     +refbounds .meta:n =
673     {
674       +refbounds-first = {##1} ,
675       +refbounds-mid = {##1} ,
676       +refbounds-last = {##1} ,
677     } ,
678     refbounds .meta:n = { +refbounds = {##1} } ,
679   }
680 }
681 \clist_map_inline:nn
682 {
683   reference ,
684   typesetup ,
685 }
686 {

```

```

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

```

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.

```

715 \tl_new:N \l__zrefclever_ref_language_tl
716 \tl_new:N \l__zrefclever_current_language_tl
717 \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`.

```

718 \tl_new:N \l_zrefclever_ref_language_tl
719 \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⟩}

```



```

720 \cs_new:Npn \__zrefclever_language_varname:n #1
721 { g__zrefclever_declared_language_ #1 _t1 }

```

(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.

```

722 \cs_set_eq:NN \zrefclever_language_varname:n
723 \__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>}

```

```

724 \prg_new_conditional:Npnn \__zrefclever_language_if_declared:n #1 { T , F , TF }
725 {
726   \t1_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
727   { \prg_return_true: }
728   { \prg_return_false: }
729 }
730 \prg_generate_conditional_variant:Nmn
731 \__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.

```

732 \prg_set_eq_conditional:Nmn \zrefclever_language_if_declared:n
733 \__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>}

```

```

734 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
735 {
736   \group_begin:
737   \tl_if_empty:nF {#2}
738   {
739     \__zrefclever_language_if_declared:nTF {#2}
740     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
741     {
742       \tl_new:c { \__zrefclever_language_varname:n {#2} }
743       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
744       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
745       \keys_set:nn { zref-clever/declarelang } {#1}
746     }
747   }
748   \group_end:
749 }
750 \@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 $\rangle$ }

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

```

(End of definition for \zcDeclareLanguageAlias.)

```

765 \keys_define:nn { zref-clever/declarelang }
766 {
767   declension .code:n =
768   {
769     \seq_new:c
770     {
771       \__zrefclever_opt_varname_language:enn
772       { \l__zrefclever_setup_language_tl } { declension } { seq }
773     }
774     \seq_gset_from_clist:cn
775     {
776       \__zrefclever_opt_varname_language:enn
777       { \l__zrefclever_setup_language_tl } { declension } { seq }

```

```

778     }
779     {#1}
780   } ,
781   declension .value_required:n = true ,
782   gender .code:n =
783   {
784     \seq_new:c
785     {
786       \__zrefclever_opt_varname_language:enn
787       { \l__zrefclever_setup_language_tl } { gender } { seq }
788     }
789     \seq_gset_from_clist:cn
790     {
791       \__zrefclever_opt_varname_language:enn
792       { \l__zrefclever_setup_language_tl } { gender } { seq }
793     }
794     {#1}
795   } ,
796   gender .value_required:n = true ,
797   allcaps .choices:nn =
798   { true , false }
799   {
800     \bool_new:c
801     {
802       \__zrefclever_opt_varname_language:enn
803       { \l__zrefclever_setup_language_tl } { allcaps } { bool }
804     }
805     \use:c { bool_gset_ \l_keys_choice_tl :c }
806     {
807       \__zrefclever_opt_varname_language:enn
808       { \l__zrefclever_setup_language_tl } { allcaps } { bool }
809     }
810   } ,
811   allcaps .default:n = true ,
812 }

```

`__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.

```

813 \cs_new_protected:Npn \__zrefclever_process_language_settings:
814 {
815   \__zrefclever_language_if_declared:xTF
816   { \l__zrefclever_ref_language_tl }
817   {

```

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.

```

818     \__zrefclever_opt_seq_get:cNF
819     {
820         \__zrefclever_opt_varname_language:enn
821         { \l__zrefclever_ref_language_tl } { declension } { seq }
822     }
823     \l__zrefclever_lang_declension_seq
824     { \seq_clear:N \l__zrefclever_lang_declension_seq }
825     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
826     {
827         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
828         {
829             \msg_warning:nxxx { zref-clever }
830             { language-no-decl-ref }
831             { \l__zrefclever_ref_language_tl }
832             { \l__zrefclever_ref_decl_case_tl }
833             \tl_clear:N \l__zrefclever_ref_decl_case_tl
834         }
835     }
836     {
837         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
838         {
839             \seq_get_left:NN \l__zrefclever_lang_declension_seq
840             \l__zrefclever_ref_decl_case_tl
841         }
842         {
843             \seq_if_in:NVF \l__zrefclever_lang_declension_seq
844             \l__zrefclever_ref_decl_case_tl
845             {
846                 \msg_warning:nxxx { zref-clever }
847                 { unknown-decl-case }
848                 { \l__zrefclever_ref_decl_case_tl }
849                 { \l__zrefclever_ref_language_tl }
850                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
851                 \l__zrefclever_ref_decl_case_tl
852             }
853         }
854     }

```

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.

```

855     \__zrefclever_opt_seq_get:cNF
856     {
857         \__zrefclever_opt_varname_language:enn
858         { \l__zrefclever_ref_language_tl } { gender } { seq }
859     }
860     \l__zrefclever_lang_gender_seq
861     { \seq_clear:N \l__zrefclever_lang_gender_seq }
862     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
863     {
864         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
865         {

```

```

866         \msg_warning:nxxxx { zref-clever }
867         { language-no-gender }
868         { \l__zrefclever_ref_language_tl }
869         { g }
870         { \l__zrefclever_ref_gender_tl }
871         \tl_clear:N \l__zrefclever_ref_gender_tl
872     }
873 }
874 {
875     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
876     {
877         \seq_if_in:NVF \l__zrefclever_lang_gender_seq
878         \l__zrefclever_ref_gender_tl
879         {
880             \msg_warning:nxxx { zref-clever }
881             { gender-not-declared }
882             { \l__zrefclever_ref_language_tl }
883             { \l__zrefclever_ref_gender_tl }
884             \tl_clear:N \l__zrefclever_ref_gender_tl
885         }
886     }
887 }

```

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

```

888     \__zrefclever_opt_bool_if:cT
889     {
890         \__zrefclever_opt_varname_language:enn
891         { \l__zrefclever_ref_language_tl } { allcaps } { bool }
892     }
893     { \keys_set:nn { zref-clever/reference } { cap = true } }
894 }
895 {

```

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

```

896     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
897     {
898         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
899         { \l__zrefclever_ref_decl_case_tl }
900         { \l__zrefclever_ref_language_tl }
901         \tl_clear:N \l__zrefclever_ref_decl_case_tl
902     }
903     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
904     {
905         \msg_warning:nxxxx { zref-clever }
906         { language-no-gender }
907         { \l__zrefclever_ref_language_tl }
908         { g }
909         { \l__zrefclever_ref_gender_tl }
910         \tl_clear:N \l__zrefclever_ref_gender_tl
911     }
912 }
913 }

```

(End of definition for `_zrefclever_process_language_settings:`)

4.7 Language files

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

The built-in language files and their related infrastructure are designed to perform “on the fly” loading of the language files, “lazily” as needed. Much like `babel` does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of parsimony, of “loading the least possible”. Therefore, we load at `begindocument` one single language (see [lang option](#)), as specified by the user in the preamble with the `lang` option or, failing any specification, the current language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

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

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

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

```
914 \seq_new:N \g_zrefclever_loaded_langfiles_seq
```

(End of definition for `\g_zrefclever_loaded_langfiles_seq`.)

`_zrefclever_provide_langfile:n` Load language file for known $\langle language \rangle$ if it is available and if it has not already been loaded.

```
\_zrefclever_provide_langfile:n {\language}
```

```

915 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
916 {
917   \group_begin:
918   \@bsphack
919   \__zrefclever_language_if_declared:nT {#1}
920   {
921     \seq_if_in:NxF
922     \g__zrefclever_loaded_langfiles_seq
923     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
924     {
925       \exp_args:Nx \file_get:nnNTF
926       {
927         zref-clever-
928         \tl_use:c { \__zrefclever_language_varname:n {#1} }
929         .lang
930       }
931       { \ExplSyntaxOn }
932       \l_tmpa_tl
933       {
934         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
935         \tl_clear:N \l__zrefclever_setup_type_tl
936         \__zrefclever_opt_seq_get:cNF
937         {
938           \__zrefclever_opt_varname_language:nnn
939           {#1} { declension } { seq }
940         }
941         \l__zrefclever_lang_declension_seq
942         { \seq_clear:N \l__zrefclever_lang_declension_seq }
943         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
944         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
945         {
946           \seq_get_left:NN \l__zrefclever_lang_declension_seq
947           \l__zrefclever_lang_decl_case_tl
948         }
949         \__zrefclever_opt_seq_get:cNF
950         {
951           \__zrefclever_opt_varname_language:nnn
952           {#1} { gender } { seq }
953         }
954         \l__zrefclever_lang_gender_seq
955         { \seq_clear:N \l__zrefclever_lang_gender_seq }
956         \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
957         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
958         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
959         \msg_info:nnx { zref-clever } { langfile-loaded }
960         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
961       }
962     }

```

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.

```

963     \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
964     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }

```

```

965     }
966   }
967 }
968 \@esphack
969 \group_end:
970 }
971 \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.

```

972 \keys_define:nn { zref-clever/langfile }
973 {
974   type .code:n =
975   {
976     \tl_if_empty:NTF {#1}
977     { \tl_clear:N \l__zrefclever_setup_type_tl }
978     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
979   } ,
980
981   case .code:n =
982   {
983     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
984     {
985       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
986       { \l__zrefclever_setup_language_tl } {#1}
987     }
988     {
989       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
990       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
991       {
992         \msg_info:nxxx { zref-clever } { unknown-decl-case }
993         {#1} { \l__zrefclever_setup_language_tl }
994         \seq_get_left:NN \l__zrefclever_lang_declension_seq
995         \l__zrefclever_lang_decl_case_tl
996       }
997     }
998   } ,
999   case .value_required:n = true ,
1000
1001   gender .value_required:n = true ,
1002   gender .code:n =
1003   {
1004     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1005     {
1006       \msg_info:nxxxx { zref-clever } { language-no-gender }
1007       { \l__zrefclever_setup_language_tl } { gender } {#1}
1008     }
1009     {
1010       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1011       {

```



```

1012         \msg_info:nnn { zref-clever }
1013         { option-only-type-specific } { gender }
1014     }
1015     {
1016     \seq_clear:N \l_tmpa_seq
1017     \clist_map_inline:nn {#1}
1018     {
1019         \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1020         { \seq_put_right:Nn \l_tmpa_seq {##1} }
1021         {
1022             \msg_info:nxxx { zref-clever }
1023             { gender-not-declared }
1024             { \l__zrefclever_setup_language_tl } {##1}
1025         }
1026     }
1027     \__zrefclever_opt_seq_if_set:cF
1028     {
1029         \__zrefclever_opt_varname_lang_type:eenn
1030         { \l__zrefclever_setup_language_tl }
1031         { \l__zrefclever_setup_type_tl }
1032         { gender }
1033         { seq }
1034     }
1035     {
1036         \seq_new:c
1037         {
1038             \__zrefclever_opt_varname_lang_type:eenn
1039             { \l__zrefclever_setup_language_tl }
1040             { \l__zrefclever_setup_type_tl }
1041             { gender }
1042             { seq }
1043         }
1044         \seq_gset_eq:cN
1045         {
1046             \__zrefclever_opt_varname_lang_type:eenn
1047             { \l__zrefclever_setup_language_tl }
1048             { \l__zrefclever_setup_type_tl }
1049             { gender }
1050             { seq }
1051         }
1052         \l_tmpa_seq
1053     }
1054     }
1055     } ,
1056 }
1057 }
1058 \seq_map_inline:Nn
1059 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1060 {
1061     \keys_define:nn { zref-clever/langfile }
1062     {
1063         #1 .value_required:n = true ,
1064         #1 .code:n =
1065         {

```

```

1066         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1067         {
1068             \__zrefclever_opt_tl_gset_if_new:cn
1069             {
1070                 \__zrefclever_opt_varname_lang_default:enn
1071                 { \l__zrefclever_setup_language_tl }
1072                 {#1} { t1 }
1073             }
1074             {##1}
1075         }
1076         {
1077             \msg_info:nnn { zref-clever }
1078             { option-not-type-specific } {#1}
1079         }
1080     } ,
1081 }
1082 }
1083 \seq_map_inline:Nn
1084 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1085 {
1086     \keys_define:nn { zref-clever/langfile }
1087     {
1088         #1 .value_required:n = true ,
1089         #1 .code:n =
1090         {
1091             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1092             {
1093                 \__zrefclever_opt_tl_gset_if_new:cn
1094                 {
1095                     \__zrefclever_opt_varname_lang_default:enn
1096                     { \l__zrefclever_setup_language_tl }
1097                     {#1} { t1 }
1098                 }
1099                 {##1}
1100             }
1101             {
1102                 \__zrefclever_opt_tl_gset_if_new:cn
1103                 {
1104                     \__zrefclever_opt_varname_lang_type:eenn
1105                     { \l__zrefclever_setup_language_tl }
1106                     { \l__zrefclever_setup_type_tl }
1107                     {#1} { t1 }
1108                 }
1109                 {##1}
1110             }
1111         } ,
1112     }
1113 }
1114 \keys_define:nn { zref-clever/langfile }
1115 {
1116     endrange .value_required:n = true ,
1117     endrange .code:n =
1118     {
1119         \str_case:nnF {#1}

```

```

1120 {
1121   { ref }
1122   {
1123     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1124     {
1125       \__zrefclever_opt_tl_gclear_if_new:c
1126       {
1127         \__zrefclever_opt_varname_lang_default:enn
1128         { \l__zrefclever_setup_language_tl }
1129         { endrangefunc } { tl }
1130       }
1131       \__zrefclever_opt_tl_gclear_if_new:c
1132       {
1133         \__zrefclever_opt_varname_lang_default:enn
1134         { \l__zrefclever_setup_language_tl }
1135         { endrangeprop } { tl }
1136       }
1137     }
1138     {
1139       \__zrefclever_opt_tl_gclear_if_new:c
1140       {
1141         \__zrefclever_opt_varname_lang_type:eenn
1142         { \l__zrefclever_setup_language_tl }
1143         { \l__zrefclever_setup_type_tl }
1144         { endrangefunc } { tl }
1145       }
1146       \__zrefclever_opt_tl_gclear_if_new:c
1147       {
1148         \__zrefclever_opt_varname_lang_type:eenn
1149         { \l__zrefclever_setup_language_tl }
1150         { \l__zrefclever_setup_type_tl }
1151         { endrangeprop } { tl }
1152       }
1153     }
1154   }
1155
1156 { stripprefix }
1157 {
1158   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1159   {
1160     \__zrefclever_opt_tl_gset_if_new:cn
1161     {
1162       \__zrefclever_opt_varname_lang_default:enn
1163       { \l__zrefclever_setup_language_tl }
1164       { endrangefunc } { tl }
1165     }
1166     { __zrefclever_get_endrange_stripprefix }
1167     \__zrefclever_opt_tl_gclear_if_new:c
1168     {
1169       \__zrefclever_opt_varname_lang_default:enn
1170       { \l__zrefclever_setup_language_tl }
1171       { endrangeprop } { tl }
1172     }
1173   }

```

```

1174     {
1175         \__zrefclever_opt_tl_gset_if_new:cn
1176         {
1177             \__zrefclever_opt_varname_lang_type:eenn
1178             { \l__zrefclever_setup_language_tl }
1179             { \l__zrefclever_setup_type_tl }
1180             { endrangefunc } { tl }
1181         }
1182         { __zrefclever_get_endrange_stripprefix }
1183         \__zrefclever_opt_tl_gclear_if_new:c
1184         {
1185             \__zrefclever_opt_varname_lang_type:eenn
1186             { \l__zrefclever_setup_language_tl }
1187             { \l__zrefclever_setup_type_tl }
1188             { endrangeprop } { tl }
1189         }
1190     }
1191 }
1192
1193 { pagecomp }
1194 {
1195     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1196     {
1197         \__zrefclever_opt_tl_gset_if_new:cn
1198         {
1199             \__zrefclever_opt_varname_lang_default:enn
1200             { \l__zrefclever_setup_language_tl }
1201             { endrangefunc } { tl }
1202         }
1203         { __zrefclever_get_endrange_pagecomp }
1204         \__zrefclever_opt_tl_gclear_if_new:c
1205         {
1206             \__zrefclever_opt_varname_lang_default:enn
1207             { \l__zrefclever_setup_language_tl }
1208             { endrangeprop } { tl }
1209         }
1210     }
1211     {
1212         \__zrefclever_opt_tl_gset_if_new:cn
1213         {
1214             \__zrefclever_opt_varname_lang_type:eenn
1215             { \l__zrefclever_setup_language_tl }
1216             { \l__zrefclever_setup_type_tl }
1217             { endrangefunc } { tl }
1218         }
1219         { __zrefclever_get_endrange_pagecomp }
1220         \__zrefclever_opt_tl_gclear_if_new:c
1221         {
1222             \__zrefclever_opt_varname_lang_type:eenn
1223             { \l__zrefclever_setup_language_tl }
1224             { \l__zrefclever_setup_type_tl }
1225             { endrangeprop } { tl }
1226         }
1227     }

```

```

1228     }
1229
1230 { pagecomp2 }
1231 {
1232   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1233   {
1234     \__zrefclever_opt_tl_gset_if_new:cn
1235     {
1236       \__zrefclever_opt_varname_lang_default:enn
1237       { \l__zrefclever_setup_language_tl }
1238       { endrangefunc } { t1 }
1239     }
1240     { __zrefclever_get_endrange_pagecomptwo }
1241     \__zrefclever_opt_tl_gclear_if_new:c
1242     {
1243       \__zrefclever_opt_varname_lang_default:enn
1244       { \l__zrefclever_setup_language_tl }
1245       { endrangeprop } { t1 }
1246     }
1247   }
1248   {
1249     \__zrefclever_opt_tl_gset_if_new:cn
1250     {
1251       \__zrefclever_opt_varname_lang_type:eenn
1252       { \l__zrefclever_setup_language_tl }
1253       { \l__zrefclever_setup_type_tl }
1254       { endrangefunc } { t1 }
1255     }
1256     { __zrefclever_get_endrange_pagecomptwo }
1257     \__zrefclever_opt_tl_gclear_if_new:c
1258     {
1259       \__zrefclever_opt_varname_lang_type:eenn
1260       { \l__zrefclever_setup_language_tl }
1261       { \l__zrefclever_setup_type_tl }
1262       { endrangeprop } { t1 }
1263     }
1264   }
1265 }
1266 }
1267 {
1268   \tl_if_empty:nTF {#1}
1269   {
1270     \msg_info:nnn { zref-clever }
1271     { endrange-property-undefined } {#1}
1272   }
1273   {
1274     \zref@ifpropundefined {#1}
1275     {
1276       \msg_info:nnn { zref-clever }
1277       { endrange-property-undefined } {#1}
1278     }
1279     {
1280       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1281       {

```

```

1282         \_zrefclever_opt_tl_gset_if_new:cn
1283         {
1284             \_zrefclever_opt_varname_lang_default:enn
1285             { \l__zrefclever_setup_language_tl }
1286             { endrangefunc } { tl }
1287         }
1288         { __zrefclever_get_endrange_property }
1289         \_zrefclever_opt_tl_gset_if_new:cn
1290         {
1291             \_zrefclever_opt_varname_lang_default:enn
1292             { \l__zrefclever_setup_language_tl }
1293             { endrangeprop } { tl }
1294         }
1295         {#1}
1296     }
1297 {
1298     \_zrefclever_opt_tl_gset_if_new:cn
1299     {
1300         \_zrefclever_opt_varname_lang_type:eenn
1301         { \l__zrefclever_setup_language_tl }
1302         { \l__zrefclever_setup_type_tl }
1303         { endrangefunc } { tl }
1304     }
1305     { __zrefclever_get_endrange_property }
1306     \_zrefclever_opt_tl_gset_if_new:cn
1307     {
1308         \_zrefclever_opt_varname_lang_type:eenn
1309         { \l__zrefclever_setup_language_tl }
1310         { \l__zrefclever_setup_type_tl }
1311         { endrangeprop } { tl }
1312     }
1313     {#1}
1314 }
1315 }
1316 }
1317 } ,
1318 } ,
1319 }
1320 \seq_map_inline:Nn
1321 \g__zrefclever_rf_opts_tl_type_names_seq
1322 {
1323     \keys_define:nn { zref-clever/langfile }
1324     {
1325         #1 .value_required:n = true ,
1326         #1 .code:n =
1327         {
1328             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1329             {
1330                 \msg_info:nnn { zref-clever }
1331                 { option-only-type-specific } {#1}
1332             }
1333             {
1334                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1335                 {

```

```

1336     \_zrefclever_opt_tl_gset_if_new:cn
1337     {
1338         \_zrefclever_opt_varname_lang_type:eenn
1339         { \l_zrefclever_setup_language_tl }
1340         { \l_zrefclever_setup_type_tl }
1341         {#1} { tl }
1342     }
1343     {##1}
1344 }
1345 {
1346     \_zrefclever_opt_tl_gset_if_new:cn
1347     {
1348         \_zrefclever_opt_varname_lang_type:eeen
1349         { \l_zrefclever_setup_language_tl }
1350         { \l_zrefclever_setup_type_tl }
1351         { \l_zrefclever_lang_decl_case_tl - #1 } { tl }
1352     }
1353     {##1}
1354 }
1355 }
1356 } ,
1357 }
1358 }
1359 \seq_map_inline:Nn
1360 \g_zrefclever_rf_opts_seq_refbounds_seq
1361 {
1362     \keys_define:nn { zref-clever/langfile }
1363     {
1364         #1 .value_required:n = true ,
1365         #1 .code:n =
1366         {
1367             \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1368             {
1369                 \_zrefclever_opt_seq_if_set:cF
1370                 {
1371                     \_zrefclever_opt_varname_lang_default:enn
1372                     { \l_zrefclever_setup_language_tl } {#1} { seq }
1373                 }
1374                 {
1375                     \seq_gclear:N \g_tmpa_seq
1376                     \_zrefclever_opt_seq_gset_clist_split:Nn
1377                     \g_tmpa_seq {##1}
1378                     \bool_lazy_or:nnTF
1379                     { \tl_if_empty_p:n {##1} }
1380                     {
1381                         \int_compare_p:nNn
1382                         { \seq_count:N \g_tmpa_seq } = { 4 }
1383                     }
1384                     {
1385                         \_zrefclever_opt_seq_gset_eq:cN
1386                         {
1387                             \_zrefclever_opt_varname_lang_default:enn
1388                             { \l_zrefclever_setup_language_tl }
1389                             {#1} { seq }

```

```

1390     }
1391     \g_tmpa_seq
1392   }
1393   {
1394     \msg_info:nxxx { zref-clever }
1395     { refbounds-must-be-four }
1396     {#1} { \seq_count:N \g_tmpa_seq }
1397   }
1398 }
1399 }
1400 {
1401   \__zrefclever_opt_seq_if_set:cF
1402   {
1403     \__zrefclever_opt_varname_lang_type:eenn
1404     { \l__zrefclever_setup_language_tl }
1405     { \l__zrefclever_setup_type_tl } {#1} { seq }
1406   }
1407   {
1408     \seq_gclear:N \g_tmpa_seq
1409     \__zrefclever_opt_seq_gset_clist_split:Nn
1410     \g_tmpa_seq {##1}
1411     \bool_lazy_or:nnTF
1412     { \tl_if_empty_p:n {##1} }
1413     {
1414       \int_compare_p:nNn
1415       { \seq_count:N \g_tmpa_seq } = { 4 }
1416     }
1417     {
1418       \__zrefclever_opt_seq_gset_eq:cN
1419       {
1420         \__zrefclever_opt_varname_lang_type:eenn
1421         { \l__zrefclever_setup_language_tl }
1422         { \l__zrefclever_setup_type_tl }
1423         {#1} { seq }
1424       }
1425       \g_tmpa_seq
1426     }
1427     {
1428       \msg_info:nxxx { zref-clever }
1429       { refbounds-must-be-four }
1430       {#1} { \seq_count:N \g_tmpa_seq }
1431     }
1432   }
1433 } ,
1434 }
1435 }
1436 }
1437 \seq_map_inline:Nn
1438 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1439 {
1440   \keys_define:nn { zref-clever/langfile }
1441   {
1442     #1 .choice: ,
1443     #1 / true .code:n =

```



```

1444 {
1445   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1446   {
1447     \__zrefclever_opt_bool_if_set:cF
1448     {
1449       \__zrefclever_opt_varname_lang_default:enn
1450       { \l__zrefclever_setup_language_tl }
1451       {#1} { bool }
1452     }
1453     {
1454       \__zrefclever_opt_bool_gset_true:c
1455       {
1456         \__zrefclever_opt_varname_lang_default:enn
1457         { \l__zrefclever_setup_language_tl }
1458         {#1} { bool }
1459       }
1460     }
1461   }
1462   {
1463     \__zrefclever_opt_bool_if_set:cF
1464     {
1465       \__zrefclever_opt_varname_lang_type:eenn
1466       { \l__zrefclever_setup_language_tl }
1467       { \l__zrefclever_setup_type_tl }
1468       {#1} { bool }
1469     }
1470     {
1471       \__zrefclever_opt_bool_gset_true:c
1472       {
1473         \__zrefclever_opt_varname_lang_type:eenn
1474         { \l__zrefclever_setup_language_tl }
1475         { \l__zrefclever_setup_type_tl }
1476         {#1} { bool }
1477       }
1478     }
1479   }
1480 } ,
1481 #l / false .code:n =
1482 {
1483   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1484   {
1485     \__zrefclever_opt_bool_if_set:cF
1486     {
1487       \__zrefclever_opt_varname_lang_default:enn
1488       { \l__zrefclever_setup_language_tl }
1489       {#1} { bool }
1490     }
1491     {
1492       \__zrefclever_opt_bool_gset_false:c
1493       {
1494         \__zrefclever_opt_varname_lang_default:enn
1495         { \l__zrefclever_setup_language_tl }
1496         {#1} { bool }
1497       }
1498     }
1499   }

```

```

1498     }
1499   }
1500   {
1501     \__zrefclever_opt_bool_if_set:cF
1502     {
1503       \__zrefclever_opt_varname_lang_type:eenn
1504       { \l__zrefclever_setup_language_tl }
1505       { \l__zrefclever_setup_type_tl }
1506       {#1} { bool }
1507     }
1508     {
1509       \__zrefclever_opt_bool_gset_false:c
1510       {
1511         \__zrefclever_opt_varname_lang_type:eenn
1512         { \l__zrefclever_setup_language_tl }
1513         { \l__zrefclever_setup_type_tl }
1514         {#1} { bool }
1515       }
1516     }
1517   }
1518 } ,
1519 #1 .default:n = true ,
1520 no #1 .meta:n = { #1 = false } ,
1521 no #1 .value_forbidden:n = true ,
1522 }
1523 }

```

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.

```

1524 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1525 {
1526   \tl_const:cn
1527   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1528 }
1529 \keyval_parse:nnn
1530 { }
1531 { \__zrefclever_opt_tl_cset_fallback:nn }
1532 {
1533   tpairsep = {,~} ,
1534   tlistsep = {,~} ,
1535   tlastsep = {,~} ,
1536   notesep = {~} ,
1537   namesep = {\nobreakspace} ,
1538   pairsep = {,~} ,
1539   listsep = {,~} ,
1540   lastsep = {,~} ,
1541   rangesep = {\textendash} ,
1542 }

```

4.8 Options

Auxiliary

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

```

    \_zrefclever_prop_put_non_empty:Nnn \langle property list \rangle \{ \langle key \rangle \} \{ \langle value \rangle \}
1543 \cs_new_protected:Npn \_zrefclever_prop_put_non_empty:Nnn #1#2#3
1544 {
1545   \tl_if_empty:nTF {#3}
1546     { \prop_remove:Nn #1 {#2} }
1547     { \prop_put:Nnn #1 {#2} {#3} }
1548 }
```

(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`.

```

1549 \tl_new:N \l_zrefclever_ref_property_tl
1550 \keys_define:nn { zref-clever/reference }
1551 {
1552   ref .code:n =
1553   {
1554     \tl_if_empty:nTF {#1}
1555     {
1556       \msg_warning:nnn { zref-clever }
1557       { zref-property-undefined } {#1}
1558       \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1559     }
1560     {
1561       \zref@ifpropundefined {#1}
1562       {
1563         \msg_warning:nnn { zref-clever }
1564         { zref-property-undefined } {#1}
1565         \tl_set:Nn \l_zrefclever_ref_property_tl { default }
1566       }
1567       { \tl_set:Nn \l_zrefclever_ref_property_tl {#1} }
1568     }
1569   } ,
1570   ref .initial:n = default ,
1571   ref .value_required:n = true ,
1572   page .meta:n = { ref = page } ,
1573   page .value_forbidden:n = true ,
1574 }
```

typeset option

```
1575 \bool_new:N \l__zrefclever_typeset_ref_bool
1576 \bool_new:N \l__zrefclever_typeset_name_bool
1577 \keys_define:nn { zref-clever/reference }
1578 {
1579   typeset .choice: ,
1580   typeset / both .code:n =
1581     {
1582       \bool_set_true:N \l__zrefclever_typeset_ref_bool
1583       \bool_set_true:N \l__zrefclever_typeset_name_bool
1584     } ,
1585   typeset / ref .code:n =
1586     {
1587       \bool_set_true:N \l__zrefclever_typeset_ref_bool
1588       \bool_set_false:N \l__zrefclever_typeset_name_bool
1589     } ,
1590   typeset / name .code:n =
1591     {
1592       \bool_set_false:N \l__zrefclever_typeset_ref_bool
1593       \bool_set_true:N \l__zrefclever_typeset_name_bool
1594     } ,
1595   typeset .initial:n = both ,
1596   typeset .value_required:n = true ,
1597
1598   noname .meta:n = { typeset = ref } ,
1599   noname .value_forbidden:n = true ,
1600   noref .meta:n = { typeset = name } ,
1601   noref .value_forbidden:n = true ,
1602 }
```

sort option

```
1603 \bool_new:N \l__zrefclever_typeset_sort_bool
1604 \keys_define:nn { zref-clever/reference }
1605 {
1606   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1607   sort .initial:n = true ,
1608   sort .default:n = true ,
1609   nosort .meta:n = { sort = false } ,
1610   nosort .value_forbidden:n = true ,
1611 }
```

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`.

```
1612 \seq_new:N \l__zrefclever_typesort_seq
1613 \keys_define:nn { zref-clever/reference }
1614 {
1615   typesort .code:n =
1616     {
1617       \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1618       \seq_reverse:N \l__zrefclever_typesort_seq

```

```

1619     } ,
1620     typesort .initial:n =
1621     { part , chapter , section , paragraph },
1622     typesort .value_required:n = true ,
1623     notypesort .code:n =
1624     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1625     notypesort .value_forbidden:n = true ,
1626 }

```

comp option

```

1627 \bool_new:N \l__zrefclever_typeset_compress_bool
1628 \keys_define:nn { zref-clever/reference }
1629 {
1630   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1631   comp .initial:n = true ,
1632   comp .default:n = true ,
1633   nocomp .meta:n = { comp = false },
1634   nocomp .value_forbidden:n = true ,
1635 }

```

endrange option

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

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

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

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

some commands which may interfere with the processing. This is a good idea, thus we offer a similar hook for the same purpose: `endrange-setup`.

```

1636 \NewHook { zref-clever/endrange-setup }
1637 \keys_define:nn { zref-clever/reference }
1638 {
1639   endrange .code:n =
1640   {
1641     \str_case:nnF {#1}
1642     {
1643       { ref }
1644       {
1645         \__zrefclever_opt_tl_clear:c
1646         {
1647           \__zrefclever_opt_varname_general:nn
1648           { endrangefunc } { tl }
1649         }
1650         \__zrefclever_opt_tl_clear:c
1651         {
1652           \__zrefclever_opt_varname_general:nn
1653           { endrangeprop } { tl }
1654         }
1655       }
1656     }
1657     { stripprefix }
1658     {
1659       \__zrefclever_opt_tl_set:cn
1660       {
1661         \__zrefclever_opt_varname_general:nn
1662         { endrangefunc } { tl }
1663       }
1664       { __zrefclever_get_endrange_stripprefix }
1665       \__zrefclever_opt_tl_clear:c
1666       {
1667         \__zrefclever_opt_varname_general:nn
1668         { endrangeprop } { tl }
1669       }
1670     }
1671     { pagecomp }
1672     {
1673       \__zrefclever_opt_tl_set:cn
1674       {
1675         \__zrefclever_opt_varname_general:nn
1676         { endrangefunc } { tl }
1677       }
1678       { __zrefclever_get_endrange_pagecomp }
1679       \__zrefclever_opt_tl_clear:c
1680       {
1681         \__zrefclever_opt_varname_general:nn
1682         { endrangeprop } { tl }
1683       }
1684     }
1685   }
1686

```

```

1687 { pagecomp2 }
1688 {
1689   \_zrefclever_opt_tl_set:cn
1690   {
1691     \_zrefclever_opt_varname_general:nn
1692     { endrangefunc } { t1 }
1693   }
1694   { \_zrefclever_get_endrange_pagecomptwo }
1695   \_zrefclever_opt_tl_clear:c
1696   {
1697     \_zrefclever_opt_varname_general:nn
1698     { endrangeprop } { t1 }
1699   }
1700 }
1701
1702 { unset }
1703 {
1704   \_zrefclever_opt_tl_unset:c
1705   {
1706     \_zrefclever_opt_varname_general:nn
1707     { endrangefunc } { t1 }
1708   }
1709   \_zrefclever_opt_tl_unset:c
1710   {
1711     \_zrefclever_opt_varname_general:nn
1712     { endrangeprop } { t1 }
1713   }
1714 }
1715 }
1716 {
1717   \tl_if_empty:nTF {#1}
1718   {
1719     \msg_warning:nnn { zref-clever }
1720     { endrange-property-undefined } {#1}
1721   }
1722   {
1723     \zref@ifpropundefined {#1}
1724     {
1725       \msg_warning:nnn { zref-clever }
1726       { endrange-property-undefined } {#1}
1727     }
1728     {
1729       \_zrefclever_opt_tl_set:cn
1730       {
1731         \_zrefclever_opt_varname_general:nn
1732         { endrangefunc } { t1 }
1733       }
1734       { \_zrefclever_get_endrange_property }
1735       \_zrefclever_opt_tl_set:cn
1736       {
1737         \_zrefclever_opt_varname_general:nn
1738         { endrangeprop } { t1 }
1739       }
1740       {#1}

```

```

1741     }
1742   }
1743 } ,
1744   ,
1745   \endrange .value_required:n = true ,
1746 }
1747 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1748 {
1749   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1750   {
1751     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1752     {
1753       \__zrefclever_extract_default:Nnvn #3
1754       {#2} { \l__zrefclever_ref_property_tl } { }
1755     }
1756     { \tl_set:Nn #3 { zc@missingproperty } }
1757   }
1758   {
1759     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1760     {

```

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.

```

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

```



```

1789         \exp_args:Nxx \tl_if_eq:nnT
1790         {
1791             \__zrefclever_extract_unexp:nnn
1792             {#1} { zc@counter } { }
1793         }
1794         {
1795             \__zrefclever_extract_unexp:nnn
1796             {#2} { zc@counter } { }
1797         }
1798         {
1799             \exp_args:Nxx \tl_if_eq:nnT
1800             {
1801                 \__zrefclever_extract_unexp:nnn
1802                 {#1} { zc@enclval } { }
1803             }
1804             {
1805                 \__zrefclever_extract_unexp:nnn
1806                 {#2} { zc@enclval } { }
1807             }
1808             { \bool_set_true:N \l_tmpa_bool }
1809         }
1810     }
1811 }
1812 \bool_if:NTF \l_tmpa_bool
1813 {
1814     \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1815     {#2} { l__zrefclever_endrangeprop_tl } { }
1816 }
1817 {
1818     \zref@ifrefcontainsprop
1819     {#2} { \l__zrefclever_ref_property_tl }
1820     {
1821         \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1822         {#2} { l__zrefclever_ref_property_tl } { }
1823     }
1824     { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1825 }
1826 \exp_args:NNNV
1827 \group_end:
1828 \tl_set:Nn #3 \l_tmpb_tl
1829 }
1830 {
1831     \__zrefclever_extract_default:Nnvn #3
1832     {#2} { l__zrefclever_endrangeprop_tl } { }
1833 }
1834 }
1835 {
1836     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1837     {
1838         \__zrefclever_extract_default:Nnvn #3
1839         {#2} { l__zrefclever_ref_property_tl } { }
1840     }
1841     { \tl_set:Nn #3 { zc@missingproperty } }
1842 }

```

```

1843     }
1844   }
1845 \cs_generate_variant:Nn \_zrefclever_get_endrange_property:nnN { VVN }

```

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

```

1846 \cs_new_protected:Npn \_zrefclever_get_endrange_stripprefix:nnN #1#2#3
1847 {
1848   \zref@ifrefcontainsprop {#2} { \l_zrefclever_ref_property_tl }
1849   {
1850     \group_begin:
1851     \UseHook { zref-clever/endrange-setup }
1852     \tl_set:Nx \l_tmpa_tl
1853       {
1854         \_zrefclever_extract:nnn
1855         {#1} { \l_zrefclever_ref_property_tl } { }
1856       }
1857     \tl_set:Nx \l_tmpb_tl
1858       {
1859         \_zrefclever_extract:nnn
1860         {#2} { \l_zrefclever_ref_property_tl } { }
1861       }
1862     \bool_set_false:N \l_tmpa_bool
1863     \bool_until_do:Nn \l_tmpa_bool
1864     {
1865       \exp_args:Nxx \tl_if_eq:nnTF
1866       { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1867       {
1868         \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1869         \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1870         \tl_if_empty:NT \l_tmpb_tl
1871         { \bool_set_true:N \l_tmpa_bool }
1872       }
1873       { \bool_set_true:N \l_tmpa_bool }
1874     }
1875     \exp_args:NNNV
1876     \group_end:
1877     \tl_set:Nn #3 \l_tmpb_tl
1878   }
1879   { \tl_set:Nn #3 { zc@missingproperty } }
1880 }
1881 \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>).

```

1882 \prg_new_protected_conditional:Npnn
1883 \_zrefclever_is_integer_rgx:n #1 { F , TF }
1884 {
1885   \regex_match:nnTF { \A\d+\Z } {#1}
1886   { \prg_return_true: }
1887   { \prg_return_false: }
1888 }
1889 \prg_generate_conditional_variant:Nnn
1890 \_zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End of definition for _zrefclever_is_integer_rgx:n.)

```
1891 \cs_new_protected:Npn \_zrefclever_get_endrange_pagecomp:nnN #1#2#3
1892 {
1893   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1894   {
1895     \group_begin:
1896     \UseHook { zref-clever/endrange-setup }
1897     \tl_set:Nx \l_tmpa_tl
1898     {
1899       \_zrefclever_extract:nnn
1900       {#1} { \l__zrefclever_ref_property_tl } { }
1901     }
1902     \tl_set:Nx \l_tmpb_tl
1903     {
1904       \_zrefclever_extract:nnn
1905       {#2} { \l__zrefclever_ref_property_tl } { }
1906     }
1907     \bool_set_false:N \l_tmpa_bool
1908     \_zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1909     {
1910       \_zrefclever_is_integer_rgx:VF \l_tmpb_tl
1911       { \bool_set_true:N \l_tmpa_bool }
1912     }
1913     { \bool_set_true:N \l_tmpa_bool }
1914     \bool_until_do:Nn \l_tmpa_bool
1915     {
1916       \exp_args:Nxx \tl_if_eq:nnTF
1917       { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1918       {
1919         \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1920         \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1921         \tl_if_empty:NT \l_tmpb_tl
1922         { \bool_set_true:N \l_tmpa_bool }
1923       }
1924       { \bool_set_true:N \l_tmpa_bool }
1925     }
1926     \exp_args:NNNV
1927     \group_end:
1928     \tl_set:Nn #3 \l_tmpb_tl
1929   }
1930   { \tl_set:Nn #3 { zc@missingproperty } }
1931 }
1932 \cs_generate_variant:Nn \_zrefclever_get_endrange_pagecomp:nnN { VVN }
1933 \cs_new_protected:Npn \_zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1934 {
1935   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1936   {
1937     \group_begin:
1938     \UseHook { zref-clever/endrange-setup }
1939     \tl_set:Nx \l_tmpa_tl
1940     {
1941       \_zrefclever_extract:nnn
1942       {#1} { \l__zrefclever_ref_property_tl } { }
```

```

1943     }
1944     \tl_set:Nx \l_tmpb_tl
1945     {
1946         \__zrefclever_extract:nnn
1947         {#2} { \l__zrefclever_ref_property_tl } { }
1948     }
1949     \bool_set_false:N \l_tmpa_bool
1950     \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1951     {
1952         \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1953         { \bool_set_true:N \l_tmpa_bool }
1954     }
1955     { \bool_set_true:N \l_tmpa_bool }
1956     \bool_until_do:Nn \l_tmpa_bool
1957     {
1958         \exp_args:Nxx \tl_if_eq:nnTF
1959         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1960         {
1961             \bool_lazy_or:nnTF
1962             { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1963             { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1964             {
1965                 \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1966                 \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1967             }
1968             { \bool_set_true:N \l_tmpa_bool }
1969         }
1970         { \bool_set_true:N \l_tmpa_bool }
1971     }
1972     \exp_args:NNNV
1973     \group_end:
1974     \tl_set:Nn #3 \l_tmpb_tl
1975 }
1976 { \tl_set:Nn #3 { zc@missingproperty } }
1977 }
1978 \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`.

```

1979 \bool_new:N \l__zrefclever_typeset_range_bool
1980 \keys_define:nn { zref-clever/reference }
1981 {
1982     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1983     range .initial:n = false ,
1984     range .default:n = true ,
1985 }

```

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`.

```

1986 \bool_new:N \l__zrefclever_capfirst_bool
1987 \keys_define:nn { zref-clever/reference }
1988 {
1989   capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1990   capfirst .initial:n = false ,
1991   capfirst .default:n = true ,
1992 }

```

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`.

```

1993 \bool_new:N \l__zrefclever_noabbrev_first_bool
1994 \keys_define:nn { zref-clever/reference }
1995 {
1996   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1997   noabbrevfirst .initial:n = false ,
1998   noabbrevfirst .default:n = true ,
1999 }

```

S option

```

2000 \keys_define:nn { zref-clever/reference }
2001 {
2002   S .meta:n =
2003     { capfirst = {#1} , noabbrevfirst = {#1} },
2004   S .default:n = true ,
2005 }

```

hyperref option

```

2006 \bool_new:N \l__zrefclever_hyperlink_bool
2007 \bool_new:N \l__zrefclever_hyperref_warn_bool
2008 \keys_define:nn { zref-clever/reference }
2009 {
2010   hyperref .choice: ,
2011   hyperref / auto .code:n =
2012     {
2013       \bool_set_true:N \l__zrefclever_hyperlink_bool
2014       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2015     } ,
2016   hyperref / true .code:n =
2017     {
2018       \bool_set_true:N \l__zrefclever_hyperlink_bool
2019       \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2020     } ,
2021   hyperref / false .code:n =
2022     {
2023       \bool_set_false:N \l__zrefclever_hyperlink_bool
2024       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2025     } ,
2026   hyperref .initial:n = auto ,
2027   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.

```

2028     nohyperref .meta:n = { hyperref = false } ,
2029     nohyperref .value_forbidden:n = true ,
2030   }
2031 \AddToHook { begindocument }
2032   {
2033     \__zrefclever_if_package_loaded:nTF { hyperref }
2034     {
2035       \bool_if:NT \l__zrefclever_hyperlink_bool
2036         { \RequirePackage { zref-hyperref } }
2037     }
2038     {
2039       \bool_if:NT \l__zrefclever_hyperref_warn_bool
2040         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2041       \bool_set_false:N \l__zrefclever_hyperlink_bool
2042     }
2043     \keys_define:nn { zref-clever/reference }
2044     {
2045       hyperref .code:n =
2046         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2047       nohyperref .code:n =
2048         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2049     }
2050   }

```

nameinlink option

```

2051 \str_new:N \l__zrefclever_nameinlink_str
2052 \keys_define:nn { zref-clever/reference }
2053   {
2054     nameinlink .choice: ,
2055     nameinlink / true .code:n =
2056       { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2057     nameinlink / false .code:n =
2058       { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2059     nameinlink / single .code:n =
2060       { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2061     nameinlink / tsingle .code:n =
2062       { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2063     nameinlink .initial:n = tsingle ,
2064     nameinlink .default:n = true ,
2065   }

```

preposinlink option (deprecated)

```

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

```

```

2072         { preposinlink } { refbounds }
2073     } ,
2074 }

```

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`.

```

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

```

```

2104         \tl_set:Nn \l__zrefclever_ref_language_tl
2105         { \l__zrefclever_current_language_tl }
2106     }
2107
2108     { main }
2109     {
2110         \tl_set:Nn \l__zrefclever_ref_language_tl
2111         { \l__zrefclever_main_language_tl }
2112     }
2113 }
2114 {
2115     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2116     \__zrefclever_language_if_declared:nF {#1}
2117     {
2118         \msg_warning:nnn { zref-clever }
2119         { unknown-language-opt } {#1}
2120     }
2121 }
2122 \__zrefclever_provide_langfile:x
2123 { \l__zrefclever_ref_language_tl }
2124 }
2125 } ,
2126 lang .initial:n = current ,
2127 lang .value_required:n = true ,
2128 }
2129 \AddToHook { begindocument / before }
2130 {
2131     \AddToHook { begindocument }
2132     {

```

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.

```

2133     \keys_define:nn { zref-clever/reference }
2134     {
2135         lang .code:n =
2136         {
2137             \str_case:nnF {#1}
2138             {
2139                 { current }
2140                 {
2141                     \tl_set:Nn \l__zrefclever_ref_language_tl
2142                     { \l__zrefclever_current_language_tl }
2143                 }
2144
2145                 { main }
2146                 {
2147                     \tl_set:Nn \l__zrefclever_ref_language_tl
2148                     { \l__zrefclever_main_language_tl }
2149                 }
2150             }
2151         }
2152         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2153         \__zrefclever_language_if_declared:nF {#1}

```



```

2154         {
2155             \msg_warning:nnn { zref-clever }
2156             { unknown-language-opt } {#1}
2157         }
2158     }
2159 },
2160 }
2161 }
2162 }

```

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.

```

2163 \tl_new:N \l__zrefclever_ref_decl_case_tl
2164 \keys_define:nn { zref-clever/reference }
2165 {
2166     d .code:n =
2167     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2168 }
2169 \AddToHook { begindocument }
2170 {
2171     \keys_define:nn { zref-clever/reference }
2172     {

```

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

```

2173         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2174         d .value_required:n = true ,
2175     }
2176 }

```

nudge & co. options

```

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

```

```

2192     \ifdraft
2193         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2194         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2195     } ,
2196 nudge / iffinal .code:n =
2197     {
2198     \ifoptionfinal
2199         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2200         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2201     } ,
2202 nudge .initial:n = false ,
2203 nudge .default:n = true ,
2204 nonudge .meta:n = { nudge = false } ,
2205 nonudge .value_forbidden:n = true ,
2206 nudgeif .code:n =
2207     {
2208     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2209     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2210     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2211     \clist_map_inline:nn {#1}
2212     {
2213     \str_case:nnF {##1}
2214     {
2215     { multitype }
2216     { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2217     { comptosing }
2218     { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2219     { gender }
2220     { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2221     { all }
2222     {
2223     \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2224     \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2225     \bool_set_true:N \l__zrefclever_nudge_gender_bool
2226     }
2227     }
2228     {
2229     \msg_warning:nnn { zref-clever }
2230     { nudgeif-unknown-value } {##1}
2231     }
2232     }
2233     } ,
2234 nudgeif .value_required:n = true ,
2235 nudgeif .initial:n = all ,
2236 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2237 sg .initial:n = false ,
2238 sg .default:n = true ,
2239 g .code:n =
2240     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2241 }
2242 \AddToHook { begindocument }
2243 {
2244     \keys_define:nn { zref-clever/reference }
2245     {

```

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

```

2246     g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2247     g .value_required:n = true ,
2248   }
2249 }

```

font option

```

2250 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2251 \keys_define:nn { zref-clever/reference }
2252 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2253 \keys_define:nn { zref-clever/reference }
2254 {
2255   titleref .code:n =
2256   {
2257     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2258     \msg_warning:nxxx { zref-clever }{ option-deprecated } { titleref }
2259     { \iow_char:N\usepackage\iow_char:N{zref-titleref\iow_char:N} }
2260   } ,
2261 }

```

vario option

```

2262 \keys_define:nn { zref-clever/reference }
2263 {
2264   vario .code:n =
2265   {
2266     % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2267     \msg_warning:nxxx { zref-clever }{ option-deprecated } { vario }
2268     { \iow_char:N\usepackage\iow_char:N{zref-vario\iow_char:N} }
2269   } ,
2270 }

```

note option

```

2271 \tl_new:N \l__zrefclever_zceref_note_tl
2272 \keys_define:nn { zref-clever/reference }
2273 {
2274   note .tl_set:N = \l__zrefclever_zceref_note_tl ,
2275   note .value_required:n = true ,
2276 }

```

check option

Integration with `zref-check`.

```

2277 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2278 \bool_new:N \l__zrefclever_zceref_with_check_bool
2279 \keys_define:nn { zref-clever/reference }
2280 {
2281   check .code:n =
2282   { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2283 }
2284 \AddToHook { begindocument }
2285 {

```

```

2286 \__zrefclever_if_package_loaded:nTF { zref-check }
2287 {
2288   \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2289   {
2290     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2291     \keys_define:nn { zref-clever/reference }
2292     {
2293       check .code:n =
2294       {
2295         \bool_set_true:N \l__zrefclever_zceref_with_check_bool
2296         \keys_set:nn { zref-check / zcheck } {#1}
2297       } ,
2298       check .value_required:n = true ,
2299     }
2300   }
2301   {
2302     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2303     \keys_define:nn { zref-clever/reference }
2304     {
2305       check .code:n =
2306       {
2307         \msg_warning:mn { zref-clever }
2308         { zref-check-too-old } { 2021-09-16-v0.2.1 }
2309       } ,
2310     }
2311   }
2312 }
2313 {
2314   \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2315   \keys_define:nn { zref-clever/reference }
2316   {
2317     check .code:n =
2318     { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2319   }
2320 }
2321 }

```

reftype option

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

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

```

2322 \tl_new:N \l__zrefclever_reftype_override_tl
2323 \keys_define:nn { zref-clever/label }
2324 {
2325   reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2326   reftype .default:n = {} ,
2327   reftype .initial:n = {} ,
2328 }

```

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`.

```
2329 \prop_new:N \l__zrefclever_counter_type_prop
2330 \keys_define:nn { zref-clever/label }
2331 {
2332   countertype .code:n =
2333   {
2334     \keyval_parse:nnn
2335     {
2336       \msg_warning:nnnn { zref-clever }
2337       { key-requires-value } { countertype }
2338     }
2339     {
2340       \__zrefclever_prop_put_non_empty:Nnn
2341       \l__zrefclever_counter_type_prop
2342     }
2343     {#1}
2344   } ,
2345   countertype .value_required:n = true ,
2346   countertype .initial:n =
2347   {
2348     subsection      = section ,
2349     subsubsection   = section ,
2350     subparagraph    = paragraph ,
2351     enumi           = item ,
2352     enumii          = item ,
2353     enumiii         = item ,
2354     enumiv          = item ,
2355     mpfootnote     = footnote ,
2356   } ,
2357 }
```

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.

```
2358 \seq_new:N \l__zrefclever_counter_resetters_seq
2359 \keys_define:nn { zref-clever/label }
2360 {
2361   counterresetters .code:n =
2362   {
2363     \clist_map_inline:nn {#1}
2364     {
2365       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2366       {
2367         \seq_put_right:Nn
2368         \l__zrefclever_counter_resetters_seq {##1}
2369       }
2370     }
2371   } ,
2372   counterresetters .initial:n =
2373   {
2374     part ,
2375     chapter ,
2376     section ,
2377     subsection ,
2378     subsubsection ,
2379     paragraph ,
2380     subparagraph ,
2381   },
2382   counterresetters .value_required:n = true ,
2383 }
```

counterresetby option

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

```
2384 \prop_new:N \l__zrefclever_counter_resetby_prop
2385 \keys_define:nn { zref-clever/label }
2386 {
2387   counterresetby .code:n =
2388   {
2389     \keyval_parse:nnn
2390     {
2391       \msg_warning:nnn { zref-clever }
2392       { key-requires-value } { counterresetby }
2393     }
2394 }
```

```

2394     {
2395         \__zrefclever_prop_put_non_empty:Nnn
2396         \l__zrefclever_counter_resetby_prop
2397     }
2398     {#1}
2399   } ,
2400   counterresetby .value_required:n = true ,
2401   counterresetby .initial:n =
2402   {

```

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.

```

2403     enumii = enumi ,
2404     enumiii = enumii ,
2405     enumiv = enumiii ,
2406   } ,
2407 }

```

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.

```

2408 \tl_new:N \l__zrefclever_current_counter_tl
2409 \keys_define:nn { zref-clever/label }
2410 {
2411   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2412   currentcounter .default:n = \@currentcounter ,
2413   currentcounter .initial:n = \@currentcounter ,
2414 }

```

labelhook option

```

2415 \bool_new:N \l__zrefclever_labelhook_bool
2416 \keys_define:nn { zref-clever/label }
2417 {
2418   labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2419   labelhook .initial:n = true ,
2420   labelhook .default:n = true ,
2421 }

```

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>.

```

2422 \AddToHookWithArguments { label }
2423 {
2424   \bool_if:NT \l__zrefclever_labelhook_bool
2425     { \zref@wrapper@babel \zref@label {#1} }
2426 }

```

nocompat option

```
2427 \bool_new:N \g__zrefclever_nocompat_bool
2428 \seq_new:N \g__zrefclever_nocompat_modules_seq
2429 \keys_define:nn { zref-clever/reference }
2430 {
2431   nocompat .code:n =
2432   {
2433     \tl_if_empty:nTF {#1}
2434     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2435     {
2436       \clist_map_inline:nn {#1}
2437       {
2438         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2439         {
2440           \seq_gput_right:Nn
2441             \g__zrefclever_nocompat_modules_seq {##1}
2442         }
2443       }
2444     }
2445   } ,
2446 }
2447 \AddToHook { begindocument }
2448 {
2449   \keys_define:nn { zref-clever/reference }
2450   {
2451     nocompat .code:n =
2452     {
2453       \msg_warning:nnn { zref-clever }
2454         { option-preamble-only } { nocompat }
2455     }
2456   }
2457 }
2458 \AtEndOfPackage
2459 {
2460   \AddToHook { begindocument }
2461   {
2462     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2463       { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2464   }
2465 }
```

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

```
\_zrefclever_compat_module:nn {<module>} {<code>}
```



```

2466 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2467 {
2468   \AddToHook { begindocument }
2469   {
2470     \bool_if:NF \g__zrefclever_nocompat_bool
2471     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2472     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2473   }
2474 }

```

(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.

```

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

```

```

2512         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2513         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2514         { preref } { rebounds }
2515     } ,
2516     postref .code:n =
2517     {
2518         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2519         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2520         { postref } { rebounds }
2521     } ,
2522 }
2523 \seq_map_inline:Nn
2524 \g__zrefclever_rf_opts_seq_rebounds_seq
2525 {
2526     \keys_define:nm { zref-clever/reference }
2527     {
2528         #1 .default:o = \c_novalue_tl ,
2529         #1 .code:n =
2530         {
2531             \tl_if_novalue:nTF {##1}
2532             {
2533                 \__zrefclever_opt_seq_unset:c
2534                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2535             }
2536             {
2537                 \seq_clear:N \l_tmpa_seq
2538                 \__zrefclever_opt_seq_set_clist_split:Nn
2539                 \l_tmpa_seq {##1}
2540                 \bool_lazy_or:nnTF
2541                 { \tl_if_empty_p:n {##1} }
2542                 { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2543                 {
2544                     \__zrefclever_opt_seq_set_eq:cN
2545                     { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2546                     \l_tmpa_seq
2547                 }
2548                 {
2549                     \msg_warning:nxxx { zref-clever }
2550                     { rebounds-must-be-four }
2551                     {#1} { \seq_count:N \l_tmpa_seq }
2552                 }
2553             }
2554         } ,
2555     }
2556 }
2557 \seq_map_inline:Nn
2558 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2559 {
2560     \keys_define:nm { zref-clever/reference }
2561     {
2562         #1 .choice: ,
2563         #1 / true .code:n =
2564         {
2565             \__zrefclever_opt_bool_set_true:c

```

```

2566         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2567     } ,
2568     #1 / false .code:n =
2569     {
2570         \_zrefclever_opt_bool_set_false:c
2571         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2572     } ,
2573     #1 / unset .code:n =
2574     {
2575         \_zrefclever_opt_bool_unset:c
2576         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2577     } ,
2578     #1 .default:n = true ,
2579     no #1 .meta:n = { #1 = false } ,
2580     no #1 .value_forbidden:n = true ,
2581 }
2582 }

```

Package options

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

```

2583 \keys_define:nn { }
2584 {
2585     zref-clever/zcsetup .inherit:n =
2586     {
2587         zref-clever/label ,
2588         zref-clever/reference ,
2589     }
2590 }

```

`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.

```

2591 \bool_lazy_and:nnT
2592 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2593 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2594 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

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

```

2595 \NewDocumentCommand \zcsetup { m }
2596 { \_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>}

2597 \cs_new_protected:Npn \_zrefclever_zcsetup:n #1
2598 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2599 \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>}

2600 \NewDocumentCommand \zcRefTypeSetup { m m }
2601 {
2602   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2603   \keys_set:nn { zref-clever/typesetup } {#2}
2604   \tl_clear:N \l__zrefclever_setup_type_tl
2605 }

(End of definition for \zcRefTypeSetup.)

2606 \seq_map_inline:Nn
2607 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2608 {
2609   \keys_define:nn { zref-clever/typesetup }
2610   {
2611     #1 .code:n =
2612     {
2613       \msg_warning:nnn { zref-clever }
2614       { option-not-type-specific } {#1}
2615     } ,
2616   }
2617 }
2618 \seq_map_inline:Nn
2619 \g__zrefclever_rf_opts_tl_typesetup_seq
2620 {
2621   \keys_define:nn { zref-clever/typesetup }
2622   {
2623     #1 .default:o = \c_novalue_tl ,
2624     #1 .code:n =
2625     {
2626       \tl_if_novalue:nTF {##1}

```

```

2627     {
2628         \__zrefclever_opt_t1_unset:c
2629         {
2630             \__zrefclever_opt_varname_type:enn
2631             { \l__zrefclever_setup_type_t1 } {#1} { t1 }
2632         }
2633     }
2634     {
2635         \__zrefclever_opt_t1_set:cn
2636         {
2637             \__zrefclever_opt_varname_type:enn
2638             { \l__zrefclever_setup_type_t1 } {#1} { t1 }
2639         }
2640         {##1}
2641     }
2642     } ,
2643 }
2644 }
2645 \keys_define:nn { zref-clever/typesetup }
2646 {
2647     endrange .code:n =
2648     {
2649         \str_case:nnF {#1}
2650         {
2651             { ref }
2652             {
2653                 \__zrefclever_opt_t1_clear:c
2654                 {
2655                     \__zrefclever_opt_varname_type:enn
2656                     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2657                 }
2658                 \__zrefclever_opt_t1_clear:c
2659                 {
2660                     \__zrefclever_opt_varname_type:enn
2661                     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2662                 }
2663             }
2664         }
2665         { stripprefix }
2666         {
2667             \__zrefclever_opt_t1_set:cn
2668             {
2669                 \__zrefclever_opt_varname_type:enn
2670                 { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2671             }
2672             { __zrefclever_get_endrange_stripprefix }
2673             \__zrefclever_opt_t1_clear:c
2674             {
2675                 \__zrefclever_opt_varname_type:enn
2676                 { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2677             }
2678         }
2679     }
2680     { pagecomp }

```

```

2681 {
2682   \__zrefclever_opt_t1_set:cn
2683   {
2684     \__zrefclever_opt_varname_type:enn
2685     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2686   }
2687   { __zrefclever_get_endrange_pagecomp }
2688   \__zrefclever_opt_t1_clear:c
2689   {
2690     \__zrefclever_opt_varname_type:enn
2691     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2692   }
2693 }
2694
2695 { pagecomp2 }
2696 {
2697   \__zrefclever_opt_t1_set:cn
2698   {
2699     \__zrefclever_opt_varname_type:enn
2700     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2701   }
2702   { __zrefclever_get_endrange_pagecomptwo }
2703   \__zrefclever_opt_t1_clear:c
2704   {
2705     \__zrefclever_opt_varname_type:enn
2706     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2707   }
2708 }
2709
2710 { unset }
2711 {
2712   \__zrefclever_opt_t1_unset:c
2713   {
2714     \__zrefclever_opt_varname_type:enn
2715     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2716   }
2717   \__zrefclever_opt_t1_unset:c
2718   {
2719     \__zrefclever_opt_varname_type:enn
2720     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2721   }
2722 }
2723 }
2724 {
2725   \tl_if_empty:nTF {#1}
2726   {
2727     \msg_warning:nnn { zref-clever }
2728     { endrange-property-undefined } {#1}
2729   }
2730   {
2731     \zref@ifpropundefined {#1}
2732     {
2733       \msg_warning:nnn { zref-clever }
2734       { endrange-property-undefined } {#1}

```

```

2735     }
2736     {
2737         \__zrefclever_opt_tl_set:cn
2738         {
2739             \__zrefclever_opt_varname_type:enn
2740             { \l__zrefclever_setup_type_tl }
2741             { endrangefunc } { tl }
2742         }
2743         { __zrefclever_get_endrange_property }
2744         \__zrefclever_opt_tl_set:cn
2745         {
2746             \__zrefclever_opt_varname_type:enn
2747             { \l__zrefclever_setup_type_tl }
2748             { endrangeprop } { tl }
2749         }
2750         {#1}
2751     }
2752 }
2753 }
2754 },
2755 endrange .value_required:n = true ,
2756 }
2757 \keys_define:nn { zref-clever/typesetup }
2758 {
2759     refpre .code:n =
2760     {
2761         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2762         \msg_warning:nnon { zref-clever }{ option-deprecated }
2763         { refpre } { refbounds }
2764     } ,
2765     refpos .code:n =
2766     {
2767         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2768         \msg_warning:nnon { zref-clever }{ option-deprecated }
2769         { refpos } { refbounds }
2770     } ,
2771     preref .code:n =
2772     {
2773         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2774         \msg_warning:nnon { zref-clever }{ option-deprecated }
2775         { preref } { refbounds }
2776     } ,
2777     postref .code:n =
2778     {
2779         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2780         \msg_warning:nnon { zref-clever }{ option-deprecated }
2781         { postref } { refbounds }
2782     } ,
2783 }
2784 \seq_map_inline:Nn
2785 \g__zrefclever_rf_opts_seq_refbounds_seq
2786 {
2787     \keys_define:nn { zref-clever/typesetup }
2788     {

```

```

2789     #1 .default:o = \c_novalue_tl ,
2790     #1 .code:n =
2791     {
2792         \tl_if_novalue:nTF {##1}
2793         {
2794             \__zrefclever_opt_seq_unset:c
2795             {
2796                 \__zrefclever_opt_varname_type:enn
2797                 { \l__zrefclever_setup_type_tl } {#1} { seq }
2798             }
2799         }
2800         {
2801             \seq_clear:N \l_tmpa_seq
2802             \__zrefclever_opt_seq_set_clist_split:Nn
2803             \l_tmpa_seq {##1}
2804             \bool_lazy_or:nnTF
2805             { \tl_if_empty_p:n {##1} }
2806             { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2807             {
2808                 \__zrefclever_opt_seq_set_eq:cN
2809                 {
2810                     \__zrefclever_opt_varname_type:enn
2811                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2812                 }
2813                 \l_tmpa_seq
2814             }
2815             {
2816                 \msg_warning:nxxx { zref-clever }
2817                 { refbounds-must-be-four }
2818                 {#1} { \seq_count:N \l_tmpa_seq }
2819             }
2820         }
2821     } ,
2822 }
2823 }
2824 \seq_map_inline:Nn
2825 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2826 {
2827     \keys_define:nn { zref-clever/typesetup }
2828     {
2829         #1 .choice: ,
2830         #1 / true .code:n =
2831         {
2832             \__zrefclever_opt_bool_set_true:c
2833             {
2834                 \__zrefclever_opt_varname_type:enn
2835                 { \l__zrefclever_setup_type_tl }
2836                 {#1} { bool }
2837             }
2838         } ,
2839         #1 / false .code:n =
2840         {
2841             \__zrefclever_opt_bool_set_false:c
2842             {

```



```

2843         \zrefclever_opt_varname_type:enn
2844         { \l__zrefclever_setup_type_tl }
2845         {#1} { bool }
2846     }
2847 } ,
2848 #1 / unset .code:n =
2849 {
2850     \zrefclever_opt_bool_unset:c
2851     {
2852         \zrefclever_opt_varname_type:enn
2853         { \l__zrefclever_setup_type_tl }
2854         {#1} { bool }
2855     }
2856 } ,
2857 #1 .default:n = true ,
2858 no #1 .meta:n = { #1 = false } ,
2859 no #1 .value_forbidden:n = true ,
2860 }
2861 }

```

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>}
2862 \NewDocumentCommand \zcLanguageSetup { m m }
2863 {
2864     \group_begin:
2865     \zrefclever_language_if_declared:nTF {#1}
2866     {
2867         \tl_clear:N \l__zrefclever_setup_type_tl
2868         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2869         \zrefclever_opt_seq_get:cNF
2870         {
2871             \zrefclever_opt_varname_language:nnn
2872             {#1} { declension } { seq }
2873         }
2874         \l__zrefclever_lang_declension_seq
2875         { \seq_clear:N \l__zrefclever_lang_declension_seq }
2876         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2877         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2878         {
2879             \seq_get_left:NN \l__zrefclever_lang_declension_seq
2880             \l__zrefclever_lang_decl_case_tl
2881         }
2882         \zrefclever_opt_seq_get:cNF
2883         {

```

```

2884         \_zrefclever_opt_varname_language:nnn
2885         {#1} { gender } { seq }
2886     }
2887     \l_zrefclever_lang_gender_seq
2888     { \seq_clear:N \l_zrefclever_lang_gender_seq }
2889     \keys_set:nn { zref-clever/langsetup } {#2}
2890 }
2891 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2892 \group_end:
2893 }
2894 \@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.

```

2895 \keys_define:nn { zref-clever/langsetup }
2896 {
2897     type .code:n =
2898     {
2899         \tl_if_empty:NTF {#1}
2900         { \tl_clear:N \l_zrefclever_setup_type_tl }
2901         { \tl_set:Nn \l_zrefclever_setup_type_tl {#1} }
2902     } ,
2903
2904     case .code:n =
2905     {
2906         \seq_if_empty:NTF \l_zrefclever_lang_declension_seq
2907         {
2908             \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2909             { \l_zrefclever_setup_language_tl } {#1}
2910         }
2911         {
2912             \seq_if_in:NnTF \l_zrefclever_lang_declension_seq {#1}
2913             { \tl_set:Nn \l_zrefclever_lang_decl_case_tl {#1} }
2914             {
2915                 \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2916                 {#1} { \l_zrefclever_setup_language_tl }
2917                 \seq_get_left:NN \l_zrefclever_lang_declension_seq
2918                 \l_zrefclever_lang_decl_case_tl
2919             }
2920         }
2921     } ,
2922     case .value_required:n = true ,
2923
2924     gender .value_required:n = true ,
2925     gender .code:n =
2926     {
2927         \seq_if_empty:NTF \l_zrefclever_lang_gender_seq
2928         {
2929             \msg_warning:nxxx { zref-clever } { language-no-gender }
2930             { \l_zrefclever_setup_language_tl } { gender } {#1}
2931         }
2932         {
2933             \tl_if_empty:NTF \l_zrefclever_setup_type_tl

```

```

2934     {
2935       \msg_warning:nnn { zref-clever }
2936       { option-only-type-specific } { gender }
2937     }
2938     {
2939       \seq_clear:N \l_tmpa_seq
2940       \clist_map_inline:nn {#1}
2941       {
2942         \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2943         { \seq_put_right:Nn \l_tmpa_seq {##1} }
2944         {
2945           \msg_warning:nxxx { zref-clever }
2946           { gender-not-declared }
2947           { \l__zrefclever_setup_language_tl } {##1}
2948         }
2949       }
2950       \__zrefclever_opt_seq_gset_eq:cN
2951       {
2952         \__zrefclever_opt_varname_lang_type:eenn
2953         { \l__zrefclever_setup_language_tl }
2954         { \l__zrefclever_setup_type_tl }
2955         { gender }
2956         { seq }
2957       }
2958       \l_tmpa_seq
2959     }
2960   } ,
2961 } ,
2962 }
2963 \seq_map_inline:Nn
2964 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2965 {
2966   \keys_define:nn { zref-clever/langsetup }
2967   {
2968     #1 .value_required:n = true ,
2969     #1 .code:n =
2970     {
2971       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2972       {
2973         \__zrefclever_opt_tl_gset:cn
2974         {
2975           \__zrefclever_opt_varname_lang_default:enn
2976           { \l__zrefclever_setup_language_tl } {#1} { tl }
2977         }
2978         {##1}
2979       }
2980     }
2981     \msg_warning:nnn { zref-clever }
2982     { option-not-type-specific } {#1}
2983   }
2984 } ,
2985 }
2986 }
2987 \seq_map_inline:Nn

```

```

2988 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2989 {
2990   \keys_define:nn { zref-clever/langsetup }
2991   {
2992     #1 .value_required:n = true ,
2993     #1 .code:n =
2994     {
2995       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2996       {
2997         \__zrefclever_opt_tl_gset:cn
2998         {
2999           \__zrefclever_opt_varname_lang_default:enn
3000           { \l__zrefclever_setup_language_tl } {#1} { t1 }
3001         }
3002         {##1}
3003       }
3004       {
3005         \__zrefclever_opt_tl_gset:cn
3006         {
3007           \__zrefclever_opt_varname_lang_type:eenn
3008           { \l__zrefclever_setup_language_tl }
3009           { \l__zrefclever_setup_type_tl }
3010           {#1} { t1 }
3011         }
3012         {##1}
3013       }
3014     } ,
3015   }
3016 }
3017 \keys_define:nn { zref-clever/langsetup }
3018 {
3019   endrange .value_required:n = true ,
3020   endrange .code:n =
3021   {
3022     \str_case:nnF {#1}
3023     {
3024       { ref }
3025       {
3026         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3027         {
3028           \__zrefclever_opt_tl_gclear:c
3029           {
3030             \__zrefclever_opt_varname_lang_default:enn
3031             { \l__zrefclever_setup_language_tl }
3032             { endrangefunc } { t1 }
3033           }
3034           \__zrefclever_opt_tl_gclear:c
3035           {
3036             \__zrefclever_opt_varname_lang_default:enn
3037             { \l__zrefclever_setup_language_tl }
3038             { endrangeprop } { t1 }
3039           }
3040         }
3041       }

```

```

3042     \_zrefclever_opt_tl_gclear:c
3043     {
3044         \_zrefclever_opt_varname_lang_type:eenn
3045         { \l_zrefclever_setup_language_tl }
3046         { \l_zrefclever_setup_type_tl }
3047         { endrangefunc } { tl }
3048     }
3049     \_zrefclever_opt_tl_gclear:c
3050     {
3051         \_zrefclever_opt_varname_lang_type:eenn
3052         { \l_zrefclever_setup_language_tl }
3053         { \l_zrefclever_setup_type_tl }
3054         { endrangeprop } { tl }
3055     }
3056 }
3057 }
3058
3059 { stripprefix }
3060 {
3061     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3062     {
3063         \_zrefclever_opt_tl_gset:cn
3064         {
3065             \_zrefclever_opt_varname_lang_default:enn
3066             { \l_zrefclever_setup_language_tl }
3067             { endrangefunc } { tl }
3068         }
3069         { __zrefclever_get_endrange_stripprefix }
3070         \_zrefclever_opt_tl_gclear:c
3071         {
3072             \_zrefclever_opt_varname_lang_default:enn
3073             { \l_zrefclever_setup_language_tl }
3074             { endrangeprop } { tl }
3075         }
3076     }
3077     {
3078         \_zrefclever_opt_tl_gset:cn
3079         {
3080             \_zrefclever_opt_varname_lang_type:eenn
3081             { \l_zrefclever_setup_language_tl }
3082             { \l_zrefclever_setup_type_tl }
3083             { endrangefunc } { tl }
3084         }
3085         { __zrefclever_get_endrange_stripprefix }
3086         \_zrefclever_opt_tl_gclear:c
3087         {
3088             \_zrefclever_opt_varname_lang_type:eenn
3089             { \l_zrefclever_setup_language_tl }
3090             { \l_zrefclever_setup_type_tl }
3091             { endrangeprop } { tl }
3092         }
3093     }
3094 }
3095

```

```

3096 { pagecomp }
3097 {
3098   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3099   {
3100     \__zrefclever_opt_tl_gset:cn
3101     {
3102       \__zrefclever_opt_varname_lang_default:enn
3103       { \l__zrefclever_setup_language_tl }
3104       { endrangefunc } { tl }
3105     }
3106     { __zrefclever_get_endrange_pagecomp }
3107     \__zrefclever_opt_tl_gclear:c
3108     {
3109       \__zrefclever_opt_varname_lang_default:enn
3110       { \l__zrefclever_setup_language_tl }
3111       { endrangeprop } { tl }
3112     }
3113   }
3114   {
3115     \__zrefclever_opt_tl_gset:cn
3116     {
3117       \__zrefclever_opt_varname_lang_type:eenn
3118       { \l__zrefclever_setup_language_tl }
3119       { \l__zrefclever_setup_type_tl }
3120       { endrangefunc } { tl }
3121     }
3122     { __zrefclever_get_endrange_pagecomp }
3123     \__zrefclever_opt_tl_gclear:c
3124     {
3125       \__zrefclever_opt_varname_lang_type:eenn
3126       { \l__zrefclever_setup_language_tl }
3127       { \l__zrefclever_setup_type_tl }
3128       { endrangeprop } { tl }
3129     }
3130   }
3131 }
3132
3133 { pagecomp2 }
3134 {
3135   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3136   {
3137     \__zrefclever_opt_tl_gset:cn
3138     {
3139       \__zrefclever_opt_varname_lang_default:enn
3140       { \l__zrefclever_setup_language_tl }
3141       { endrangefunc } { tl }
3142     }
3143     { __zrefclever_get_endrange_pagecomptwo }
3144     \__zrefclever_opt_tl_gclear:c
3145     {
3146       \__zrefclever_opt_varname_lang_default:enn
3147       { \l__zrefclever_setup_language_tl }
3148       { endrangeprop } { tl }
3149     }

```

```

3150     }
3151     {
3152         \zrefclever_opt_tl_gset:cn
3153         {
3154             \zrefclever_opt_varname_lang_type:eenn
3155             { \l__zrefclever_setup_language_tl }
3156             { \l__zrefclever_setup_type_tl }
3157             { endrangefunc } { tl }
3158         }
3159         { __zrefclever_get_endrange_pagecomptwo }
3160         \zrefclever_opt_tl_gclear:c
3161         {
3162             \zrefclever_opt_varname_lang_type:eenn
3163             { \l__zrefclever_setup_language_tl }
3164             { \l__zrefclever_setup_type_tl }
3165             { endrangeprop } { tl }
3166         }
3167     }
3168 }
3169 }
3170 {
3171     \tl_if_empty:nTF {#1}
3172     {
3173         \msg_warning:nnn { zref-clever }
3174         { endrange-property-undefined } {#1}
3175     }
3176     {
3177         \zref@ifpropundefined {#1}
3178         {
3179             \msg_warning:nnn { zref-clever }
3180             { endrange-property-undefined } {#1}
3181         }
3182         {
3183             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3184             {
3185                 \zrefclever_opt_tl_gset:cn
3186                 {
3187                     \zrefclever_opt_varname_lang_default:enn
3188                     { \l__zrefclever_setup_language_tl }
3189                     { endrangefunc } { tl }
3190                 }
3191                 { __zrefclever_get_endrange_property }
3192                 \zrefclever_opt_tl_gset:cn
3193                 {
3194                     \zrefclever_opt_varname_lang_default:enn
3195                     { \l__zrefclever_setup_language_tl }
3196                     { endrangeprop } { tl }
3197                 }
3198                 {#1}
3199             }
3200         }
3201         \zrefclever_opt_tl_gset:cn
3202         {
3203             \zrefclever_opt_varname_lang_type:eenn

```

```

3204         { \l__zrefclever_setup_language_tl }
3205         { \l__zrefclever_setup_type_tl }
3206         { endrangefunc } { tl }
3207     }
3208     { __zrefclever_get_endrange_property }
3209     \__zrefclever_opt_tl_gset:cn
3210     {
3211         \__zrefclever_opt_varname_lang_type:eenn
3212         { \l__zrefclever_setup_language_tl }
3213         { \l__zrefclever_setup_type_tl }
3214         { endrangeprop } { tl }
3215     }
3216     {#1}
3217 }
3218 }
3219 }
3220 }
3221 } ,
3222 }
3223 \keys_define:nn { zref-clever/langsetup }
3224 {
3225     refpre .code:n =
3226     {
3227         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3228         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3229         { refpre } { refbounds }
3230     } ,
3231     refpos .code:n =
3232     {
3233         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3234         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3235         { refpos } { refbounds }
3236     } ,
3237     preref .code:n =
3238     {
3239         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3240         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3241         { preref } { refbounds }
3242     } ,
3243     postref .code:n =
3244     {
3245         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3246         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3247         { postref } { refbounds }
3248     } ,
3249 }
3250 \seq_map_inline:Nn
3251   \g__zrefclever_rf_opts_tl_type_names_seq
3252   {
3253     \keys_define:nn { zref-clever/langsetup }
3254     {
3255         #1 .value_required:n = true ,
3256         #1 .code:n =
3257         {

```



```

3258 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3259 {
3260   \msg_warning:nnn { zref-clever }
3261   { option-only-type-specific } {#1}
3262 }
3263 {
3264   \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3265   {
3266     \__zrefclever_opt_tl_gset:cn
3267     {
3268       \__zrefclever_opt_varname_lang_type:een
3269       { \l__zrefclever_setup_language_tl }
3270       { \l__zrefclever_setup_type_tl }
3271       {#1} { tl }
3272     }
3273     {##1}
3274   }
3275   {
3276     \__zrefclever_opt_tl_gset:cn
3277     {
3278       \__zrefclever_opt_varname_lang_type:een
3279       { \l__zrefclever_setup_language_tl }
3280       { \l__zrefclever_setup_type_tl }
3281       { \l__zrefclever_lang_decl_case_tl - #1 }
3282       { tl }
3283     }
3284     {##1}
3285   }
3286 } ,
3287 }
3288 }
3289 }
3290 \seq_map_inline:Nn
3291 \g__zrefclever_rf_opts_seq_refbounds_seq
3292 {
3293   \keys_define:nn { zref-clever/langsetup }
3294   {
3295     #1 .value_required:n = true ,
3296     #1 .code:n =
3297     {
3298       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3299       {
3300         \seq_gclear:N \g_tmpa_seq
3301         \__zrefclever_opt_seq_gset_clist_split:Nn
3302         \g_tmpa_seq {##1}
3303         \bool_lazy_or:nnTF
3304         { \tl_if_empty_p:n {##1} }
3305         {
3306           \int_compare_p:nNn
3307           { \seq_count:N \g_tmpa_seq } = { 4 }
3308         }
3309         {
3310           \__zrefclever_opt_seq_gset_eq:cN
3311           {

```

```

3312         \_zrefclever_opt_varname_lang_default:enn
3313         { \l__zrefclever_setup_language_tl }
3314         {#1} { seq }
3315     }
3316     \g_tmpa_seq
3317 }
3318 {
3319     \msg_warning:nxxx { zref-clever }
3320     { rebounds-must-be-four }
3321     {#1} { \seq_count:N \g_tmpa_seq }
3322 }
3323 }
3324 {
3325     \seq_gclear:N \g_tmpa_seq
3326     \_zrefclever_opt_seq_gset_clist_split:Nn
3327     \g_tmpa_seq {##1}
3328     \bool_lazy_or:nnTF
3329     { \tl_if_empty_p:n {##1} }
3330     {
3331         \int_compare_p:nNn
3332         { \seq_count:N \g_tmpa_seq } = { 4 }
3333     }
3334     {
3335         \_zrefclever_opt_seq_gset_eq:cN
3336         {
3337             \_zrefclever_opt_varname_lang_type:enn
3338             { \l__zrefclever_setup_language_tl }
3339             { \l__zrefclever_setup_type_tl } {#1} { seq }
3340         }
3341         \g_tmpa_seq
3342     }
3343     {
3344         \msg_warning:nxxx { zref-clever }
3345         { rebounds-must-be-four }
3346         {#1} { \seq_count:N \g_tmpa_seq }
3347     }
3348 }
3349 } ,
3350 }
3351 }
3352 \seq_map_inline:Nn
3353 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3354 {
3355     \keys_define:nn { zref-clever/langsetup }
3356     {
3357         #1 .choice: ,
3358         #1 / true .code:n =
3359         {
3360             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3361             {
3362                 \_zrefclever_opt_bool_gset_true:c
3363                 {
3364                     \_zrefclever_opt_varname_lang_default:enn
3365                     { \l__zrefclever_setup_language_tl }

```

```

3366         {#1} { bool }
3367     }
3368 }
3369 {
3370     \__zrefclever_opt_bool_gset_true:c
3371     {
3372         \__zrefclever_opt_varname_lang_type:eenn
3373         { \l__zrefclever_setup_language_tl }
3374         { \l__zrefclever_setup_type_tl }
3375         {#1} { bool }
3376     }
3377 }
3378 } ,
3379 #1 / false .code:n =
3380 {
3381     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3382     {
3383         \__zrefclever_opt_bool_gset_false:c
3384         {
3385             \__zrefclever_opt_varname_lang_default:enn
3386             { \l__zrefclever_setup_language_tl }
3387             {#1} { bool }
3388         }
3389     }
3390     {
3391         \__zrefclever_opt_bool_gset_false:c
3392         {
3393             \__zrefclever_opt_varname_lang_type:eenn
3394             { \l__zrefclever_setup_language_tl }
3395             { \l__zrefclever_setup_type_tl }
3396             {#1} { bool }
3397         }
3398     }
3399 } ,
3400 #1 .default:n = true ,
3401 no #1 .meta:n = { #1 = false } ,
3402 no #1 .value_forbidden:n = true ,
3403 }
3404 }

```

6 User interface

6.1 `\zcref`

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

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

```

3405 \NewDocumentCommand \zcref { s O { } m }
3406 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End of definition for `\zcref`.)

`__zrefclever_zcref:nmmn` 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:nmmn {<labels>} {<*>} {<options>}
```

```
3407 \cs_new_protected:Npn \__zrefclever_zcref:nmm #1#2#3
3408 {
3409   \group_begin:
```

Set options.

```
3410   \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```
3411   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3412   \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.

```
3413   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3414   \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3415   \bool_lazy_and:nnT
3416     { \l__zrefclever_zrefcheck_available_bool }
3417     { \l__zrefclever_zcref_with_check_bool }
3418     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3419   \bool_lazy_or:nnT
3420     { \l__zrefclever_typeset_sort_bool }
3421     { \l__zrefclever_typeset_range_bool }
3422     { \__zrefclever_sort_labels: }
```

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

```
3423   \group_begin:
3424   \l__zrefclever_ref_typeset_font_tl
3425   \__zrefclever_typeset_refs:
3426   \group_end:
```

Typeset note.

```
3427   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3428   {
3429     \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3430     { \l__zrefclever_label_type_a_tl }
3431     { \l__zrefclever_ref_language_tl }
3432     \l_tmpa_tl
3433     \l_tmpa_tl
3434     \l__zrefclever_zcref_note_tl
3435   }
```

Integration with zref-check.

```
3436     \bool_lazy_and:nnT
3437     { \l__zrefclever_zrefcheck_available_bool }
3438     { \l__zrefclever_zcref_with_check_bool }
3439     {
3440     \zrefcheck_zcref_end_label_maybe:
3441     \zrefcheck_zcref_run_checks_on_labels:n
3442     { \l__zrefclever_zcref_labels_seq }
3443     }
```

Integration with mathtools.

```
3444     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3445     {
3446     \__zrefclever_mathtools_showonlyrefs:n
3447     { \l__zrefclever_zcref_labels_seq }
3448     }
3449     \group_end:
3450 }
```

(End of definition for __zrefclever_zcref:nmnn.)

```
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
```

```
3451 \seq_new:N \l__zrefclever_zcref_labels_seq
3452 \bool_new:N \l__zrefclever_link_star_bool
```

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

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

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

```
3453 \NewDocumentCommand \zcpageref { s O { } m }
3454 {
3455     \group_begin:
3456     \IfBooleanT {#1}
3457     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3458     \zcref [#2, ref = page] {#3}
3459     \group_end:
3460 }
```

(End of definition for \zcpageref.)

7 Sorting

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

single reference type. Because of this, sorting must take into account the whole chain of “enclosing counters” for the counters of the labels at hand.

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

`\l_zrefclever_label_type_b_tl`

`\l_zrefclever_label_enclval_a_tl` 3461 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 3462 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 3463 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 3464 `\tl_new:N \l__zrefclever_label_enclval_b_tl`

3465 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`

3466 `\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.

3467 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End of definition for `\l__zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

3468 `\int_new:N \l__zrefclever_sort_prior_a_int`

3469 `\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`.

3470 `\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.

3471 `\cs_new_protected:Npn __zrefclever_sort_labels:`

3472 `{`

Store label types sequence.

3473 `\seq_clear:N \l__zrefclever_label_types_seq`

3474 `\tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }`

3475 `{`

3476 `\seq_map_function:NN \l__zrefclever_zcref_labels_seq`

3477 `__zrefclever_label_type_put_new_right:n`

3478 `}`

Sort.

```
3479 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3480 {
3481   \zref@ifrefundefined {##1}
3482   {
3483     \zref@ifrefundefined {##2}
3484     {
3485       % Neither label is defined.
3486       \sort_return_same:
3487     }
3488     {
3489       % The second label is defined, but the first isn't, leave the
3490       % undefined first (to be more visible).
3491       \sort_return_same:
3492     }
3493   }
3494   {
3495     \zref@ifrefundefined {##2}
3496     {
3497       % The first label is defined, but the second isn't, bring the
3498       % second forward.
3499       \sort_return_swapped:
3500     }
3501     {
3502       % The interesting case: both labels are defined. References
3503       % to the "default" property or to the "page" are quite
3504       % different with regard to sorting, so we branch them here to
3505       % specialized functions.
3506       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3507         { \__zrefclever_sort_page:nn {##1} {##2} }
3508         { \__zrefclever_sort_default:nn {##1} {##2} }
3509     }
3510   }
3511 }
3512 }
```

(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)}

3513 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3514 {
3515   \__zrefclever_extract_default:Nnnn
3516   \l__zrefclever_label_type_a_tl {#1} {zc@type} {}
3517   \seq_if_in:NVF \l__zrefclever_label_types_seq
```

```

3518     \l__zrefclever_label_type_a_tl
3519     {
3520     \seq_put_right:NV \l__zrefclever_label_types_seq
3521     \l__zrefclever_label_type_a_tl
3522     }
3523 }

```

(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}}
3524 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3525 {
3526   \__zrefclever_extract_default:Nnnn
3527   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3528   \__zrefclever_extract_default:Nnnn
3529   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3530
3531   \tl_if_eq:NNTF
3532   \l__zrefclever_label_type_a_tl
3533   \l__zrefclever_label_type_b_tl
3534   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3535   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3536 }

```

(End of definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn
\__zrefclever_sort_default_same_type:nn {\label a}} {\label b}}
3537 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3538 {
3539   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3540   {#1} {zc@enclval} { }
3541   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3542   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3543   {#2} {zc@enclval} { }
3544   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3545   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3546   {#1} {externaldocument} { }
3547   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3548   {#2} {externaldocument} { }
3549
3550   \bool_set_false:N \l__zrefclever_sort_decided_bool
3551
3552   % First we check if there's any "external document" difference (coming
3553   % from 'zref-xr') and, if so, sort based on that.
3554   \tl_if_eq:NNTF
3555   \l__zrefclever_label_extdoc_a_tl
3556   \l__zrefclever_label_extdoc_b_tl
3557   {

```



```

3558 \bool_if:nTF
3559 {
3560   \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3561   ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3562 }
3563 {
3564   \bool_set_true:N \l__zrefclever_sort_decided_bool
3565   \sort_return_same:
3566 }
3567 {
3568   \bool_if:nTF
3569   {
3570     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3571     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3572   }
3573   {
3574     \bool_set_true:N \l__zrefclever_sort_decided_bool
3575     \sort_return_swapped:
3576   }
3577   {
3578     \bool_set_true:N \l__zrefclever_sort_decided_bool
3579     % Two different "external documents": last resort, sort by the
3580     % document name itself.
3581     \str_compare:eNeTF
3582     { \l__zrefclever_label_extdoc_b_tl } <
3583     { \l__zrefclever_label_extdoc_a_tl }
3584     { \sort_return_swapped: }
3585     { \sort_return_same: }
3586   }
3587 }
3588 }
3589
3590 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3591 {
3592   \bool_if:nTF
3593   {
3594     % Both are empty: neither label has any (further) "enclosing
3595     % counters" (left).
3596     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3597     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3598   }
3599   {
3600     \bool_set_true:N \l__zrefclever_sort_decided_bool
3601     \int_compare:nNnTF
3602     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3603     >
3604     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3605     { \sort_return_swapped: }
3606     { \sort_return_same: }
3607   }
3608   {
3609     \bool_if:nTF
3610     {
3611       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.

```

```

3612     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3613   }
3614   {
3615     \bool_set_true:N \l__zrefclever_sort_decided_bool
3616     \int_compare:nNnTF
3617       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3618       >
3619       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3620       { \sort_return_swapped: }
3621       { \sort_return_same:   }
3622   }
3623   {
3624     \bool_if:nTF
3625     {
3626       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3627       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3628     }
3629     {
3630       \bool_set_true:N \l__zrefclever_sort_decided_bool
3631       \int_compare:nNnTF
3632         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3633         <
3634         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3635         { \sort_return_same:   }
3636         { \sort_return_swapped: }
3637     }
3638     {
3639       % Neither is empty: we can compare the values of the
3640       % current enclosing counter in the loop, if they are
3641       % equal, we are still in the loop, if they are not, a
3642       % sorting decision can be made directly.
3643       \int_compare:nNnTF
3644         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3645         =
3646         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3647         {
3648           \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3649             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3650           \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3651             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3652         }
3653         {
3654           \bool_set_true:N \l__zrefclever_sort_decided_bool
3655           \int_compare:nNnTF
3656             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3657             >
3658             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3659             { \sort_return_swapped: }
3660             { \sort_return_same:   }
3661         }
3662     }
3663   }
3664 }
3665 }

```

```
3666 }
```

(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>}
```

```
3667 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
3668 {
```

Retrieve sort priorities for *<label a>* and *<label b>*. `\l_zrefclever_typesort_seq` was stored in reverse sequence, and we compute the sort priorities in the negative range, so that we can implicitly rely on ‘0’ being the “last value”.

```
3669 \int_zero:N \l_zrefclever_sort_prior_a_int
3670 \int_zero:N \l_zrefclever_sort_prior_b_int
3671 \seq_map_indexed_inline:Nn \l_zrefclever_typesort_seq
3672 {
3673   \tl_if_eq:nnTF {##2} {{othertypes}}
3674   {
3675     \int_compare:nNnT { \l_zrefclever_sort_prior_a_int } = { 0 }
3676     { \int_set:Nn \l_zrefclever_sort_prior_a_int { - ##1 } }
3677     \int_compare:nNnT { \l_zrefclever_sort_prior_b_int } = { 0 }
3678     { \int_set:Nn \l_zrefclever_sort_prior_b_int { - ##1 } }
3679   }
3680   {
3681     \tl_if_eq:NnTF \l_zrefclever_label_type_a_tl {##2}
3682     { \int_set:Nn \l_zrefclever_sort_prior_a_int { - ##1 } }
3683     {
3684       \tl_if_eq:NnT \l_zrefclever_label_type_b_tl {##2}
3685       { \int_set:Nn \l_zrefclever_sort_prior_b_int { - ##1 } }
3686     }
3687   }
3688 }
```

Then do the actual sorting.

```
3689 \bool_if:nTF
3690 {
3691   \int_compare_p:nNn
3692   { \l_zrefclever_sort_prior_a_int } <
3693   { \l_zrefclever_sort_prior_b_int }
3694 }
3695 { \sort_return_same: }
3696 {
3697   \bool_if:nTF
3698   {
3699     \int_compare_p:nNn
3700     { \l_zrefclever_sort_prior_a_int } >
3701     { \l_zrefclever_sort_prior_b_int }
3702   }
3703   { \sort_return_swapped: }
3704   {
3705     % Sort priorities are equal: the type that occurs first in
3706     % ‘labels’, as given by the user, is kept (or brought) forward.
3707     \seq_map_inline:Nn \l_zrefclever_label_types_seq
3708     {
3709       \tl_if_eq:NnTF \l_zrefclever_label_type_a_tl {##1}
```

```

3710         { \seq_map_break:n { \sort_return_same: } }
3711         {
3712             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3713             { \seq_map_break:n { \sort_return_swapped: } }
3714         }
3715     }
3716 }
3717 }
3718 }

```

(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}

3719 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3720 {
3721     \int_compare:nNnTF
3722     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3723     >
3724     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3725     { \sort_return_swapped: }
3726     { \sort_return_same: }
3727 }

```

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

8 Typesetting

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

While processing the label stack (kept in `\l__zrefclever_typeset_labels_seq`), `__zrefclever_typeset_refs:` “sees” two labels, and two labels only, the “current” one (kept in `\l__zrefclever_label_a_tl`), and the “next” one (kept in `\l__zrefclever_label_b_tl`). However, the typesetting needs (a lot) more information than just these two immediate labels to make a number of critical decisions. Some examples: i) We cannot know if labels “current” and “next” of the same type are a “pair”, or just “elements in a list”, until we examine the label after “next”; ii) If the “next” label is of the same type as the “current”, and it is in immediate sequence to it, it potentially forms a “range”, but we cannot know if “next” is actually the end of the range until we examined an arbitrary number of labels, and found one which is not in sequence from the previous one; iii)

When processing a type block, the “name” comes first, however, we only know if that name should be plural, or if it should be included in the hyperlink, after processing an arbitrary number of labels and find one of a different type. One could naively assume that just examining “next” would be enough for this, since we can know if it is of the same type or not. Alas, “there be ranges”, and a compression operation may boil down to a single element, so we have to process the whole type block to know how its name should be typeset; iv) Similar issues apply to lists of type blocks, each of which is of arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

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

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

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

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

Variables

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

`\l_zrefclever_typeset_last_bool` 3728 `\seq_new:N \l_zrefclever_typeset_labels_seq`

`\l_zrefclever_last_of_type_bool` 3729 `\bool_new:N \l_zrefclever_typeset_last_bool`

3730 `\bool_new:N \l_zrefclever_last_of_type_bool`

(End of definition for `\l_zrefclever_typeset_labels_seq`, `\l_zrefclever_typeset_last_bool`, and `\l_zrefclever_last_of_type_bool`.)

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

`\l_zrefclever_label_count_int` 3731 `\int_new:N \l_zrefclever_type_count_int`

`\l_zrefclever_ref_count_int` 3732 `\int_new:N \l_zrefclever_label_count_int`

3733 `\int_new:N \l_zrefclever_ref_count_int`

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

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

`\l_zrefclever_label_b_tl` 3734 `\tl_new:N \l_zrefclever_label_a_tl`

`\l_zrefclever_typeset_queue_prev_tl` 3735 `\tl_new:N \l_zrefclever_label_b_tl`

`\l_zrefclever_typeset_queue_curr_tl` 3736 `\tl_new:N \l_zrefclever_typeset_queue_prev_tl`

`\l_zrefclever_type_first_label_tl` 3737 `\tl_new:N \l_zrefclever_typeset_queue_curr_tl`

3738 `\tl_new:N \l_zrefclever_type_first_label_tl`

3739 `\tl_new:N \l_zrefclever_type_first_label_type_tl`

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

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

`\l_zrefclever_name_in_link_bool` 3740 `\tl_new:N \l_zrefclever_type_name_tl`

`\l_zrefclever_type_name_missing_bool` 3741 `\bool_new:N \l_zrefclever_name_in_link_bool`

`\l_zrefclever_name_format_tl` 3742 `\bool_new:N \l_zrefclever_type_name_missing_bool`

`\l_zrefclever_name_format_fallback_tl` 3743 `\tl_new:N \l_zrefclever_name_format_tl`

`\l_zrefclever_type_name_gender_seq` 3744 `\tl_new:N \l_zrefclever_name_format_fallback_tl`

3745 `\seq_new:N \l_zrefclever_type_name_gender_seq`

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

`\l_zrefclever_range_count_int` Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

`\l_zrefclever_range_same_count_int` 3746 `\int_new:N \l_zrefclever_range_count_int`

`\l_zrefclever_range_beg_label_tl` 3747 `\int_new:N \l_zrefclever_range_same_count_int`

`\l_zrefclever_range_beg_is_first_bool` 3748 `\tl_new:N \l_zrefclever_range_beg_label_tl`

`\l_zrefclever_range_end_ref_tl` 3749 `\bool_new:N \l_zrefclever_range_beg_is_first_bool`

`\l_zrefclever_next_maybe_range_bool` 3750 `\tl_new:N \l_zrefclever_range_end_ref_tl`

`\l_zrefclever_next_is_same_bool` 3751 `\bool_new:N \l_zrefclever_next_maybe_range_bool`

3752 `\bool_new:N \l_zrefclever_next_is_same_bool`

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

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

```

\l__zrefclever_tlistsep_tl 3753 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_tlastsep_tl 3754 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_namesep_tl 3755 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_pairsep_tl 3756 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_listsep_tl 3757 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_lastsep_tl 3758 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_rangesep_tl 3759 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_namefont_tl 3760 \tl_new:N \l__zrefclever_rangesep_tl
\l__zrefclever_reffont_tl 3761 \tl_new:N \l__zrefclever_namefont_tl
  \l__zrefclever_endrangefunc_tl 3762 \tl_new:N \l__zrefclever_reffont_tl
  \l__zrefclever_endrangeprop_tl 3763 \tl_new:N \l__zrefclever_endrangefunc_tl
  \l__zrefclever_cap_bool 3764 \tl_new:N \l__zrefclever_endrangeprop_tl
\l__zrefclever_abbrev_bool 3765 \bool_new:N \l__zrefclever_cap_bool
  \l__zrefclever_rangetopair_bool 3766 \bool_new:N \l__zrefclever_abbrev_bool
  3767 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

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

```

\l__zrefclever_refbounds_first_sg_seq 3768 \seq_new:N \l__zrefclever_refbounds_first_seq
\l__zrefclever_refbounds_first_pb_seq 3769 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
\l__zrefclever_refbounds_first_rb_seq 3770 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
  \l__zrefclever_refbounds_mid_seq 3771 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
  \l__zrefclever_refbounds_mid_rb_seq 3772 \seq_new:N \l__zrefclever_refbounds_mid_seq
  \l__zrefclever_refbounds_mid_re_seq 3773 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
  \l__zrefclever_refbounds_last_seq 3774 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
  \l__zrefclever_refbounds_last_pe_seq 3775 \seq_new:N \l__zrefclever_refbounds_last_seq
  \l__zrefclever_refbounds_last_re_seq 3776 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_type_first_refbounds_seq 3777 \seq_new:N \l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_set_bool 3778 \seq_new:N \l__zrefclever_type_first_refbounds_seq
  3779 \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`.

```
3780 \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`.

```

3781 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3782 {
3783   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3784   \l__zrefclever_zcref_labels_seq
3785   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3786   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3787   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

3788 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3789 \tl_clear:N \l__zrefclever_range_beg_label_tl
3790 \tl_clear:N \l__zrefclever_range_end_ref_tl
3791 \int_zero:N \l__zrefclever_label_count_int
3792 \int_zero:N \l__zrefclever_type_count_int
3793 \int_zero:N \l__zrefclever_ref_count_int
3794 \int_zero:N \l__zrefclever_range_count_int
3795 \int_zero:N \l__zrefclever_range_same_count_int
3796 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3797 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3798
3799 % Get type block options (not type-specific).
3800 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3801   { \l__zrefclever_label_type_a_tl }
3802   { \l__zrefclever_ref_language_tl }
3803   \l__zrefclever_tpairsep_tl
3804 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3805   { \l__zrefclever_label_type_a_tl }
3806   { \l__zrefclever_ref_language_tl }
3807   \l__zrefclever_tlistsep_tl
3808 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3809   { \l__zrefclever_label_type_a_tl }
3810   { \l__zrefclever_ref_language_tl }
3811   \l__zrefclever_tlastsep_tl
3812
3813 % Process label stack.
3814 \bool_set_false:N \l__zrefclever_typeset_last_bool
3815 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3816   {
3817     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3818     \l__zrefclever_label_a_tl
3819     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3820       {
3821         \tl_clear:N \l__zrefclever_label_b_tl
3822         \bool_set_true:N \l__zrefclever_typeset_last_bool
3823       }
3824       {
3825         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3826         \l__zrefclever_label_b_tl
3827       }
3828
3829     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3830     {
3831       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3832       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3833     }
3834     {
3835       \__zrefclever_extract_default:NVnn
3836       \l__zrefclever_label_type_a_tl
3837       \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3838       \__zrefclever_extract_default:NVnn
3839       \l__zrefclever_label_type_b_tl
3840       \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3841     }

```



```

3842
3843 % First, we establish whether the "current label" (i.e. 'a') is the
3844 % last one of its type. This can happen because the "next label"
3845 % (i.e. 'b') is of a different type (or different definition status),
3846 % or because we are at the end of the list.
3847 \bool_if:NTF \l__zrefclever_typeset_last_bool
3848 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3849 {
3850   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3851   {
3852     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3853     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3854     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3855   }
3856   {
3857     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3858     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3859     {
3860       % Neither is undefined, we must check the types.
3861       \tl_if_eq:NNTF
3862         \l__zrefclever_label_type_a_tl
3863         \l__zrefclever_label_type_b_tl
3864         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3865         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3866     }
3867   }
3868 }
3869
3870 % Handle warnings in case of reference or type undefined.
3871 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3872 \zref@refused { \l__zrefclever_label_a_tl }
3873 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3874 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3875 {}
3876 {
3877   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3878   {
3879     \msg_warning:nmx { zref-clever } { missing-type }
3880     { \l__zrefclever_label_a_tl }
3881   }
3882   \zref@ifrefcontainsprop
3883   { \l__zrefclever_label_a_tl }
3884   { \l__zrefclever_ref_property_tl }
3885   { }
3886   {
3887     \msg_warning:nmx { zref-clever } { missing-property }
3888     { \l__zrefclever_ref_property_tl }
3889     { \l__zrefclever_label_a_tl }
3890   }
3891 }
3892
3893 % Get possibly type-specific separators, refbounds, font and other
3894 % options, once per type.
3895 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```

```

3896 {
3897   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3898     { \l__zrefclever_label_type_a_tl }
3899     { \l__zrefclever_ref_language_tl }
3900     \l__zrefclever_namesep_tl
3901   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3902     { \l__zrefclever_label_type_a_tl }
3903     { \l__zrefclever_ref_language_tl }
3904     \l__zrefclever_pairsep_tl
3905   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3906     { \l__zrefclever_label_type_a_tl }
3907     { \l__zrefclever_ref_language_tl }
3908     \l__zrefclever_listsep_tl
3909   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3910     { \l__zrefclever_label_type_a_tl }
3911     { \l__zrefclever_ref_language_tl }
3912     \l__zrefclever_lastsep_tl
3913   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3914     { \l__zrefclever_label_type_a_tl }
3915     { \l__zrefclever_ref_language_tl }
3916     \l__zrefclever_rangesep_tl
3917   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3918     { \l__zrefclever_label_type_a_tl }
3919     { \l__zrefclever_ref_language_tl }
3920     \l__zrefclever_namefont_tl
3921   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3922     { \l__zrefclever_label_type_a_tl }
3923     { \l__zrefclever_ref_language_tl }
3924     \l__zrefclever_reffont_tl
3925   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3926     { \l__zrefclever_label_type_a_tl }
3927     { \l__zrefclever_ref_language_tl }
3928     \l__zrefclever_endrangefunc_tl
3929   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3930     { \l__zrefclever_label_type_a_tl }
3931     { \l__zrefclever_ref_language_tl }
3932     \l__zrefclever_endrangeprop_tl
3933   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3934     { \l__zrefclever_label_type_a_tl }
3935     { \l__zrefclever_ref_language_tl }
3936     \l__zrefclever_cap_bool
3937   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3938     { \l__zrefclever_label_type_a_tl }
3939     { \l__zrefclever_ref_language_tl }
3940     \l__zrefclever_abbrev_bool
3941   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3942     { \l__zrefclever_label_type_a_tl }
3943     { \l__zrefclever_ref_language_tl }
3944     \l__zrefclever_rangetopair_bool
3945   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3946     { \l__zrefclever_label_type_a_tl }
3947     { \l__zrefclever_ref_language_tl }
3948     \l__zrefclever_refbounds_first_seq
3949   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3950         { \l__zrefclever_label_type_a_tl }
3951         { \l__zrefclever_ref_language_tl }
3952         \l__zrefclever_refbounds_first_sg_seq
3953     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3954         { \l__zrefclever_label_type_a_tl }
3955         { \l__zrefclever_ref_language_tl }
3956         \l__zrefclever_refbounds_first_pb_seq
3957     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3958         { \l__zrefclever_label_type_a_tl }
3959         { \l__zrefclever_ref_language_tl }
3960         \l__zrefclever_refbounds_first_rb_seq
3961     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3962         { \l__zrefclever_label_type_a_tl }
3963         { \l__zrefclever_ref_language_tl }
3964         \l__zrefclever_refbounds_mid_seq
3965     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3966         { \l__zrefclever_label_type_a_tl }
3967         { \l__zrefclever_ref_language_tl }
3968         \l__zrefclever_refbounds_mid_rb_seq
3969     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3970         { \l__zrefclever_label_type_a_tl }
3971         { \l__zrefclever_ref_language_tl }
3972         \l__zrefclever_refbounds_mid_re_seq
3973     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3974         { \l__zrefclever_label_type_a_tl }
3975         { \l__zrefclever_ref_language_tl }
3976         \l__zrefclever_refbounds_last_seq
3977     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3978         { \l__zrefclever_label_type_a_tl }
3979         { \l__zrefclever_ref_language_tl }
3980         \l__zrefclever_refbounds_last_pe_seq
3981     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3982         { \l__zrefclever_label_type_a_tl }
3983         { \l__zrefclever_ref_language_tl }
3984         \l__zrefclever_refbounds_last_re_seq
3985     }
3986
3987     % Here we send this to a couple of auxiliary functions.
3988     \bool_if:NTF \l__zrefclever_last_of_type_bool
3989         % There exists no next label of the same type as the current.
3990         { \__zrefclever_typeset_refs_last_of_type: }
3991         % There exists a next label of the same type as the current.
3992         { \__zrefclever_typeset_refs_not_last_of_type: }
3993     }
3994 }

```

(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.

```

3995 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3996 {
3997   % Process the current label to the current queue.
3998   \int_case:nnF { \l__zrefclever_label_count_int }
3999   {
4000     % It is the last label of its type, but also the first one, and that's
4001     % what matters here: just store it.
4002     % Test: 'zc-typeset01.lvt': "Last of type: single"
4003     { 0 }
4004     {
4005       \tl_set:NV \l__zrefclever_type_first_label_tl
4006       \l__zrefclever_label_a_tl
4007       \tl_set:NV \l__zrefclever_type_first_label_type_tl
4008       \l__zrefclever_label_type_a_tl
4009       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4010       \l__zrefclever_refbounds_first_sg_seq
4011       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4012     }
4013
4014     % The last is the second: we have a pair (if not repeated).
4015     % Test: 'zc-typeset01.lvt': "Last of type: pair"
4016     { 1 }
4017     {
4018       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4019       {
4020         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4021         \l__zrefclever_refbounds_first_sg_seq
4022         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4023       }
4024       {
4025         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4026         {
4027           \exp_not:V \l__zrefclever_pairsep_tl
4028           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4029           \l__zrefclever_refbounds_last_pe_seq
4030         }
4031         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4032         \l__zrefclever_refbounds_first_pb_seq
4033         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4034       }
4035     }
4036   }
4037   % Last is third or more of its type: without repetition, we'd have the
4038   % last element on a list, but control for possible repetition.
4039   {
4040     \int_case:nnF { \l__zrefclever_range_count_int }
4041     {
4042       % There was no range going on.
4043       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4044       { 0 }

```

```

4045 {
4046   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4047   {
4048     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4049     {
4050       \exp_not:V \l__zrefclever_pairsep_tl
4051       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4052       \l__zrefclever_refbounds_last_pe_seq
4053     }
4054   }
4055   {
4056     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4057     {
4058       \exp_not:V \l__zrefclever_lastsep_tl
4059       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4060       \l__zrefclever_refbounds_last_seq
4061     }
4062   }
4063 }
4064 % Last in the range is also the second in it.
4065 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4066 { 1 }
4067 {
4068   \int_compare:nNnTF
4069   { \l__zrefclever_range_same_count_int } = { 1 }
4070   {
4071     % We know 'range_beg_is_first_bool' is false, since this is
4072     % the second element in the range, but the third or more in
4073     % the type list.
4074     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4075     {
4076       \exp_not:V \l__zrefclever_pairsep_tl
4077       \__zrefclever_get_ref:VN
4078       \l__zrefclever_range_beg_label_tl
4079       \l__zrefclever_refbounds_last_pe_seq
4080     }
4081     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4082     \l__zrefclever_refbounds_first_pb_seq
4083     \bool_set_true:N
4084     \l__zrefclever_type_first_refbounds_set_bool
4085   }
4086   {
4087     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4088     {
4089       \exp_not:V \l__zrefclever_listsep_tl
4090       \__zrefclever_get_ref:VN
4091       \l__zrefclever_range_beg_label_tl
4092       \l__zrefclever_refbounds_mid_seq
4093       \exp_not:V \l__zrefclever_lastsep_tl
4094       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4095       \l__zrefclever_refbounds_last_seq
4096     }
4097   }
4098 }

```

```

4099     }
4100     % Last in the range is third or more in it.
4101     {
4102     \int_case:nmF
4103     {
4104     \l__zrefclever_range_count_int -
4105     \l__zrefclever_range_same_count_int
4106     }
4107     {
4108     % Repetition, not a range.
4109     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4110     { 0 }
4111     {
4112     % If 'range_beg_is_first_bool' is true, it means it was also
4113     % the first of the type, and hence its typesetting was
4114     % already handled, and we just have to set rebounds.
4115     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4116     {
4117     \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4118     \l__zrefclever_rebounds_first_sg_seq
4119     \bool_set_true:N
4120     \l__zrefclever_type_first_rebounds_set_bool
4121     }
4122     {
4123     \int_compare:nNnTF
4124     { \l__zrefclever_ref_count_int } < { 2 }
4125     {
4126     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4127     {
4128     \exp_not:V \l__zrefclever_pairsep_tl
4129     \__zrefclever_get_ref:VN
4130     \l__zrefclever_range_beg_label_tl
4131     \l__zrefclever_rebounds_last_pe_seq
4132     }
4133     }
4134     {
4135     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4136     {
4137     \exp_not:V \l__zrefclever_lastsep_tl
4138     \__zrefclever_get_ref:VN
4139     \l__zrefclever_range_beg_label_tl
4140     \l__zrefclever_rebounds_last_seq
4141     }
4142     }
4143     }
4144     }
4145     % A 'range', but with no skipped value, treat as pair if range
4146     % started with first of type, otherwise as list.
4147     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4148     { 1 }
4149     {
4150     % Ditto.
4151     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4152     {

```

```

4153         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4154         \l__zrefclever_refbounds_first_pb_seq
4155     \bool_set_true:N
4156         \l__zrefclever_type_first_refbounds_set_bool
4157     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4158     {
4159         \exp_not:V \l__zrefclever_pairsep_tl
4160         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4161         \l__zrefclever_refbounds_last_pe_seq
4162     }
4163 }
4164 {
4165     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4166     {
4167         \exp_not:V \l__zrefclever_listsep_tl
4168         \__zrefclever_get_ref:VN
4169         \l__zrefclever_range_beg_label_tl
4170         \l__zrefclever_refbounds_mid_seq
4171     }
4172     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4173     {
4174         \exp_not:V \l__zrefclever_lastsep_tl
4175         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4176         \l__zrefclever_refbounds_last_seq
4177     }
4178 }
4179 }
4180 }
4181 {
4182     % An actual range.
4183     % Test: 'zc-typeset01.lvt': "Last of type: range"
4184     % Ditto.
4185     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4186     {
4187         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4188         \l__zrefclever_refbounds_first_rb_seq
4189         \bool_set_true:N
4190         \l__zrefclever_type_first_refbounds_set_bool
4191     }
4192     {
4193         \int_compare:nNnTF
4194         { \l__zrefclever_ref_count_int } < { 2 }
4195         {
4196             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4197             {
4198                 \exp_not:V \l__zrefclever_pairsep_tl
4199                 \__zrefclever_get_ref:VN
4200                 \l__zrefclever_range_beg_label_tl
4201                 \l__zrefclever_refbounds_mid_rb_seq
4202             }
4203             \seq_set_eq:NN
4204             \l__zrefclever_type_first_refbounds_seq
4205             \l__zrefclever_refbounds_first_pb_seq
4206             \bool_set_true:N

```

```

4207         \l__zrefclever_type_first_refbounds_set_bool
4208     }
4209     {
4210         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4211         {
4212             \exp_not:V \l__zrefclever_lastsep_tl
4213             \__zrefclever_get_ref:VN
4214             \l__zrefclever_range_beg_label_tl
4215             \l__zrefclever_refbounds_mid_rb_seq
4216         }
4217     }
4218 }
4219 \bool_lazy_and:nnTF
4220 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4221 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4222 {
4223     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4224     \l__zrefclever_range_beg_label_tl
4225     \l__zrefclever_label_a_tl
4226     \l__zrefclever_range_end_ref_tl
4227     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4228     {
4229         \exp_not:V \l__zrefclever_rangesep_tl
4230         \__zrefclever_get_ref_endrange:VVN
4231         \l__zrefclever_label_a_tl
4232         \l__zrefclever_range_end_ref_tl
4233         \l__zrefclever_refbounds_last_re_seq
4234     }
4235 }
4236 {
4237     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4238     {
4239         \exp_not:V \l__zrefclever_rangesep_tl
4240         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4241         \l__zrefclever_refbounds_last_re_seq
4242     }
4243 }
4244 }
4245 }
4246 }
4247
4248 % Handle "range" option. The idea is simple: if the queue is not empty,
4249 % we replace it with the end of the range (or pair). We can still
4250 % retrieve the end of the range from 'label_a' since we know to be
4251 % processing the last label of its type at this point.
4252 \bool_if:NT \l__zrefclever_typeset_range_bool
4253 {
4254     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4255     {
4256         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4257         { }
4258         {
4259             \msg_warning:nxx { zref-clever } { single-element-range }
4260             { \l__zrefclever_type_first_label_type_tl }

```



```

4261     }
4262   }
4263   {
4264     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4265     \bool_if:NT \l__zrefclever_rangetopair_bool
4266     {
4267       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4268       { }
4269       {
4270         \__zrefclever_labels_in_sequence:nn
4271         { \l__zrefclever_type_first_label_tl }
4272         { \l__zrefclever_label_a_tl }
4273       }
4274     }
4275     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4276     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4277     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4278     {
4279       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4280       {
4281         \exp_not:V \l__zrefclever_pairsep_tl
4282         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4283         \l__zrefclever_refbounds_last_pe_seq
4284       }
4285       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4286       \l__zrefclever_refbounds_first_pb_seq
4287       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4288     }
4289     {
4290       \bool_lazy_and:nnTF
4291       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4292       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VWN } }
4293       {
4294         % We must get 'type_first_label_tl' instead of
4295         % 'range_beg_label_tl' here, since it is not necessary
4296         % that the first of type was actually starting a range for
4297         % the 'range' option to be used.
4298         \use:c { \l__zrefclever_endrangefunc_tl :VWN }
4299         \l__zrefclever_type_first_label_tl
4300         \l__zrefclever_label_a_tl
4301         \l__zrefclever_range_end_ref_tl
4302         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4303         {
4304           \exp_not:V \l__zrefclever_rangesep_tl
4305           \__zrefclever_get_ref_endrange:VWN
4306           \l__zrefclever_label_a_tl
4307           \l__zrefclever_range_end_ref_tl
4308           \l__zrefclever_refbounds_last_re_seq
4309         }
4310       }
4311     }
4312     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4313     {
4314       \exp_not:V \l__zrefclever_rangesep_tl

```

```

4315         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4316         \l__zrefclever_refbounds_last_re_seq
4317     }
4318 }
4319 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4320 \l__zrefclever_refbounds_first_rb_seq
4321 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4322 }
4323 }
4324 }
4325
4326 % If none of the special cases for the first of type refbounds have been
4327 % set, do it.
4328 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4329 {
4330     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4331     \l__zrefclever_refbounds_first_seq
4332 }
4333
4334 % Now that the type block is finished, we can add the name and the first
4335 % ref to the queue. Also, if "typeset" option is not "both", handle it
4336 % here as well.
4337 \_zrefclever_type_name_setup:
4338 \bool_if:nTF
4339 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4340 {
4341     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4342     { \_zrefclever_get_ref_first: }
4343 }
4344 {
4345     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4346     {
4347         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4348         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4349         {
4350             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4351             \l__zrefclever_type_first_refbounds_seq
4352         }
4353     }
4354     {
4355         \bool_if:NTF \l__zrefclever_typeset_name_bool
4356         {
4357             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4358             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4359             {
4360                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4361                 {
4362                     \exp_not:N \group_begin:
4363                     \exp_not:V \l__zrefclever_namefont_tl
4364                     \_zrefclever_hyperlink:nnn
4365                     {
4366                         \_zrefclever_extract_url_unexp:V
4367                         \l__zrefclever_type_first_label_tl
4368                     }

```

```

4369         {
4370             \l__zrefclever_extract_unexp:Vnn
4371             \l__zrefclever_type_first_label_tl
4372             { anchor } { }
4373         }
4374         { \exp_not:V \l__zrefclever_type_name_tl }
4375         \exp_not:N \group_end:
4376     }
4377     {
4378         \exp_not:N \group_begin:
4379         \exp_not:V \l__zrefclever_namefont_tl
4380         \exp_not:V \l__zrefclever_type_name_tl
4381         \exp_not:N \group_end:
4382     }
4383 }
4384 }
4385 {
4386     % Logically, this case would correspond to "typeset=none", but
4387     % it should not occur, given that the options are set up to
4388     % typeset either "ref" or "name". Still, leave here a
4389     % sensible fallback, equal to the behavior of "both".
4390     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4391     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4392     { \l__zrefclever_get_ref_first: }
4393 }
4394 }
4395 }
4396
4397 % Typeset the previous type block, if there is one.
4398 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4399 {
4400     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4401     { \l__zrefclever_tlistsep_tl }
4402     \l__zrefclever_typeset_queue_prev_tl
4403 }
4404
4405 % Extra log for testing.
4406 \bool_if:NT \l__zrefclever_verbose_testing_bool
4407 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4408
4409 % Wrap up loop, or prepare for next iteration.
4410 \bool_if:NTF \l__zrefclever_typeset_last_bool
4411 {
4412     % We are finishing, typeset the current queue.
4413     \int_case:nnF { \l__zrefclever_type_count_int }
4414     {
4415         % Single type.
4416         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4417         { 0 }
4418         { \l__zrefclever_typeset_queue_curr_tl }
4419         % Pair of types.
4420         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4421         { 1 }
4422     }

```

```

4423         \l__zrefclever_tpairsep_tl
4424         \l__zrefclever_typeset_queue_curr_tl
4425     }
4426 }
4427 {
4428     % Last in list of types.
4429     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4430     \l__zrefclever_tlastsep_tl
4431     \l__zrefclever_typeset_queue_curr_tl
4432 }
4433 % And nudge in case of multitype reference.
4434 \bool_lazy_all:nT
4435 {
4436     { \l__zrefclever_nudge_enabled_bool }
4437     { \l__zrefclever_nudge_multitype_bool }
4438     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4439 }
4440 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4441 }
4442 {
4443     % There are further labels, set variables for next iteration.
4444     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4445         \l__zrefclever_typeset_queue_curr_tl
4446     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4447     \tl_clear:N \l__zrefclever_type_first_label_tl
4448     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4449     \tl_clear:N \l__zrefclever_range_beg_label_tl
4450     \tl_clear:N \l__zrefclever_range_end_ref_tl
4451     \int_zero:N \l__zrefclever_label_count_int
4452     \int_zero:N \l__zrefclever_ref_count_int
4453     \int_incr:N \l__zrefclever_type_count_int
4454     \int_zero:N \l__zrefclever_range_count_int
4455     \int_zero:N \l__zrefclever_range_same_count_int
4456     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4457     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4458 }
4459 }

```

(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.

```

4460 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4461 {
4462     % Signal if next label may form a range with the current one (only
4463     % considered if compression is enabled in the first place).
4464     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4465     \bool_set_false:N \l__zrefclever_next_is_same_bool
4466     \bool_if:NT \l__zrefclever_typeset_compress_bool
4467     {
4468         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4469         { }
4470         {
4471             \__zrefclever_labels_in_sequence:mn
4472             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4473     }
4474   }
4475
4476 % Process the current label to the current queue.
4477 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4478 {
4479   % Current label is the first of its type (also not the last, but it
4480   % doesn't matter here): just store the label.
4481   \tl_set:NV \l__zrefclever_type_first_label_tl
4482     \l__zrefclever_label_a_tl
4483   \tl_set:NV \l__zrefclever_type_first_label_type_tl
4484     \l__zrefclever_label_type_a_tl
4485   \int_incr:N \l__zrefclever_ref_count_int
4486
4487   % If the next label may be part of a range, signal it (we deal with it
4488   % as the "first", and must do it there, to handle hyperlinking), but
4489   % also step the range counters.
4490   % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4491   \bool_if:NT \l__zrefclever_next_maybe_range_bool
4492     {
4493       \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4494       \tl_set:NV \l__zrefclever_range_beg_label_tl
4495         \l__zrefclever_label_a_tl
4496       \tl_clear:N \l__zrefclever_range_end_ref_tl
4497       \int_incr:N \l__zrefclever_range_count_int
4498       \bool_if:NT \l__zrefclever_next_is_same_bool
4499         { \int_incr:N \l__zrefclever_range_same_count_int }
4500     }
4501   }
4502 {
4503   % Current label is neither the first (nor the last) of its type.
4504   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4505     {
4506     % Starting, or continuing a range.
4507     \int_compare:nNnTF
4508       { \l__zrefclever_range_count_int } = { 0 }
4509       {
4510       % There was no range going, we are starting one.
4511       \tl_set:NV \l__zrefclever_range_beg_label_tl
4512         \l__zrefclever_label_a_tl
4513       \tl_clear:N \l__zrefclever_range_end_ref_tl
4514       \int_incr:N \l__zrefclever_range_count_int
4515       \bool_if:NT \l__zrefclever_next_is_same_bool
4516         { \int_incr:N \l__zrefclever_range_same_count_int }
4517       }
4518       {
4519       % Second or more in the range, but not the last.
4520       \int_incr:N \l__zrefclever_range_count_int
4521       \bool_if:NT \l__zrefclever_next_is_same_bool
4522         { \int_incr:N \l__zrefclever_range_same_count_int }
4523       }
4524     }
4525   }
4526   % Next element is not in sequence: there was no range, or we are

```

```

4527 % closing one.
4528 \int_case:nNF { \l__zrefclever_range_count_int }
4529 {
4530 % There was no range going on.
4531 % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4532 { 0 }
4533 {
4534 \int_incr:N \l__zrefclever_ref_count_int
4535 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4536 {
4537 \exp_not:V \l__zrefclever_listsep_tl
4538 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4539 \l__zrefclever_refbounds_mid_seq
4540 }
4541 }
4542 % Last is second in the range: if 'range_same_count' is also
4543 % '1', it's a repetition (drop it), otherwise, it's a "pair
4544 % within a list", treat as list.
4545 % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4546 % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4547 { 1 }
4548 {
4549 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4550 {
4551 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4552 \l__zrefclever_refbounds_first_seq
4553 \bool_set_true:N
4554 \l__zrefclever_type_first_refbounds_set_bool
4555 }
4556 {
4557 \int_incr:N \l__zrefclever_ref_count_int
4558 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4559 {
4560 \exp_not:V \l__zrefclever_listsep_tl
4561 \__zrefclever_get_ref:VN
4562 \l__zrefclever_range_beg_label_tl
4563 \l__zrefclever_refbounds_mid_seq
4564 }
4565 }
4566 \int_compare:nNF
4567 { \l__zrefclever_range_same_count_int } = { 1 }
4568 {
4569 \int_incr:N \l__zrefclever_ref_count_int
4570 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4571 {
4572 \exp_not:V \l__zrefclever_listsep_tl
4573 \__zrefclever_get_ref:VN
4574 \l__zrefclever_label_a_tl
4575 \l__zrefclever_refbounds_mid_seq
4576 }
4577 }
4578 }
4579 }
4580 {

```

```

4581 % Last is third or more in the range: if 'range_count' and
4582 % 'range_same_count' are the same, its a repetition (drop it),
4583 % if they differ by '1', its a list, if they differ by more,
4584 % it is a real range.
4585 \int_case:nnF
4586 {
4587   \l__zrefclever_range_count_int -
4588   \l__zrefclever_range_same_count_int
4589 }
4590 {
4591   % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4592   { 0 }
4593   {
4594     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4595     {
4596       \seq_set_eq:NN
4597         \l__zrefclever_type_first_refbounds_seq
4598         \l__zrefclever_refbounds_first_seq
4599       \bool_set_true:N
4600         \l__zrefclever_type_first_refbounds_set_bool
4601     }
4602     {
4603       \int_incr:N \l__zrefclever_ref_count_int
4604       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4605         {
4606           \exp_not:V \l__zrefclever_listsep_tl
4607           \__zrefclever_get_ref:VN
4608             \l__zrefclever_range_beg_label_tl
4609             \l__zrefclever_refbounds_mid_seq
4610         }
4611     }
4612   }
4613   % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4614   { 1 }
4615   {
4616     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4617     {
4618       \seq_set_eq:NN
4619         \l__zrefclever_type_first_refbounds_seq
4620         \l__zrefclever_refbounds_first_seq
4621       \bool_set_true:N
4622         \l__zrefclever_type_first_refbounds_set_bool
4623     }
4624     {
4625       \int_incr:N \l__zrefclever_ref_count_int
4626       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4627         {
4628           \exp_not:V \l__zrefclever_listsep_tl
4629           \__zrefclever_get_ref:VN
4630             \l__zrefclever_range_beg_label_tl
4631             \l__zrefclever_refbounds_mid_seq
4632         }
4633     }
4634     \int_incr:N \l__zrefclever_ref_count_int

```

```

4635 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4636 {
4637   \exp_not:V \l__zrefclever_listsep_tl
4638   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4639   \l__zrefclever_refbounds_mid_seq
4640 }
4641 }
4642 }
4643 {
4644   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4645   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4646   {
4647     \seq_set_eq:NN
4648     \l__zrefclever_type_first_refbounds_seq
4649     \l__zrefclever_refbounds_first_rb_seq
4650     \bool_set_true:N
4651     \l__zrefclever_type_first_refbounds_set_bool
4652   }
4653   {
4654     \int_incr:N \l__zrefclever_ref_count_int
4655     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4656     {
4657       \exp_not:V \l__zrefclever_listsep_tl
4658       \__zrefclever_get_ref:VN
4659       \l__zrefclever_range_beg_label_tl
4660       \l__zrefclever_refbounds_mid_rb_seq
4661     }
4662   }
4663   % For the purposes of the serial comma, and thus for the
4664   % distinction of 'lastsep' and 'pairsep', a "range" counts
4665   % as one. Since 'range_beg' has already been counted
4666   % (here or with the first of type), we refrain from
4667   % incrementing 'ref_count_int'.
4668   \bool_lazy_and:nnTF
4669   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4670   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4671   {
4672     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4673     \l__zrefclever_range_beg_label_tl
4674     \l__zrefclever_label_a_tl
4675     \l__zrefclever_range_end_ref_tl
4676     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4677     {
4678       \exp_not:V \l__zrefclever_rangesep_tl
4679       \__zrefclever_get_ref_endrange:VVN
4680       \l__zrefclever_label_a_tl
4681       \l__zrefclever_range_end_ref_tl
4682       \l__zrefclever_refbounds_mid_re_seq
4683     }
4684   }
4685   {
4686     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4687     {
4688       \exp_not:V \l__zrefclever_rangesep_tl

```



```

4689             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4690             \l__zrefclever_refbounds_mid_re_seq
4691         }
4692     }
4693 }
4694 }
4695 % We just closed a range, reset 'range_beg_is_first' in case a
4696 % second range for the same type occurs, in which case its
4697 % 'range_beg' will no longer be 'first'.
4698 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4699 % Reset counters.
4700 \int_zero:N \l__zrefclever_range_count_int
4701 \int_zero:N \l__zrefclever_range_same_count_int
4702 }
4703 }
4704 % Step label counter for next iteration.
4705 \int_incr:N \l__zrefclever_label_count_int
4706 }

```

(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 ncical requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref:nN` and `__zrefclever_get_ref_first:` get called, as they must, in the context of `x` type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the `n` signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, `\zref@default` or `\hyper@@link`). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

```

\__zrefclever_ref_default: Default values for undefined references and undefined type names, respectively. We are
\__zrefclever_name_default: 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.
4707 \cs_new_protected:Npn \__zrefclever_ref_default:
4708   { \zref@default }
4709 \cs_new_protected:Npn \__zrefclever_name_default:
4710   { \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`.

```

\zref@ifrefcontainsprop {#1} { \l_zrefclever_ref_property_tl }
{
  \bool_if:nTF
  {
    \l_zrefclever_hyperlink_bool &&
    ! \l_zrefclever_link_star_bool
  }
  {
    \seq_item:Nn #2 { 1 }
    \_zrefclever_hyperlink:nnn
    { \_zrefclever_extract_url_unexp:n {#1} }
    { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
    {
      \seq_item:Nn #2 { 2 }
      \exp_not:N \group_begin:
      \exp_not:V \l_zrefclever_reffont_tl
      \_zrefclever_extract_unexp:nvn {#1}
      { \l_zrefclever_ref_property_tl } { }
      \exp_not:N \group_end:
      \seq_item:Nn #2 { 3 }
    }
    \seq_item:Nn #2 { 4 }
  }
  {
    \seq_item:Nn #2 { 1 }
    \seq_item:Nn #2 { 2 }
    \exp_not:N \group_begin:
    \exp_not:V \l_zrefclever_reffont_tl
    \_zrefclever_extract_unexp:nvn {#1}
    { \l_zrefclever_ref_property_tl } { }
    \exp_not:N \group_end:
    \seq_item:Nn #2 { 3 }
    \seq_item:Nn #2 { 4 }
  }
}
{ \_zrefclever_ref_default: }
}
\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>}
4751 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4752 {
4753   \str_if_eq:nnTF {#2} { zc@missingproperty }
4754   { \__zrefclever_ref_default: }
4755   {
4756     \bool_if:nTF
4757     {
4758       \l__zrefclever_hyperlink_bool &&
4759       ! \l__zrefclever_link_star_bool
4760     }
4761     {
4762       \seq_item:Nn #3 { 1 }
4763       \__zrefclever_hyperlink:nnn
4764       { \__zrefclever_extract_url_unexp:n {#1} }
4765       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4766       {
4767         \seq_item:Nn #3 { 2 }
4768         \exp_not:N \group_begin:
4769         \exp_not:V \l__zrefclever_reffont_tl
4770         \exp_not:n {#2}
4771         \exp_not:N \group_end:
4772         \seq_item:Nn #3 { 3 }
4773       }
4774       \seq_item:Nn #3 { 4 }
4775     }
4776     {
4777       \seq_item:Nn #3 { 1 }
4778       \seq_item:Nn #3 { 2 }
4779       \exp_not:N \group_begin:
4780       \exp_not:V \l__zrefclever_reffont_tl
4781       \exp_not:n {#2}
4782       \exp_not:N \group_end:
4783       \seq_item:Nn #3 { 3 }
4784       \seq_item:Nn #3 { 4 }
4785     }
4786   }
4787 }
4788 \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.

```

4789 \cs_new:Npn \__zrefclever_get_ref_first:
4790 {
4791   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4792   { \__zrefclever_ref_default: }
4793   {
4794     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4795 {
4796 \zref@ifrefcontainsprop
4797 { \l__zrefclever_type_first_label_tl }
4798 { \l__zrefclever_ref_property_tl }
4799 {
4800 \__zrefclever_hyperlink:nnn
4801 {
4802 \__zrefclever_extract_url_unexp:V
4803 \l__zrefclever_type_first_label_tl
4804 }
4805 {
4806 \__zrefclever_extract_unexp:Vnn
4807 \l__zrefclever_type_first_label_tl { anchor } { }
4808 }
4809 {
4810 \exp_not:N \group_begin:
4811 \exp_not:V \l__zrefclever_namefont_tl
4812 \exp_not:V \l__zrefclever_type_name_tl
4813 \exp_not:N \group_end:
4814 \exp_not:V \l__zrefclever_namesep_tl
4815 \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4816 \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4817 \exp_not:N \group_begin:
4818 \exp_not:V \l__zrefclever_reffont_tl
4819 \__zrefclever_extract_unexp:Vvn
4820 \l__zrefclever_type_first_label_tl
4821 { \l__zrefclever_ref_property_tl } { }
4822 \exp_not:N \group_end:
4823 \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4824 }
4825 \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4826 }
4827 {
4828 \exp_not:N \group_begin:
4829 \exp_not:V \l__zrefclever_namefont_tl
4830 \exp_not:V \l__zrefclever_type_name_tl
4831 \exp_not:N \group_end:
4832 \exp_not:V \l__zrefclever_namesep_tl
4833 \__zrefclever_ref_default:
4834 }
4835 }
4836 {
4837 \bool_if:nTF \l__zrefclever_type_name_missing_bool
4838 {
4839 \__zrefclever_name_default:
4840 \exp_not:V \l__zrefclever_namesep_tl
4841 }
4842 {
4843 \exp_not:N \group_begin:
4844 \exp_not:V \l__zrefclever_namefont_tl
4845 \exp_not:V \l__zrefclever_type_name_tl
4846 \exp_not:N \group_end:
4847 \tl_if_empty:NF \l__zrefclever_type_name_tl
4848 { \exp_not:V \l__zrefclever_namesep_tl }

```

```

4849     }
4850 \zref@ifrefcontainsprop
4851 { \l__zrefclever_type_first_label_tl }
4852 { \l__zrefclever_ref_property_tl }
4853 {
4854   \bool_if:nTF
4855   {
4856     \l__zrefclever_hyperlink_bool &&
4857     ! \l__zrefclever_link_star_bool
4858   }
4859   {
4860     \seq_item:Nn
4861     \l__zrefclever_type_first_refbounds_seq { 1 }
4862     \__zrefclever_hyperlink:nnn
4863     {
4864       \__zrefclever_extract_url_unexp:V
4865       \l__zrefclever_type_first_label_tl
4866     }
4867     {
4868       \__zrefclever_extract_unexp:Vnn
4869       \l__zrefclever_type_first_label_tl { anchor } { }
4870     }
4871     {
4872       \seq_item:Nn
4873       \l__zrefclever_type_first_refbounds_seq { 2 }
4874       \exp_not:N \group_begin:
4875       \exp_not:V \l__zrefclever_reffont_tl
4876       \__zrefclever_extract_unexp:Vvn
4877       \l__zrefclever_type_first_label_tl
4878       { \l__zrefclever_ref_property_tl } { }
4879       \exp_not:N \group_end:
4880       \seq_item:Nn
4881       \l__zrefclever_type_first_refbounds_seq { 3 }
4882     }
4883     \seq_item:Nn
4884     \l__zrefclever_type_first_refbounds_seq { 4 }
4885   }
4886   {
4887     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4888     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4889     \exp_not:N \group_begin:
4890     \exp_not:V \l__zrefclever_reffont_tl
4891     \__zrefclever_extract_unexp:Vvn
4892     \l__zrefclever_type_first_label_tl
4893     { \l__zrefclever_ref_property_tl } { }
4894     \exp_not:N \group_end:
4895     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4896     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4897   }
4898 }
4899 { \__zrefclever_ref_default: }
4900 }
4901 }
4902 }

```

(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`.

```
4903 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4904   {
4905     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4906     {
4907       \tl_clear:N \l__zrefclever_type_name_tl
4908       \bool_set_true:N \l__zrefclever_type_name_missing_bool
4909     }
4910     {
4911       \tl_if_eq:NnTF
4912         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4913         {
4914           \tl_clear:N \l__zrefclever_type_name_tl
4915           \bool_set_true:N \l__zrefclever_type_name_missing_bool
4916         }
4917         {
4918           % Determine whether we should use capitalization, abbreviation,
4919           % and plural.
4920           \bool_lazy_or:nnTF
4921             { \l__zrefclever_cap_bool }
4922             {
4923               \l__zrefclever_capfirst_bool &&
4924               \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4925             }
4926             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4927             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4928           % If the queue is empty, we have a singular, otherwise, plural.
4929           \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4930             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4931             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4932           \bool_lazy_and:nnTF
4933             { \l__zrefclever_abbrev_bool }
4934             {
4935               ! \int_compare_p:nNn
4936                 { \l__zrefclever_type_count_int } = { 0 } ||
4937               ! \l__zrefclever_noabbrev_first_bool
4938             }
4939             {
4940               \tl_set:NV \l__zrefclever_name_format_fallback_tl
4941                 \l__zrefclever_name_format_tl
4942               \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }

```

```

4943     }
4944     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4945
4946 % Handle number and gender nudges.
4947 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4948 {
4949     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4950     {
4951         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4952         {
4953             \msg_warning:nxx { zref-clever }
4954             { nudge-plural-when-sg }
4955             { \l__zrefclever_type_first_label_type_tl }
4956         }
4957     }
4958     {
4959         \bool_lazy_all:nT
4960         {
4961             { \l__zrefclever_nudge_comptosing_bool }
4962             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4963             {
4964                 \int_compare_p:nNn
4965                 { \l__zrefclever_label_count_int } > { 0 }
4966             }
4967         }
4968         {
4969             \msg_warning:nxx { zref-clever }
4970             { nudge-comptosing }
4971             { \l__zrefclever_type_first_label_type_tl }
4972         }
4973     }
4974     \bool_lazy_and:nnT
4975     { \l__zrefclever_nudge_gender_bool }
4976     { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4977     {
4978         \__zrefclever_get_rf_opt_seq:nxxN { gender }
4979         { \l__zrefclever_type_first_label_type_tl }
4980         { \l__zrefclever_ref_language_tl }
4981         \l__zrefclever_type_name_gender_seq
4982         \seq_if_in:NVF
4983         \l__zrefclever_type_name_gender_seq
4984         \l__zrefclever_ref_gender_tl
4985         {
4986             \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4987             {
4988                 \msg_warning:nxxxx { zref-clever }
4989                 { nudge-gender-not-declared-for-type }
4990                 { \l__zrefclever_ref_gender_tl }
4991                 { \l__zrefclever_type_first_label_type_tl }
4992                 { \l__zrefclever_ref_language_tl }
4993             }
4994             {
4995                 \msg_warning:nxxxxx { zref-clever }
4996                 { nudge-gender-mismatch }

```

```

4997         { \l__zrefclever_type_first_label_type_tl }
4998         { \l__zrefclever_ref_gender_tl }
4999         {
5000             \seq_use:Nn
5001             \l__zrefclever_type_name_gender_seq { ,~ }
5002         }
5003         { \l__zrefclever_ref_language_tl }
5004     }
5005 }
5006 }
5007 }
5008
5009 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5010 {
5011     \__zrefclever_opt_tl_get:cNF
5012     {
5013         \__zrefclever_opt_varname_type:een
5014         { \l__zrefclever_type_first_label_type_tl }
5015         { \l__zrefclever_name_format_tl }
5016         { tl }
5017     }
5018     \l__zrefclever_type_name_tl
5019     {
5020         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5021         {
5022             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5023             \tl_put_left:NV \l__zrefclever_name_format_tl
5024             \l__zrefclever_ref_decl_case_tl
5025         }
5026         \__zrefclever_opt_tl_get:cNF
5027         {
5028             \__zrefclever_opt_varname_lang_type:een
5029             { \l__zrefclever_ref_language_tl }
5030             { \l__zrefclever_type_first_label_type_tl }
5031             { \l__zrefclever_name_format_tl }
5032             { tl }
5033         }
5034         \l__zrefclever_type_name_tl
5035         {
5036             \tl_clear:N \l__zrefclever_type_name_tl
5037             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5038             \msg_warning:nxxx { zref-clever } { missing-name }
5039             { \l__zrefclever_name_format_tl }
5040             { \l__zrefclever_type_first_label_type_tl }
5041         }
5042     }
5043 }
5044 {
5045     \__zrefclever_opt_tl_get:cNF
5046     {
5047         \__zrefclever_opt_varname_type:een
5048         { \l__zrefclever_type_first_label_type_tl }
5049         { \l__zrefclever_name_format_tl }
5050         { tl }

```



```

5051 }
5052 \l__zrefclever_type_name_tl
5053 {
5054   \__zrefclever_opt_tl_get:cNF
5055   {
5056     \__zrefclever_opt_varname_type:een
5057     { \l__zrefclever_type_first_label_type_tl }
5058     { \l__zrefclever_name_format_fallback_tl }
5059     { tl }
5060   }
5061   \l__zrefclever_type_name_tl
5062   {
5063     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5064     {
5065       \tl_put_left:Nn
5066         \l__zrefclever_name_format_tl { - }
5067       \tl_put_left:NV \l__zrefclever_name_format_tl
5068         \l__zrefclever_ref_decl_case_tl
5069       \tl_put_left:Nn
5070         \l__zrefclever_name_format_fallback_tl { - }
5071       \tl_put_left:NV
5072         \l__zrefclever_name_format_fallback_tl
5073         \l__zrefclever_ref_decl_case_tl
5074     }
5075     \__zrefclever_opt_tl_get:cNF
5076     {
5077       \__zrefclever_opt_varname_lang_type:een
5078       { \l__zrefclever_ref_language_tl }
5079       { \l__zrefclever_type_first_label_type_tl }
5080       { \l__zrefclever_name_format_tl }
5081       { tl }
5082     }
5083     \l__zrefclever_type_name_tl
5084     {
5085       \__zrefclever_opt_tl_get:cNF
5086       {
5087         \__zrefclever_opt_varname_lang_type:een
5088         { \l__zrefclever_ref_language_tl }
5089         { \l__zrefclever_type_first_label_type_tl }
5090         { \l__zrefclever_name_format_fallback_tl }
5091         { tl }
5092       }
5093       \l__zrefclever_type_name_tl
5094       {
5095         \tl_clear:N \l__zrefclever_type_name_tl
5096         \bool_set_true:N
5097           \l__zrefclever_type_name_missing_bool
5098         \msg_warning:nxxx { zref-clever }
5099           { missing-name }
5100         { \l__zrefclever_name_format_tl }
5101         { \l__zrefclever_type_first_label_type_tl }
5102       }
5103     }
5104   }

```

```

5105         }
5106     }
5107 }
5108 }
5109
5110 % Signal whether the type name is to be included in the hyperlink or not.
5111 \bool_lazy_any:nTF
5112 {
5113     { ! \l__zrefclever_hyperlink_bool }
5114     { \l__zrefclever_link_star_bool }
5115     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5116     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5117 }
5118 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5119 {
5120     \bool_lazy_any:nTF
5121     {
5122         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5123         {
5124             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5125             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5126         }
5127         {
5128             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5129             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5130             \l__zrefclever_typeset_last_bool &&
5131             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5132         }
5133     }
5134     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5135     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5136 }
5137 }

```

(End of definition for __zrefclever_type_name_setup:.)

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

```

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

```

(End of definition for __zrefclever_hyperlink:nnn.)

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

```

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

```

(End of definition for __zrefclever_extract_url_unexp:n.)

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

```

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

5155 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5156 {
5157   \exp_args:Nxx \tl_if_eq:nnT
5158   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5159   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5160   {
5161     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5162     {
5163       \exp_args:Nxx \tl_if_eq:nnT
5164       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5165       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5166       {
5167         \int_compare:nNnTF
5168         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5169         =
5170         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5171         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5172         {
5173           \int_compare:nNnT
5174           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5175           =
5176           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5177           {
5178             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5179             \bool_set_true:N \l__zrefclever_next_is_same_bool
5180           }
5181         }
5182       }
5183     }
5184   }
5185   \exp_args:Nxx \tl_if_eq:nnT

```

```

5186 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5187 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5188 {
5189   \exp_args:Nxx \tl_if_eq:nnT
5190   { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5191   { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5192   {
5193     \int_compare:nNnTF
5194     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5195     =
5196     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5197     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5198     {
5199       \int_compare:nNnT
5200       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5201       =
5202       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5203       {

```

If `zc@counters` are equal, `zc@enclvals` are equal, and `zc@cntvals` are equal, but the references themselves are different, this means that `\@currentlabel` has somehow been set manually (e.g. by an `amsmath`'s `\tag`), in which case we have no idea what's in there, and we should not even consider this is still a range. If they are equal, though, of course it is a range, and it is the same.

```

5204   \exp_args:Nxx \tl_if_eq:nnT
5205   {
5206     \_zrefclever_extract_unexp:nvn {#1}
5207     { \l__zrefclever_ref_property_tl } { }
5208   }
5209   {
5210     \_zrefclever_extract_unexp:nvn {#2}
5211     { \l__zrefclever_ref_property_tl } { }
5212   }
5213   {
5214     \bool_set_true:N
5215     \l__zrefclever_next_maybe_range_bool
5216     \bool_set_true:N
5217     \l__zrefclever_next_is_same_bool
5218   }
5219 }
5220 }
5221 }
5222 }
5223 }
5224 }
5225 }

```

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

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

```

\__zrefclever_get_rf_opt_tl:nnnN
\__zrefclever_get_rf_opt_tl:nnnN {<option>}
  {<ref type>} {<language>} {<tl variable>}
5226 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5227 {
5228   % First attempt: general options.
5229   \__zrefclever_opt_tl_get:cNF
5230   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5231   #4
5232   {
5233     % If not found, try type specific options.
5234     \__zrefclever_opt_tl_get:cNF
5235     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5236     #4
5237     {
5238       % If not found, try type- and language-specific.
5239       \__zrefclever_opt_tl_get:cNF
5240       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5241       #4
5242       {
5243         % If not found, try language-specific default.
5244         \__zrefclever_opt_tl_get:cNF
5245         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5246         #4
5247         {
5248           % If not found, try fallback.
5249           \__zrefclever_opt_tl_get:cNF
5250           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5251           #4
5252           { \tl_clear:N #4 }
5253         }
5254       }
5255     }
5256   }
5257 }
5258 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { 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>}
5259 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5260 {
5261   % First attempt: general options.
5262   \__zrefclever_opt_seq_get:cNF
5263   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5264   #4
5265   {
5266     % If not found, try type specific options.
5267     \__zrefclever_opt_seq_get:cNF
5268     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5269     #4
5270     {
5271       % If not found, try type- and language-specific.
5272       \__zrefclever_opt_seq_get:cNF

```

```

5273         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5274         #4
5275         {
5276             % If not found, try language-specific default.
5277             \__zrefclever_opt_seq_get:cNF
5278             { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5279             #4
5280             {
5281                 % If not found, try fallback.
5282                 \__zrefclever_opt_seq_get:cNF
5283                 { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5284                 #4
5285                 { \seq_clear:N #4 }
5286             }
5287         }
5288     }
5289 }
5290 }
5291 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { 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>}
5292 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5293 {
5294     % First attempt: general options.
5295     \__zrefclever_opt_bool_get:cNF
5296     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5297     #5
5298     {
5299         % If not found, try type specific options.
5300         \__zrefclever_opt_bool_get:cNF
5301         { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5302         #5
5303         {
5304             % If not found, try type- and language-specific.
5305             \__zrefclever_opt_bool_get:cNF
5306             { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5307             #5
5308             {
5309                 % If not found, try language-specific default.
5310                 \__zrefclever_opt_bool_get:cNF
5311                 { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5312                 #5
5313                 {
5314                     % If not found, try fallback.
5315                     \__zrefclever_opt_bool_get:cNF
5316                     { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5317                     #5
5318                     { \use:c { bool_set_ #2 :N } #5 }
5319                 }
5320             }
5321         }

```

```

5322     }
5323   }
5324 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nxxxN }

```

(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.

```

5325 \__zrefclever_compat_module:nn { appendix }
5326 {
5327   \AddToHook { cmd / appendix / before }
5328   {
5329     \__zrefclever_zcsetup:n
5330     {
5331       countertype =
5332       {
5333         chapter      = appendix ,
5334         section      = appendix ,
5335         subsection   = appendix ,
5336         subsubsection = appendix ,
5337         paragraph    = appendix ,
5338         subparagraph = appendix ,
5339       }
5340     }
5341   }
5342 }

```

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.

```

5343 \__zrefclever_compat_module:nn { appendices }
5344 {
5345   \__zrefclever_if_package_loaded:nT { appendix }
5346   {
5347     \newcounter { zc@appendix }
5348     \newcounter { zc@save@appendix }
5349     \setcounter { zc@appendix } { 0 }
5350     \setcounter { zc@save@appendix } { 0 }
5351     \cs_if_exist:cTF { chapter }
5352     {
5353       \__zrefclever_zcsetup:n
5354       { counterresetby = { chapter = zc@appendix } }
5355     }
5356     {
5357       \cs_if_exist:cT { section }
5358       {
5359         \__zrefclever_zcsetup:n
5360         { counterresetby = { section = zc@appendix } }
5361       }
5362     }
5363   \AddToHook { env / appendices / begin }
5364   {
5365     \stepcounter { zc@save@appendix }
5366     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5367     \__zrefclever_zcsetup:n
5368     {
5369       countertype =
5370       {
5371         chapter      = appendix ,
5372         section      = appendix ,
5373         subsection   = appendix ,
5374         subsubsection = appendix ,
5375         paragraph    = appendix ,
5376         subparagraph = appendix ,
5377       }
5378     }
5379   }
5380   \AddToHook { env / appendices / end }
5381   { \setcounter { zc@appendix } { 0 } }
5382   \AddToHook { cmd / appendix / before }
5383   {
5384     \stepcounter { zc@save@appendix }
5385     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5386   }

```



```

5387 \AddToHook { env / subappendices / begin }
5388 {
5389   \__zrefclever_zcsetup:n
5390   {
5391     countertype =
5392     {
5393       section      = appendix ,
5394       subsection   = appendix ,
5395       subsubsection = appendix ,
5396       paragraph    = appendix ,
5397       subparagraph = appendix ,
5398     } ,
5399   }
5400 }
5401 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5402 }
5403 }

```

9.3 memoir

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

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

Also, note that memoir’s appendix features “emulates” the appendix package, hence the corresponding compatibility module is loaded for memoir even if that package is not itself loaded. The same is true for the \appendix command module, since it is also defined.

```

5404 \__zrefclever_compat_module:nn { memoir }
5405 {
5406   \__zrefclever_if_class_loaded:nT { memoir }
5407   {

```

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.

```

5408   \__zrefclever_zcsetup:n
5409   {
5410     countertype =
5411     {
5412       subfigure = figure ,
5413       subtable  = table ,
5414       poemline  = line ,
5415     } ,

```

```

5416         counterresetby =
5417         {
5418             subfigure = figure ,
5419             subtable = table ,
5420         } ,
5421     }

```

Support for subcaption references.

```

5422     \zref@newprop { subcaption }
5423     { \cs_if_exist_use:c { @thesub \@capytype } }
5424     \AddToHook{ memoir/subcaption/aftercounter }
5425     { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

Support for \sidefootnote and \pagenote.

```

5426     \__zrefclever_zcsetup:n
5427     {
5428         countertype =
5429         {
5430             sidefootnote = footnote ,
5431             pagenote = endnote ,
5432         } ,
5433     }
5434     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5435 }
5436 }

```

9.4 amsmath

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

```

5437 \__zrefclever_compat_module:nn { amsmath }
5438 {
5439     \__zrefclever_if_package_loaded:nT { amsmath }
5440 }

```

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).

```

5441     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5442     \AddToHook { env / subequations / begin }
5443     {
5444         \__zrefclever_zcsetup:x
5445         {
5446             counterresetby =
5447             {

```

```

5448         parentequation =
5449             \_zrefclever_counter_reset_by:n { equation } ,
5450         equation = parentequation ,
5451     } ,
5452     currentcounter = parentequation ,
5453     countertype = { parentequation = equation } ,
5454 }
5455 \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5456 }

```

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.

```

5457     \zref@newprop { subeq } { \alph { equation } }
5458     \clist_map_inline:nn
5459     {
5460         equation ,
5461         equation* ,
5462         align ,
5463         align* ,
5464         alignat ,
5465         alignat* ,
5466         flalign ,
5467         flalign* ,
5468         xalignat ,
5469         xalignat* ,
5470         gather ,
5471         gather* ,
5472         multiline ,
5473         multiline* ,
5474     }
5475     {
5476     \AddToHook { env / #1 / begin }
5477     {
5478         \_zrefclever_zcsetup:n { currentcounter = equation }
5479         \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5480             { \zref@localaddprop \ZREF@mainlist { subeq } }
5481     }
5482     }
5483     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5484 }
5485 }

```

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.

```
5486 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5487 \__zrefclever_compat_module:nn { mathtools }
5488 {
5489   \__zrefclever_if_package_loaded:nT { mathtools }
5490   {
5491     \MH_if_boolean:nT { show_only_refs }
5492     {
5493       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5494       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5495       {
5496         \@bsphack
5497         \seq_map_inline:Nn #1
5498         {
5499           \exp_args:Nx \tl_if_eq:nnTF
5500           { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5501           { equation }
5502           {
5503             \protected@write \@auxout { }
5504             { \string \MT@newlabel {##1} }
5505           }
5506           {
5507             \exp_args:Nx \tl_if_eq:nnT
5508             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5509             { parentequation }
5510             {
5511               \protected@write \@auxout { }
5512               { \string \MT@newlabel {##1} }
5513             }
5514           }
5515         }
5516         \@esphack
5517       }
5518       \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5519     }
5520   }
5521 }
```

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.

```

5522 \__zrefclever_compat_module:nn { breqn }
5523 {
5524   \__zrefclever_if_package_loaded:nT { breqn }
5525   {

```

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>).

```

5526     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5527     \AddToHook { env / dgroup / begin }
5528     {
5529       \__zrefclever_zcsetup:x
5530       {
5531         counterresetby =
5532         {
5533           parentequation =
5534             \__zrefclever_counter_reset_by:n { equation } ,
5535           equation = parentequation ,
5536         } ,
5537         currentcounter = parentequation ,
5538         countertype = { parentequation = equation } ,
5539       }
5540     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5541     }
5542     \zref@ifpropundefined { subeq }
5543     { \zref@newprop { subeq } { \alph { equation } } }
5544     { }
5545     \clist_map_inline:nn
5546     {
5547       dmath ,
5548       dseries ,
5549       darray ,
5550     }
5551     {
5552       \AddToHook { env / #1 / begin }
5553       {
5554         \__zrefclever_zcsetup:n { currentcounter = equation }
5555         \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5556           { \zref@localaddprop \ZREF@mainlist { subeq } }
5557       }
5558     }
5559     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5560   }
5561 }

```

9.7 listings

```

5562 \__zrefclever_compat_module:nn { listings }
5563 {
5564   \__zrefclever_if_package_loaded:nT { listings }
5565   {

```

```

5566     \__zrefclever_zcsetup:n
5567     {
5568         countertype =
5569         {
5570             lstlisting = listing ,
5571             lstnumber = line ,
5572         } ,
5573         counterresetby = { lstnumber = lstlisting } ,
5574     }

```

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.

```

5575     \lst@AddToHook { Init }
5576     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5577     \msg_info:nnn { zref-clever } { compat-package } { listings }
5578     }
5579 }

```

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.

```

5580 \__zrefclever_compat_module:nn { enumitem }
5581 {
5582     \__zrefclever_if_package_loaded:nT { enumitem }
5583     {
5584         \int_set:Nn \l_tmpa_int { 5 }
5585         \bool_while_do:nn
5586         {
5587             \cs_if_exist_p:c
5588             { c@ enum \int_to_roman:n { \l_tmpa_int } }
5589         }
5590         {
5591             \__zrefclever_zcsetup:x
5592             {
5593                 counterresetby =

```

```

5594         {
5595             enum \int_to_roman:n { \l_tmpa_int } =
5596             enum \int_to_roman:n { \l_tmpa_int - 1 }
5597         } ,
5598         countertype =
5599         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5600     }
5601     \int_incr:N \l_tmpa_int
5602 }
5603 \int_compare:nNnT { \l_tmpa_int } > { 5 }
5604 { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5605 }
5606 }

```

9.9 subcaption

```

5607 \__zrefclever_compat_module:nn { subcaption }
5608 {
5609     \__zrefclever_if_package_loaded:nT { subcaption }
5610     {
5611         \__zrefclever_zcsetup:n
5612         {
5613             countertype =
5614             {
5615                 subfigure = figure ,
5616                 subtable = table ,
5617             } ,
5618             counterresetby =
5619             {
5620                 subfigure = figure ,
5621                 subtable = table ,
5622             } ,
5623         }
5624     }
5625 }

```

Support for subref reference.

```

5624 \zref@newprop { subref }
5625 { \cs_if_exist_use:c { thesub \@capttype } }
5626 \tl_put_right:Nn \caption@subtypehook
5627 { \zref@localaddprop \ZREF@mainlist { subref } }
5628 }
5629 }

```

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.

```

5630 \__zrefclever_compat_module:nn { subfig }
5631 {
5632     \__zrefclever_if_package_loaded:nT { subfig }
5633     {
5634         \__zrefclever_zcsetup:n
5635         {
5636             countertype =
5637             {
5638                 subfigure = figure ,

```

```

5639         subtable = table ,
5640     } ,
5641     counterresetby =
5642     {
5643         subfigure = figure ,
5644         subtable = table ,
5645     } ,
5646 }
5647 }
5648 }
5649 </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” `rebound`s options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that's the point where we know from `babel` or `polyglossia` the `\languagename`. But we also don't want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

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

10.2 English

English language file has been initially provided by the author.

```
5650 ⟨*package⟩
5651 \zcDeclareLanguage { english }
5652 \zcDeclareLanguageAlias { american } { english }
5653 \zcDeclareLanguageAlias { australian } { english }
5654 \zcDeclareLanguageAlias { british } { english }
5655 \zcDeclareLanguageAlias { canadian } { english }
5656 \zcDeclareLanguageAlias { newzealand } { english }
5657 \zcDeclareLanguageAlias { UKenglish } { english }
5658 \zcDeclareLanguageAlias { USenglish } { english }
5659 ⟨/package⟩

5660 ⟨*lang-english⟩

5661 namesep = {\nobreakspace} ,
5662 pairsep = {~and\nobreakspace} ,
5663 listsep = {,~} ,
5664 lastsep = {~and\nobreakspace} ,
5665 tpairsep = {~and\nobreakspace} ,
5666 tlistsep = {,~} ,
5667 tlastsep = {,~and\nobreakspace} ,
5668 notesep = {~} ,
5669 rangesep = {~to\nobreakspace} ,
5670
5671 type = book ,
5672 Name-sg = Book ,
5673 name-sg = book ,
5674 Name-pl = Books ,
5675 name-pl = books ,
5676
5677 type = part ,
5678 Name-sg = Part ,
5679 name-sg = part ,
5680 Name-pl = Parts ,
5681 name-pl = parts ,
5682
5683 type = chapter ,
5684 Name-sg = Chapter ,
5685 name-sg = chapter ,
5686 Name-pl = Chapters ,
5687 name-pl = chapters ,
5688
5689 type = section ,
5690 Name-sg = Section ,
5691 name-sg = section ,
5692 Name-pl = Sections ,
5693 name-pl = sections ,
5694
5695 type = paragraph ,
5696 Name-sg = Paragraph ,
5697 name-sg = paragraph ,
5698 Name-pl = Paragraphs ,
5699 name-pl = paragraphs ,
```

```

5700 Name-sg-ab = Par. ,
5701 name-sg-ab = par. ,
5702 Name-pl-ab = Par. ,
5703 name-pl-ab = par. ,
5704
5705 type = appendix ,
5706 Name-sg = Appendix ,
5707 name-sg = appendix ,
5708 Name-pl = Appendices ,
5709 name-pl = appendices ,
5710
5711 type = page ,
5712 Name-sg = Page ,
5713 name-sg = page ,
5714 Name-pl = Pages ,
5715 name-pl = pages ,
5716 rangeseq = {\textendash} ,
5717 rangetopair = false ,
5718
5719 type = line ,
5720 Name-sg = Line ,
5721 name-sg = line ,
5722 Name-pl = Lines ,
5723 name-pl = lines ,
5724
5725 type = figure ,
5726 Name-sg = Figure ,
5727 name-sg = figure ,
5728 Name-pl = Figures ,
5729 name-pl = figures ,
5730 Name-sg-ab = Fig. ,
5731 name-sg-ab = fig. ,
5732 Name-pl-ab = Figs. ,
5733 name-pl-ab = figs. ,
5734
5735 type = table ,
5736 Name-sg = Table ,
5737 name-sg = table ,
5738 Name-pl = Tables ,
5739 name-pl = tables ,
5740
5741 type = item ,
5742 Name-sg = Item ,
5743 name-sg = item ,
5744 Name-pl = Items ,
5745 name-pl = items ,
5746
5747 type = footnote ,
5748 Name-sg = Footnote ,
5749 name-sg = footnote ,
5750 Name-pl = Footnotes ,
5751 name-pl = footnotes ,
5752
5753 type = endnote ,

```

```

5754 Name-sg = Note ,
5755 name-sg = note ,
5756 Name-pl = Notes ,
5757 name-pl = notes ,
5758
5759 type = note ,
5760 Name-sg = Note ,
5761 name-sg = note ,
5762 Name-pl = Notes ,
5763 name-pl = notes ,
5764
5765 type = equation ,
5766 Name-sg = Equation ,
5767 name-sg = equation ,
5768 Name-pl = Equations ,
5769 name-pl = equations ,
5770 Name-sg-ab = Eq. ,
5771 name-sg-ab = eq. ,
5772 Name-pl-ab = Eqs. ,
5773 name-pl-ab = eqs. ,
5774 refbounds-first-sg = {,(,)},
5775 refbounds = {(,,)} ,
5776
5777 type = theorem ,
5778 Name-sg = Theorem ,
5779 name-sg = theorem ,
5780 Name-pl = Theorems ,
5781 name-pl = theorems ,
5782
5783 type = lemma ,
5784 Name-sg = Lemma ,
5785 name-sg = lemma ,
5786 Name-pl = Lemmas ,
5787 name-pl = lemmas ,
5788
5789 type = corollary ,
5790 Name-sg = Corollary ,
5791 name-sg = corollary ,
5792 Name-pl = Corollaries ,
5793 name-pl = corollaries ,
5794
5795 type = proposition ,
5796 Name-sg = Proposition ,
5797 name-sg = proposition ,
5798 Name-pl = Propositions ,
5799 name-pl = propositions ,
5800
5801 type = definition ,
5802 Name-sg = Definition ,
5803 name-sg = definition ,
5804 Name-pl = Definitions ,
5805 name-pl = definitions ,
5806
5807 type = proof ,

```

```

5808   Name-sg = Proof ,
5809   name-sg = proof ,
5810   Name-pl = Proofs ,
5811   name-pl = proofs ,
5812
5813   type = result ,
5814   Name-sg = Result ,
5815   name-sg = result ,
5816   Name-pl = Results ,
5817   name-pl = results ,
5818
5819   type = remark ,
5820   Name-sg = Remark ,
5821   name-sg = remark ,
5822   Name-pl = Remarks ,
5823   name-pl = remarks ,
5824
5825   type = example ,
5826   Name-sg = Example ,
5827   name-sg = example ,
5828   Name-pl = Examples ,
5829   name-pl = examples ,
5830
5831   type = algorithm ,
5832   Name-sg = Algorithm ,
5833   name-sg = algorithm ,
5834   Name-pl = Algorithms ,
5835   name-pl = algorithms ,
5836
5837   type = listing ,
5838   Name-sg = Listing ,
5839   name-sg = listing ,
5840   Name-pl = Listings ,
5841   name-pl = listings ,
5842
5843   type = exercise ,
5844   Name-sg = Exercise ,
5845   name-sg = exercise ,
5846   Name-pl = Exercises ,
5847   name-pl = exercises ,
5848
5849   type = solution ,
5850   Name-sg = Solution ,
5851   name-sg = solution ,
5852   Name-pl = Solutions ,
5853   name-pl = solutions ,
5854 </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`.

```

5855 <*package>
5856 \zcDeclareLanguage
5857 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5858 { german }
5859 \zcDeclareLanguageAlias { ngerman      } { german }
5860 \zcDeclareLanguageAlias { austrian     } { german }
5861 \zcDeclareLanguageAlias { naustrian    } { german }
5862 \zcDeclareLanguageAlias { swissgerman  } { german }
5863 \zcDeclareLanguageAlias { nswissgerman } { german }
5864 </package>

5865 <*lang-german>

5866 namesep = {\nobreakspace} ,
5867 pairsep  = {\~und\nobreakspace} ,
5868 listsep  = { , ~ } ,
5869 lastsep  = {\~und\nobreakspace} ,
5870 tpairsep = {\~und\nobreakspace} ,
5871 tlistsep = { , ~ } ,
5872 tlastsep = {\~und\nobreakspace} ,
5873 notesep  = { ~ } ,
5874 rangesep = {\~bis\nobreakspace} ,
5875
5876 type = book ,
5877 gender = n ,
5878 case = N ,
5879     Name-sg = Buch ,
5880     Name-pl = Bücher ,
5881 case = A ,
5882     Name-sg = Buch ,
5883     Name-pl = Bücher ,
5884 case = D ,
5885     Name-sg = Buch ,
5886     Name-pl = Büchern ,
5887 case = G ,
5888     Name-sg = Buches ,
5889     Name-pl = Bücher ,
5890
5891 type = part ,
5892 gender = m ,
5893 case = N ,
5894     Name-sg = Teil ,
5895     Name-pl = Teile ,
5896 case = A ,
5897     Name-sg = Teil ,
5898     Name-pl = Teile ,
5899 case = D ,
5900     Name-sg = Teil ,
5901     Name-pl = Teilen ,
5902 case = G ,
5903     Name-sg = Teiles ,
5904     Name-pl = Teile ,
5905
5906 type = chapter ,
5907 gender = n ,

```

```

5908 case = N ,
5909     Name-sg = Kapitel ,
5910     Name-pl = Kapitel ,
5911 case = A ,
5912     Name-sg = Kapitel ,
5913     Name-pl = Kapitel ,
5914 case = D ,
5915     Name-sg = Kapitel ,
5916     Name-pl = Kapiteln ,
5917 case = G ,
5918     Name-sg = Kapitels ,
5919     Name-pl = Kapitel ,
5920
5921 type = section ,
5922     gender = m ,
5923     case = N ,
5924         Name-sg = Abschnitt ,
5925         Name-pl = Abschnitte ,
5926     case = A ,
5927         Name-sg = Abschnitt ,
5928         Name-pl = Abschnitte ,
5929     case = D ,
5930         Name-sg = Abschnitt ,
5931         Name-pl = Abschnitten ,
5932     case = G ,
5933         Name-sg = Abschnitts ,
5934         Name-pl = Abschnitte ,
5935
5936 type = paragraph ,
5937     gender = m ,
5938     case = N ,
5939         Name-sg = Absatz ,
5940         Name-pl = Absätze ,
5941     case = A ,
5942         Name-sg = Absatz ,
5943         Name-pl = Absätze ,
5944     case = D ,
5945         Name-sg = Absatz ,
5946         Name-pl = Absätzen ,
5947     case = G ,
5948         Name-sg = Absatzes ,
5949         Name-pl = Absätze ,
5950
5951 type = appendix ,
5952     gender = m ,
5953     case = N ,
5954         Name-sg = Anhang ,
5955         Name-pl = Anhänge ,
5956     case = A ,
5957         Name-sg = Anhang ,
5958         Name-pl = Anhänge ,
5959     case = D ,
5960         Name-sg = Anhang ,
5961         Name-pl = Anhängen ,

```



```

5962 case = G ,
5963     Name-sg = Anhangs ,
5964     Name-pl = Anhänge ,
5965
5966 type = page ,
5967     gender = f ,
5968     case = N ,
5969     Name-sg = Seite ,
5970     Name-pl = Seiten ,
5971     case = A ,
5972     Name-sg = Seite ,
5973     Name-pl = Seiten ,
5974     case = D ,
5975     Name-sg = Seite ,
5976     Name-pl = Seiten ,
5977     case = G ,
5978     Name-sg = Seite ,
5979     Name-pl = Seiten ,
5980     rangesep = {\textendash} ,
5981     rangetopair = false ,
5982
5983 type = line ,
5984     gender = f ,
5985     case = N ,
5986     Name-sg = Zeile ,
5987     Name-pl = Zeilen ,
5988     case = A ,
5989     Name-sg = Zeile ,
5990     Name-pl = Zeilen ,
5991     case = D ,
5992     Name-sg = Zeile ,
5993     Name-pl = Zeilen ,
5994     case = G ,
5995     Name-sg = Zeile ,
5996     Name-pl = Zeilen ,
5997
5998 type = figure ,
5999     gender = f ,
6000     case = N ,
6001     Name-sg = Abbildung ,
6002     Name-pl = Abbildungen ,
6003     Name-sg-ab = Abb. ,
6004     Name-pl-ab = Abb. ,
6005     case = A ,
6006     Name-sg = Abbildung ,
6007     Name-pl = Abbildungen ,
6008     Name-sg-ab = Abb. ,
6009     Name-pl-ab = Abb. ,
6010     case = D ,
6011     Name-sg = Abbildung ,
6012     Name-pl = Abbildungen ,
6013     Name-sg-ab = Abb. ,
6014     Name-pl-ab = Abb. ,
6015     case = G ,

```

```

6016     Name-sg = Abbildung ,
6017     Name-pl = Abbildungen ,
6018     Name-sg-ab = Abb. ,
6019     Name-pl-ab = Abb. ,
6020
6021 type = table ,
6022     gender = f ,
6023     case = N ,
6024     Name-sg = Tabelle ,
6025     Name-pl = Tabellen ,
6026     case = A ,
6027     Name-sg = Tabelle ,
6028     Name-pl = Tabellen ,
6029     case = D ,
6030     Name-sg = Tabelle ,
6031     Name-pl = Tabellen ,
6032     case = G ,
6033     Name-sg = Tabelle ,
6034     Name-pl = Tabellen ,
6035
6036 type = item ,
6037     gender = m ,
6038     case = N ,
6039     Name-sg = Punkt ,
6040     Name-pl = Punkte ,
6041     case = A ,
6042     Name-sg = Punkt ,
6043     Name-pl = Punkte ,
6044     case = D ,
6045     Name-sg = Punkt ,
6046     Name-pl = Punkten ,
6047     case = G ,
6048     Name-sg = Punktes ,
6049     Name-pl = Punkte ,
6050
6051 type = footnote ,
6052     gender = f ,
6053     case = N ,
6054     Name-sg = Fußnote ,
6055     Name-pl = Fußnoten ,
6056     case = A ,
6057     Name-sg = Fußnote ,
6058     Name-pl = Fußnoten ,
6059     case = D ,
6060     Name-sg = Fußnote ,
6061     Name-pl = Fußnoten ,
6062     case = G ,
6063     Name-sg = Fußnote ,
6064     Name-pl = Fußnoten ,
6065
6066 type = endnote ,
6067     gender = f ,
6068     case = N ,
6069     Name-sg = Endnote ,

```

```

6070     Name-pl = Endnoten ,
6071     case = A ,
6072     Name-sg = Endnote ,
6073     Name-pl = Endnoten ,
6074     case = D ,
6075     Name-sg = Endnote ,
6076     Name-pl = Endnoten ,
6077     case = G ,
6078     Name-sg = Endnote ,
6079     Name-pl = Endnoten ,
6080
6081 type = note ,
6082     gender = f ,
6083     case = N ,
6084     Name-sg = Anmerkung ,
6085     Name-pl = Anmerkungen ,
6086     case = A ,
6087     Name-sg = Anmerkung ,
6088     Name-pl = Anmerkungen ,
6089     case = D ,
6090     Name-sg = Anmerkung ,
6091     Name-pl = Anmerkungen ,
6092     case = G ,
6093     Name-sg = Anmerkung ,
6094     Name-pl = Anmerkungen ,
6095
6096 type = equation ,
6097     gender = f ,
6098     case = N ,
6099     Name-sg = Gleichung ,
6100     Name-pl = Gleichungen ,
6101     case = A ,
6102     Name-sg = Gleichung ,
6103     Name-pl = Gleichungen ,
6104     case = D ,
6105     Name-sg = Gleichung ,
6106     Name-pl = Gleichungen ,
6107     case = G ,
6108     Name-sg = Gleichung ,
6109     Name-pl = Gleichungen ,
6110     refbounds-first-sg = {,(,)},
6111     refbounds = {(,,)},
6112
6113 type = theorem ,
6114     gender = n ,
6115     case = N ,
6116     Name-sg = Theorem ,
6117     Name-pl = Theoreme ,
6118     case = A ,
6119     Name-sg = Theorem ,
6120     Name-pl = Theoreme ,
6121     case = D ,
6122     Name-sg = Theorem ,
6123     Name-pl = Theoremen ,

```

```

6124     case = G ,
6125     Name-sg = Theorems ,
6126     Name-pl = Theoreme ,
6127
6128 type = lemma ,
6129     gender = n ,
6130     case = N ,
6131     Name-sg = Lemma ,
6132     Name-pl = Lemmata ,
6133     case = A ,
6134     Name-sg = Lemma ,
6135     Name-pl = Lemmata ,
6136     case = D ,
6137     Name-sg = Lemma ,
6138     Name-pl = Lemmata ,
6139     case = G ,
6140     Name-sg = Lemmas ,
6141     Name-pl = Lemmata ,
6142
6143 type = corollary ,
6144     gender = n ,
6145     case = N ,
6146     Name-sg = Korollar ,
6147     Name-pl = Korollare ,
6148     case = A ,
6149     Name-sg = Korollar ,
6150     Name-pl = Korollare ,
6151     case = D ,
6152     Name-sg = Korollar ,
6153     Name-pl = Korollaren ,
6154     case = G ,
6155     Name-sg = Korollars ,
6156     Name-pl = Korollare ,
6157
6158 type = proposition ,
6159     gender = m ,
6160     case = N ,
6161     Name-sg = Satz ,
6162     Name-pl = Sätze ,
6163     case = A ,
6164     Name-sg = Satz ,
6165     Name-pl = Sätze ,
6166     case = D ,
6167     Name-sg = Satz ,
6168     Name-pl = Sätzen ,
6169     case = G ,
6170     Name-sg = Satzes ,
6171     Name-pl = Sätze ,
6172
6173 type = definition ,
6174     gender = f ,
6175     case = N ,
6176     Name-sg = Definition ,
6177     Name-pl = Definitionen ,

```

```

6178 case = A ,
6179     Name-sg = Definition ,
6180     Name-pl = Definitionen ,
6181 case = D ,
6182     Name-sg = Definition ,
6183     Name-pl = Definitionen ,
6184 case = G ,
6185     Name-sg = Definition ,
6186     Name-pl = Definitionen ,
6187
6188 type = proof ,
6189     gender = m ,
6190     case = N ,
6191         Name-sg = Beweis ,
6192         Name-pl = Beweise ,
6193     case = A ,
6194         Name-sg = Beweis ,
6195         Name-pl = Beweise ,
6196     case = D ,
6197         Name-sg = Beweis ,
6198         Name-pl = Beweisen ,
6199     case = G ,
6200         Name-sg = Beweises ,
6201         Name-pl = Beweise ,
6202
6203 type = result ,
6204     gender = n ,
6205     case = N ,
6206         Name-sg = Ergebnis ,
6207         Name-pl = Ergebnisse ,
6208     case = A ,
6209         Name-sg = Ergebnis ,
6210         Name-pl = Ergebnisse ,
6211     case = D ,
6212         Name-sg = Ergebnis ,
6213         Name-pl = Ergebnissen ,
6214     case = G ,
6215         Name-sg = Ergebnisses ,
6216         Name-pl = Ergebnisse ,
6217
6218 type = remark ,
6219     gender = f ,
6220     case = N ,
6221         Name-sg = Bemerkung ,
6222         Name-pl = Bemerkungen ,
6223     case = A ,
6224         Name-sg = Bemerkung ,
6225         Name-pl = Bemerkungen ,
6226     case = D ,
6227         Name-sg = Bemerkung ,
6228         Name-pl = Bemerkungen ,
6229     case = G ,
6230         Name-sg = Bemerkung ,
6231         Name-pl = Bemerkungen ,

```

```

6232
6233 type = example ,
6234     gender = n ,
6235     case = N ,
6236         Name-sg = Beispiel ,
6237         Name-pl = Beispiele ,
6238     case = A ,
6239         Name-sg = Beispiel ,
6240         Name-pl = Beispiele ,
6241     case = D ,
6242         Name-sg = Beispiel ,
6243         Name-pl = Beispielen ,
6244     case = G ,
6245         Name-sg = Beispiels ,
6246         Name-pl = Beispiele ,
6247
6248 type = algorithm ,
6249     gender = m ,
6250     case = N ,
6251         Name-sg = Algorithmus ,
6252         Name-pl = Algorithmen ,
6253     case = A ,
6254         Name-sg = Algorithmus ,
6255         Name-pl = Algorithmen ,
6256     case = D ,
6257         Name-sg = Algorithmus ,
6258         Name-pl = Algorithmen ,
6259     case = G ,
6260         Name-sg = Algorithmus ,
6261         Name-pl = Algorithmen ,
6262
6263 type = listing ,
6264     gender = n ,
6265     case = N ,
6266         Name-sg = Listing ,
6267         Name-pl = Listings ,
6268     case = A ,
6269         Name-sg = Listing ,
6270         Name-pl = Listings ,
6271     case = D ,
6272         Name-sg = Listing ,
6273         Name-pl = Listings ,
6274     case = G ,
6275         Name-sg = Listings ,
6276         Name-pl = Listings ,
6277
6278 type = exercise ,
6279     gender = f ,
6280     case = N ,
6281         Name-sg = Übungsaufgabe ,
6282         Name-pl = Übungsaufgaben ,
6283     case = A ,
6284         Name-sg = Übungsaufgabe ,
6285         Name-pl = Übungsaufgaben ,

```

```

6286 case = D ,
6287   Name-sg = Übungsaufgabe ,
6288   Name-pl = Übungsaufgaben ,
6289 case = G ,
6290   Name-sg = Übungsaufgabe ,
6291   Name-pl = Übungsaufgaben ,
6292
6293 type = solution ,
6294   gender = f ,
6295   case = N ,
6296     Name-sg = Lösung ,
6297     Name-pl = Lösungen ,
6298   case = A ,
6299     Name-sg = Lösung ,
6300     Name-pl = Lösungen ,
6301   case = D ,
6302     Name-sg = Lösung ,
6303     Name-pl = Lösungen ,
6304   case = G ,
6305     Name-sg = Lösung ,
6306     Name-pl = Lösungen ,
6307 </lang-german>

```

10.4 French

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

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

```

6308 <*package>
6309 \zcDeclareLanguage [ gender = { f , m } ] { french }
6310 \zcDeclareLanguageAlias { acadian } { french }
6311 </package>
6312 <*lang-french>
6313 namesep = {\nobreakspace} ,
6314 pairsep = {\~et\nobreakspace} ,
6315 listsep = { , ~ } ,
6316 lastsep = {\~et\nobreakspace} ,
6317 tpairsep = {\~et\nobreakspace} ,
6318 tlistsep = { , ~ } ,
6319 tlastsep = {\~et\nobreakspace} ,
6320 notesep = { ~ } ,
6321 rangsep = {\~à\nobreakspace} ,
6322
6323 type = book ,
6324   gender = m ,
6325   Name-sg = Livre ,
6326   name-sg = livre ,
6327   Name-pl = Livres ,

```

```

6328     name-pl = livres ,
6329
6330 type = part ,
6331     gender = f ,
6332     Name-sg = Partie ,
6333     name-sg = partie ,
6334     Name-pl = Parties ,
6335     name-pl = parties ,
6336
6337 type = chapter ,
6338     gender = m ,
6339     Name-sg = Chapitre ,
6340     name-sg = chapitre ,
6341     Name-pl = Chapitres ,
6342     name-pl = chapitres ,
6343
6344 type = section ,
6345     gender = f ,
6346     Name-sg = Section ,
6347     name-sg = section ,
6348     Name-pl = Sections ,
6349     name-pl = sections ,
6350
6351 type = paragraph ,
6352     gender = m ,
6353     Name-sg = Paragraphe ,
6354     name-sg = paragraphe ,
6355     Name-pl = Paragraphes ,
6356     name-pl = paragraphes ,
6357
6358 type = appendix ,
6359     gender = f ,
6360     Name-sg = Annexe ,
6361     name-sg = annexe ,
6362     Name-pl = Annexes ,
6363     name-pl = annexes ,
6364
6365 type = page ,
6366     gender = f ,
6367     Name-sg = Page ,
6368     name-sg = page ,
6369     Name-pl = Pages ,
6370     name-pl = pages ,
6371     rangesep = {-} ,
6372     rangetopair = false ,
6373
6374 type = line ,
6375     gender = f ,
6376     Name-sg = Ligne ,
6377     name-sg = ligne ,
6378     Name-pl = Lignes ,
6379     name-pl = lignes ,
6380
6381 type = figure ,

```



```

6382 gender = f ,
6383 Name-sg = Figure ,
6384 name-sg = figure ,
6385 Name-pl = Figures ,
6386 name-pl = figures ,
6387
6388 type = table ,
6389 gender = f ,
6390 Name-sg = Table ,
6391 name-sg = table ,
6392 Name-pl = Tables ,
6393 name-pl = tables ,
6394
6395 type = item ,
6396 gender = m ,
6397 Name-sg = Point ,
6398 name-sg = point ,
6399 Name-pl = Points ,
6400 name-pl = points ,
6401
6402 type = footnote ,
6403 gender = f ,
6404 Name-sg = Note ,
6405 name-sg = note ,
6406 Name-pl = Notes ,
6407 name-pl = notes ,
6408
6409 type = endnote ,
6410 gender = f ,
6411 Name-sg = Note ,
6412 name-sg = note ,
6413 Name-pl = Notes ,
6414 name-pl = notes ,
6415
6416 type = note ,
6417 gender = f ,
6418 Name-sg = Note ,
6419 name-sg = note ,
6420 Name-pl = Notes ,
6421 name-pl = notes ,
6422
6423 type = equation ,
6424 gender = f ,
6425 Name-sg = Équation ,
6426 name-sg = équation ,
6427 Name-pl = Équations ,
6428 name-pl = équations ,
6429 refbounds-first-sg = {,(,)}, ,
6430 refbounds = {(,,)} ,
6431
6432 type = theorem ,
6433 gender = m ,
6434 Name-sg = Théorème ,
6435 name-sg = théorème ,

```

```

6436 Name-pl = Théorèmes ,
6437 name-pl = théorèmes ,
6438
6439 type = lemma ,
6440 gender = m ,
6441 Name-sg = Lemme ,
6442 name-sg = lemme ,
6443 Name-pl = Lemmes ,
6444 name-pl = lemmes ,
6445
6446 type = corollary ,
6447 gender = m ,
6448 Name-sg = Corollaire ,
6449 name-sg = corollaire ,
6450 Name-pl = Corollaires ,
6451 name-pl = corollaires ,
6452
6453 type = proposition ,
6454 gender = f ,
6455 Name-sg = Proposition ,
6456 name-sg = proposition ,
6457 Name-pl = Propositions ,
6458 name-pl = propositions ,
6459
6460 type = definition ,
6461 gender = f ,
6462 Name-sg = Définition ,
6463 name-sg = définition ,
6464 Name-pl = Définitions ,
6465 name-pl = définitions ,
6466
6467 type = proof ,
6468 gender = f ,
6469 Name-sg = Démonstration ,
6470 name-sg = démonstration ,
6471 Name-pl = Démonstrations ,
6472 name-pl = démonstrations ,
6473
6474 type = result ,
6475 gender = m ,
6476 Name-sg = Résultat ,
6477 name-sg = résultat ,
6478 Name-pl = Résultats ,
6479 name-pl = résultats ,
6480
6481 type = remark ,
6482 gender = f ,
6483 Name-sg = Remarque ,
6484 name-sg = remarque ,
6485 Name-pl = Remarques ,
6486 name-pl = remarques ,
6487
6488 type = example ,
6489 gender = m ,

```

```

6490 Name-sg = Exemple ,
6491 name-sg = exemple ,
6492 Name-pl = Exemples ,
6493 name-pl = exemples ,
6494
6495 type = algorithm ,
6496 gender = m ,
6497 Name-sg = Algorithme ,
6498 name-sg = algorithme ,
6499 Name-pl = Algorithmes ,
6500 name-pl = algorithmes ,
6501
6502 type = listing ,
6503 gender = m ,
6504 Name-sg = Listing ,
6505 name-sg = listing ,
6506 Name-pl = Listings ,
6507 name-pl = listings ,
6508
6509 type = exercise ,
6510 gender = m ,
6511 Name-sg = Exercice ,
6512 name-sg = exercice ,
6513 Name-pl = Exercices ,
6514 name-pl = exercices ,
6515
6516 type = solution ,
6517 gender = f ,
6518 Name-sg = Solution ,
6519 name-sg = solution ,
6520 Name-pl = Solutions ,
6521 name-pl = solutions ,
6522 </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.

```

6523 <*package>
6524 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6525 \zcDeclareLanguageAlias { brazilian } { portuguese }
6526 \zcDeclareLanguageAlias { brazil } { portuguese }
6527 \zcDeclareLanguageAlias { portuges } { portuguese }
6528 </package>
6529 <*lang-portuguese>
6530 namesep = {\nobreakspace} ,
6531 pairsep = {\~e\nobreakspace} ,
6532 listsep = { ,~ } ,
6533 lastsep = {\~e\nobreakspace} ,
6534 tpairsep = {\~e\nobreakspace} ,
6535 tlistsep = { ,~ } ,

```

```

6536 tlastsep = {\~e\nobreakspace} ,
6537 notesep = {\~} ,
6538 rangesep = {\~a\nobreakspace} ,
6539
6540 type = book ,
6541   gender = m ,
6542   Name-sg = Livro ,
6543   name-sg = livro ,
6544   Name-pl = Livros ,
6545   name-pl = livros ,
6546
6547 type = part ,
6548   gender = f ,
6549   Name-sg = Parte ,
6550   name-sg = parte ,
6551   Name-pl = Partes ,
6552   name-pl = partes ,
6553
6554 type = chapter ,
6555   gender = m ,
6556   Name-sg = Capítulo ,
6557   name-sg = capítulo ,
6558   Name-pl = Capítulos ,
6559   name-pl = capítulos ,
6560
6561 type = section ,
6562   gender = f ,
6563   Name-sg = Seção ,
6564   name-sg = seção ,
6565   Name-pl = Seções ,
6566   name-pl = seções ,
6567
6568 type = paragraph ,
6569   gender = m ,
6570   Name-sg = Parágrafo ,
6571   name-sg = parágrafo ,
6572   Name-pl = Parágrafos ,
6573   name-pl = parágrafos ,
6574   Name-sg-ab = Par. ,
6575   name-sg-ab = par. ,
6576   Name-pl-ab = Par. ,
6577   name-pl-ab = par. ,
6578
6579 type = appendix ,
6580   gender = m ,
6581   Name-sg = Apêndice ,
6582   name-sg = apêndice ,
6583   Name-pl = Apêndices ,
6584   name-pl = apêndices ,
6585
6586 type = page ,
6587   gender = f ,
6588   Name-sg = Página ,
6589   name-sg = página ,

```

```

6590 Name-pl = Páginas ,
6591 name-pl = páginas ,
6592 rangesep = {\textendash} ,
6593 rangetopair = false ,
6594
6595 type = line ,
6596 gender = f ,
6597 Name-sg = Linha ,
6598 name-sg = linha ,
6599 Name-pl = Linhas ,
6600 name-pl = linhas ,
6601
6602 type = figure ,
6603 gender = f ,
6604 Name-sg = Figura ,
6605 name-sg = figura ,
6606 Name-pl = Figuras ,
6607 name-pl = figuras ,
6608 Name-sg-ab = Fig. ,
6609 name-sg-ab = fig. ,
6610 Name-pl-ab = Figs. ,
6611 name-pl-ab = figs. ,
6612
6613 type = table ,
6614 gender = f ,
6615 Name-sg = Tabela ,
6616 name-sg = tabela ,
6617 Name-pl = Tabelas ,
6618 name-pl = tabelas ,
6619
6620 type = item ,
6621 gender = m ,
6622 Name-sg = Item ,
6623 name-sg = item ,
6624 Name-pl = Itens ,
6625 name-pl = itens ,
6626
6627 type = footnote ,
6628 gender = f ,
6629 Name-sg = Nota ,
6630 name-sg = nota ,
6631 Name-pl = Notas ,
6632 name-pl = notas ,
6633
6634 type = endnote ,
6635 gender = f ,
6636 Name-sg = Nota ,
6637 name-sg = nota ,
6638 Name-pl = Notas ,
6639 name-pl = notas ,
6640
6641 type = note ,
6642 gender = f ,
6643 Name-sg = Nota ,

```

```

6644 name-sg = nota ,
6645 Name-pl = Notas ,
6646 name-pl = notas ,
6647
6648 type = equation ,
6649 gender = f ,
6650 Name-sg = Equação ,
6651 name-sg = equação ,
6652 Name-pl = Equações ,
6653 name-pl = equações ,
6654 Name-sg-ab = Eq. ,
6655 name-sg-ab = eq. ,
6656 Name-pl-ab = Eqs. ,
6657 name-pl-ab = eqs. ,
6658 refbounds-first-sg = {,(,)}, ,
6659 refbounds = {(,,)} ,
6660
6661 type = theorem ,
6662 gender = m ,
6663 Name-sg = Teorema ,
6664 name-sg = teorema ,
6665 Name-pl = Teoremas ,
6666 name-pl = teoremas ,
6667
6668 type = lemma ,
6669 gender = m ,
6670 Name-sg = Lema ,
6671 name-sg = lema ,
6672 Name-pl = Lemas ,
6673 name-pl = lemas ,
6674
6675 type = corollary ,
6676 gender = m ,
6677 Name-sg = Corolário ,
6678 name-sg = corolário ,
6679 Name-pl = Corolários ,
6680 name-pl = corolários ,
6681
6682 type = proposition ,
6683 gender = f ,
6684 Name-sg = Proposição ,
6685 name-sg = proposição ,
6686 Name-pl = Proposições ,
6687 name-pl = proposições ,
6688
6689 type = definition ,
6690 gender = f ,
6691 Name-sg = Definição ,
6692 name-sg = definição ,
6693 Name-pl = Definições ,
6694 name-pl = definições ,
6695
6696 type = proof ,
6697 gender = f ,

```

6698 Name-sg = Demonstração ,
6699 name-sg = demonstração ,
6700 Name-pl = Demonstrações ,
6701 name-pl = demonstrações ,
6702
6703 type = result ,
6704 gender = m ,
6705 Name-sg = Resultado ,
6706 name-sg = resultado ,
6707 Name-pl = Resultados ,
6708 name-pl = resultados ,
6709
6710 type = remark ,
6711 gender = f ,
6712 Name-sg = Observação ,
6713 name-sg = observação ,
6714 Name-pl = Observações ,
6715 name-pl = observações ,
6716
6717 type = example ,
6718 gender = m ,
6719 Name-sg = Exemplo ,
6720 name-sg = exemplo ,
6721 Name-pl = Exemplos ,
6722 name-pl = exemplos ,
6723
6724 type = algorithm ,
6725 gender = m ,
6726 Name-sg = Algoritmo ,
6727 name-sg = algoritmo ,
6728 Name-pl = Algoritmos ,
6729 name-pl = algoritmos ,
6730
6731 type = listing ,
6732 gender = f ,
6733 Name-sg = Listagem ,
6734 name-sg = listagem ,
6735 Name-pl = Listagens ,
6736 name-pl = listagens ,
6737
6738 type = exercise ,
6739 gender = m ,
6740 Name-sg = Exercício ,
6741 name-sg = exercício ,
6742 Name-pl = Exercícios ,
6743 name-pl = exercícios ,
6744
6745 type = solution ,
6746 gender = f ,
6747 Name-sg = Solução ,
6748 name-sg = solução ,
6749 Name-pl = Soluções ,
6750 name-pl = soluções ,
6751 </lang-portuguese>

10.6 Spanish

Spanish language file has been initially provided by the author.

```
6752 \package)
6753 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6754 \package)

6755 \*lang-spanish)

6756 namesep = {\nobreakspace} ,
6757 pairsep = {\~y\nobreakspace} ,
6758 listsep = { , ~ } ,
6759 lastsep = {\~y\nobreakspace} ,
6760 tpairsep = {\~y\nobreakspace} ,
6761 tlistsep = { , ~ } ,
6762 tlastsep = {\~y\nobreakspace} ,
6763 notesep = { ~ } ,
6764 rangesep = {\~a\nobreakspace} ,
6765
6766 type = book ,
6767   gender = m ,
6768   Name-sg = Libro ,
6769   name-sg = libro ,
6770   Name-pl = Libros ,
6771   name-pl = libros ,
6772
6773 type = part ,
6774   gender = f ,
6775   Name-sg = Parte ,
6776   name-sg = parte ,
6777   Name-pl = Partes ,
6778   name-pl = partes ,
6779
6780 type = chapter ,
6781   gender = m ,
6782   Name-sg = Capítulo ,
6783   name-sg = capítulo ,
6784   Name-pl = Capítulos ,
6785   name-pl = capítulos ,
6786
6787 type = section ,
6788   gender = f ,
6789   Name-sg = Sección ,
6790   name-sg = sección ,
6791   Name-pl = Secciones ,
6792   name-pl = secciones ,
6793
6794 type = paragraph ,
6795   gender = m ,
6796   Name-sg = Párrafo ,
6797   name-sg = párrafo ,
6798   Name-pl = Párrafos ,
6799   name-pl = párrafos ,
6800
6801 type = appendix ,
```



```

6802   gender = m ,
6803   Name-sg = Apéndice ,
6804   name-sg = apéndice ,
6805   Name-pl = Apéndices ,
6806   name-pl = apéndices ,
6807
6808   type = page ,
6809   gender = f ,
6810   Name-sg = Página ,
6811   name-sg = página ,
6812   Name-pl = Páginas ,
6813   name-pl = páginas ,
6814   rangesep = {\textendash} ,
6815   rangetopair = false ,
6816
6817   type = line ,
6818   gender = f ,
6819   Name-sg = Línea ,
6820   name-sg = línea ,
6821   Name-pl = Líneas ,
6822   name-pl = líneas ,
6823
6824   type = figure ,
6825   gender = f ,
6826   Name-sg = Figura ,
6827   name-sg = figura ,
6828   Name-pl = Figuras ,
6829   name-pl = figuras ,
6830
6831   type = table ,
6832   gender = m ,
6833   Name-sg = Cuadro ,
6834   name-sg = cuadro ,
6835   Name-pl = Cuadros ,
6836   name-pl = cuadros ,
6837
6838   type = item ,
6839   gender = m ,
6840   Name-sg = Punto ,
6841   name-sg = punto ,
6842   Name-pl = Puntos ,
6843   name-pl = puntos ,
6844
6845   type = footnote ,
6846   gender = f ,
6847   Name-sg = Nota ,
6848   name-sg = nota ,
6849   Name-pl = Notas ,
6850   name-pl = notas ,
6851
6852   type = endnote ,
6853   gender = f ,
6854   Name-sg = Nota ,
6855   name-sg = nota ,

```

```

6856 Name-pl = Notas ,
6857 name-pl = notas ,
6858
6859 type = note ,
6860 gender = f ,
6861 Name-sg = Nota ,
6862 name-sg = nota ,
6863 Name-pl = Notas ,
6864 name-pl = notas ,
6865
6866 type = equation ,
6867 gender = f ,
6868 Name-sg = Ecuación ,
6869 name-sg = ecuación ,
6870 Name-pl = Ecuaciones ,
6871 name-pl = ecuaciones ,
6872 refbounds-first-sg = {,(,)}, ,
6873 refbounds = {(,,)} ,
6874
6875 type = theorem ,
6876 gender = m ,
6877 Name-sg = Teorema ,
6878 name-sg = teorema ,
6879 Name-pl = Teoremas ,
6880 name-pl = teoremas ,
6881
6882 type = lemma ,
6883 gender = m ,
6884 Name-sg = Lema ,
6885 name-sg = lema ,
6886 Name-pl = Lemas ,
6887 name-pl = lemas ,
6888
6889 type = corollary ,
6890 gender = m ,
6891 Name-sg = Corolario ,
6892 name-sg = corolario ,
6893 Name-pl = Corolarios ,
6894 name-pl = corolarios ,
6895
6896 type = proposition ,
6897 gender = f ,
6898 Name-sg = Proposición ,
6899 name-sg = proposición ,
6900 Name-pl = Proposiciones ,
6901 name-pl = proposiciones ,
6902
6903 type = definition ,
6904 gender = f ,
6905 Name-sg = Definición ,
6906 name-sg = definición ,
6907 Name-pl = Definiciones ,
6908 name-pl = definiciones ,
6909

```

```

6910 type = proof ,
6911     gender = f ,
6912     Name-sg = Demostración ,
6913     name-sg = demostración ,
6914     Name-pl = Demostraciones ,
6915     name-pl = demostraciones ,
6916
6917 type = result ,
6918     gender = m ,
6919     Name-sg = Resultado ,
6920     name-sg = resultado ,
6921     Name-pl = Resultados ,
6922     name-pl = resultados ,
6923
6924 type = remark ,
6925     gender = f ,
6926     Name-sg = Observación ,
6927     name-sg = observación ,
6928     Name-pl = Observaciones ,
6929     name-pl = observaciones ,
6930
6931 type = example ,
6932     gender = m ,
6933     Name-sg = Ejemplo ,
6934     name-sg = ejemplo ,
6935     Name-pl = Ejemplos ,
6936     name-pl = ejemplos ,
6937
6938 type = algorithm ,
6939     gender = m ,
6940     Name-sg = Algoritmo ,
6941     name-sg = algoritmo ,
6942     Name-pl = Algoritmos ,
6943     name-pl = algoritmos ,
6944
6945 type = listing ,
6946     gender = m ,
6947     Name-sg = Listado ,
6948     name-sg = listado ,
6949     Name-pl = Listados ,
6950     name-pl = listados ,
6951
6952 type = exercise ,
6953     gender = m ,
6954     Name-sg = Ejercicio ,
6955     name-sg = ejercicio ,
6956     Name-pl = Ejercicios ,
6957     name-pl = ejercicios ,
6958
6959 type = solution ,
6960     gender = f ,
6961     Name-sg = Solución ,
6962     name-sg = solución ,
6963     Name-pl = Soluciones ,

```

```

6964 name-pl = soluciones ,
6965 </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.

```

6966 <*package>
6967 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6968 </package>
6969 <*lang-dutch>
6970 namesep = {\nobreakspace} ,
6971 pairsep = {\sim\nobreakspace} ,
6972 listsep = {,~} ,
6973 lastsep = {\sim\nobreakspace} ,
6974 tpairsep = {\sim\nobreakspace} ,
6975 tlistsep = {,~} ,
6976 tlastsep = {,~\sim\nobreakspace} ,
6977 notesep = {~} ,
6978 rangesep = {\sim/m\nobreakspace} ,
6979
6980 type = book ,
6981 gender = n ,
6982 Name-sg = Boek ,
6983 name-sg = boek ,
6984 Name-pl = Boeken ,
6985 name-pl = boeken ,
6986
6987 type = part ,
6988 gender = n ,
6989 Name-sg = Deel ,
6990 name-sg = deel ,
6991 Name-pl = Delen ,
6992 name-pl = delen ,
6993
6994 type = chapter ,
6995 gender = n ,
6996 Name-sg = Hoofdstuk ,
6997 name-sg = hoofdstuk ,
6998 Name-pl = Hoofdstukken ,
6999 name-pl = hoofdstukken ,
7000
7001 type = section ,
7002 gender = m ,
7003 Name-sg = Paragraaf ,
7004 name-sg = paragraaf ,
7005 Name-pl = Paragrafen ,
7006 name-pl = paragrafen ,
7007
7008 type = paragraph ,
7009 gender = f ,
7010 Name-sg = Alinea ,

```

```

7011 name-sg = alinea ,
7012 Name-pl = Alinea's ,
7013 name-pl = alineas ,
7014

```

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.

```

7015 type = appendix ,
7016 gender = { f , m } ,
7017 Name-sg = Blage ,
7018 name-sg = blage ,
7019 Name-pl = Blagen ,
7020 name-pl = blagen ,
7021
7022 type = page ,
7023 gender = { f , m } ,
7024 Name-sg = Pagina ,
7025 name-sg = pagina ,
7026 Name-pl = Pagina's ,
7027 name-pl = pagina's ,
7028 rangesep = {\textendash} ,
7029 rangetopair = false ,
7030
7031 type = line ,
7032 gender = m ,
7033 Name-sg = Regel ,
7034 name-sg = regel ,
7035 Name-pl = Regels ,
7036 name-pl = regels ,
7037
7038 type = figure ,
7039 gender = { n , f , m } ,
7040 Name-sg = Figuur ,
7041 name-sg = figuur ,
7042 Name-pl = Figuren ,
7043 name-pl = figuren ,
7044
7045 type = table ,
7046 gender = { f , m } ,
7047 Name-sg = Tabel ,
7048 name-sg = tabel ,
7049 Name-pl = Tabellen ,
7050 name-pl = tabellen ,
7051
7052 type = item ,
7053 gender = n ,
7054 Name-sg = Punt ,
7055 name-sg = punt ,
7056 Name-pl = Punten ,
7057 name-pl = punten ,
7058
7059 type = footnote ,
7060 gender = { f , m } ,

```

```

7061 Name-sg = Voetnoot ,
7062 name-sg = voetnoot ,
7063 Name-pl = Voetnoten ,
7064 name-pl = voetnoten ,
7065
7066 type = endnote ,
7067 gender = { f , m } ,
7068 Name-sg = Eindnoot ,
7069 name-sg = eindnoot ,
7070 Name-pl = Eindnoten ,
7071 name-pl = eindnoten ,
7072
7073 type = note ,
7074 gender = f ,
7075 Name-sg = Opmerking ,
7076 name-sg = opmerking ,
7077 Name-pl = Opmerkingen ,
7078 name-pl = opmerkingen ,
7079
7080 type = equation ,
7081 gender = f ,
7082 Name-sg = Vergelking ,
7083 name-sg = vergelking ,
7084 Name-pl = Vergelkingen ,
7085 name-pl = vergelkingen ,
7086 Name-sg-ab = Vgl. ,
7087 name-sg-ab = vgl. ,
7088 Name-pl-ab = Vgl.'s ,
7089 name-pl-ab = vgl.'s ,
7090 refbounds-first-sg = { ,(,) } ,
7091 refbounds = { (,,) } ,
7092
7093 type = theorem ,
7094 gender = f ,
7095 Name-sg = Stelling ,
7096 name-sg = stelling ,
7097 Name-pl = Stellingen ,
7098 name-pl = stellingen ,
7099

```

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”.

```

7100 type = lemma ,
7101 gender = n ,
7102 Name-sg = Lemma ,
7103 name-sg = lemma ,
7104 Name-pl = Lemma's ,
7105 name-pl = lemma's ,
7106
7107 type = corollary ,
7108 gender = n ,
7109 Name-sg = Gevolg ,
7110 name-sg = gevolg ,

```

7111 Name-pl = Gevolgen ,
7112 name-pl = gevolgen ,
7113
7114 type = proposition ,
7115 gender = f ,
7116 Name-sg = Propositie ,
7117 name-sg = propositie ,
7118 Name-pl = Proposities ,
7119 name-pl = proposities ,
7120
7121 type = definition ,
7122 gender = f ,
7123 Name-sg = Definitie ,
7124 name-sg = definitie ,
7125 Name-pl = Definities ,
7126 name-pl = definities ,
7127
7128 type = proof ,
7129 gender = n ,
7130 Name-sg = Bews ,
7131 name-sg = bews ,
7132 Name-pl = Bewzen ,
7133 name-pl = bewzen ,
7134
7135 type = result ,
7136 gender = n ,
7137 Name-sg = Resultaat ,
7138 name-sg = resultaat ,
7139 Name-pl = Resultaten ,
7140 name-pl = resultaten ,
7141
7142 type = remark ,
7143 gender = f ,
7144 Name-sg = Opmerking ,
7145 name-sg = opmerking ,
7146 Name-pl = Opmerkingen ,
7147 name-pl = opmerkingen ,
7148
7149 type = example ,
7150 gender = n ,
7151 Name-sg = Voorbeeld ,
7152 name-sg = voorbeeld ,
7153 Name-pl = Voorbeelden ,
7154 name-pl = voorbeelden ,
7155

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".

7156 type = algorithm ,
7157 gender = { n , f , m } ,
7158 Name-sg = Algoritme ,
7159 name-sg = algoritme ,
7160 Name-pl = Algoritmen ,
7161 name-pl = algoritmen ,

7162

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

```
7163 type = listing ,
7164   gender = m ,
7165   Name-sg = Listing ,
7166   name-sg = listing ,
7167   Name-pl = Listings ,
7168   name-pl = listings ,
7169
7170 type = exercise ,
7171   gender = { f , m } ,
7172   Name-sg = Opgave ,
7173   name-sg = opgave ,
7174   Name-pl = Opgaven ,
7175   name-pl = opgaven ,
7176
7177 type = solution ,
7178   gender = f ,
7179   Name-sg = Oplossing ,
7180   name-sg = oplossing ,
7181   Name-pl = Oplossingen ,
7182   name-pl = oplossingen ,
7183 </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->

```
7184 <*package>
7185 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7186 </package>
7187 <*lang-italian>
7188 namesep = {\nobreakspace} ,
7189 pairsep = {\nobreakspace} ,
7190 listsep = {,~} ,
7191 lastsep = {\nobreakspace} ,
7192 tpairsep = {\nobreakspace} ,
7193 tlistsep = {,~} ,
7194 tlastsep = {,~\nobreakspace} ,
7195 notesep = {~} ,
7196 rangesep = {\nobreakspace} ,
7197 +refbounds-rb = {da\nobreakspace,,} ,
7198
7199 type = book ,
7200   gender = m ,
7201   Name-sg = Libro ,
7202   name-sg = libro ,
7203   Name-pl = Libri ,
7204   name-pl = libri ,
```



```

7205
7206 type = part ,
7207     gender = f ,
7208     Name-sg = Parte ,
7209     name-sg = parte ,
7210     Name-pl = Parti ,
7211     name-pl = parti ,
7212
7213 type = chapter ,
7214     gender = m ,
7215     Name-sg = Capitolo ,
7216     name-sg = capitolo ,
7217     Name-pl = Capitoli ,
7218     name-pl = capitoli ,
7219
7220 type = section ,
7221     gender = m ,
7222     Name-sg = Paragrafo ,
7223     name-sg = paragrafo ,
7224     Name-pl = Paragrafi ,
7225     name-pl = paragrafi ,
7226
7227 type = paragraph ,
7228     gender = m ,
7229     Name-sg = Capoverso ,
7230     name-sg = capoverso ,
7231     Name-pl = Capoversi ,
7232     name-pl = capoversi ,
7233
7234 type = appendix ,
7235     gender = f ,
7236     Name-sg = Appendice ,
7237     name-sg = appendice ,
7238     Name-pl = Appendici ,
7239     name-pl = appendici ,
7240
7241 type = page ,
7242     gender = f ,
7243     Name-sg = Pagina ,
7244     name-sg = pagina ,
7245     Name-pl = Pagine ,
7246     name-pl = pagine ,
7247     Name-sg-ab = Pag. ,
7248     name-sg-ab = pag. ,
7249     Name-pl-ab = Pag. ,
7250     name-pl-ab = pag. ,
7251     rangesep = {\textendash} ,
7252     rangetopair = false ,
7253     +refbounds-rb = {,,} ,
7254
7255 type = line ,
7256     gender = f ,
7257     Name-sg = Riga ,
7258     name-sg = riga ,

```

```

7259 Name-pl = Righe ,
7260 name-pl = righe ,
7261
7262 type = figure ,
7263 gender = f ,
7264 Name-sg = Figura ,
7265 name-sg = figura ,
7266 Name-pl = Figure ,
7267 name-pl = figure ,
7268 Name-sg-ab = Fig. ,
7269 name-sg-ab = fig. ,
7270 Name-pl-ab = Fig. ,
7271 name-pl-ab = fig. ,
7272
7273 type = table ,
7274 gender = f ,
7275 Name-sg = Tabella ,
7276 name-sg = tabella ,
7277 Name-pl = Tabelle ,
7278 name-pl = tabelle ,
7279 Name-sg-ab = Tab. ,
7280 name-sg-ab = tab. ,
7281 Name-pl-ab = Tab. ,
7282 name-pl-ab = tab. ,
7283
7284 type = item ,
7285 gender = m ,
7286 Name-sg = Punto ,
7287 name-sg = punto ,
7288 Name-pl = Punti ,
7289 name-pl = punti ,
7290
7291 type = footnote ,
7292 gender = f ,
7293 Name-sg = Nota ,
7294 name-sg = nota ,
7295 Name-pl = Note ,
7296 name-pl = note ,
7297
7298 type = endnote ,
7299 gender = f ,
7300 Name-sg = Nota ,
7301 name-sg = nota ,
7302 Name-pl = Note ,
7303 name-pl = note ,
7304
7305 type = note ,
7306 gender = f ,
7307 Name-sg = Nota ,
7308 name-sg = nota ,
7309 Name-pl = Note ,
7310 name-pl = note ,
7311
7312 type = equation ,

```

```

7313 gender = f ,
7314 Name-sg = Equazione ,
7315 name-sg = equazione ,
7316 Name-pl = Equazioni ,
7317 name-pl = equazioni ,
7318 Name-sg-ab = Eq. ,
7319 name-sg-ab = eq. ,
7320 Name-pl-ab = Eq. ,
7321 name-pl-ab = eq. ,
7322 +refbounds-rb = {da\nobreakspace(,,)} ,
7323 refbounds-first-sg = {,(,)}, ,
7324 refbounds = {(,,)} ,
7325
7326 type = theorem ,
7327 gender = m ,
7328 Name-sg = Teorema ,
7329 name-sg = teorema ,
7330 Name-pl = Teoremi ,
7331 name-pl = teoremi ,
7332
7333 type = lemma ,
7334 gender = m ,
7335 Name-sg = Lemma ,
7336 name-sg = lemma ,
7337 Name-pl = Lemmi ,
7338 name-pl = lemmi ,
7339
7340 type = corollary ,
7341 gender = m ,
7342 Name-sg = Corollario ,
7343 name-sg = corollario ,
7344 Name-pl = Corollari ,
7345 name-pl = corollari ,
7346
7347 type = proposition ,
7348 gender = f ,
7349 Name-sg = Proposizione ,
7350 name-sg = proposizione ,
7351 Name-pl = Proposizioni ,
7352 name-pl = proposizioni ,
7353
7354 type = definition ,
7355 gender = f ,
7356 Name-sg = Definizione ,
7357 name-sg = definizione ,
7358 Name-pl = Definizioni ,
7359 name-pl = definizioni ,
7360
7361 type = proof ,
7362 gender = f ,
7363 Name-sg = Dimostrazione ,
7364 name-sg = dimostrazione ,
7365 Name-pl = Dimostrazioni ,
7366 name-pl = dimostrazioni ,

```

```

7367
7368 type = result ,
7369     gender = m ,
7370     Name-sg = Risultato ,
7371     name-sg = risultato ,
7372     Name-pl = Risultati ,
7373     name-pl = risultati ,
7374
7375 type = remark ,
7376     gender = f ,
7377     Name-sg = Osservazione ,
7378     name-sg = osservazione ,
7379     Name-pl = Osservazioni ,
7380     name-pl = osservazioni ,
7381
7382 type = example ,
7383     gender = m ,
7384     Name-sg = Esempio ,
7385     name-sg = esempio ,
7386     Name-pl = Esempi ,
7387     name-pl = esempi ,
7388
7389 type = algorithm ,
7390     gender = m ,
7391     Name-sg = Algoritmo ,
7392     name-sg = algoritmo ,
7393     Name-pl = Algoritmi ,
7394     name-pl = algoritmi ,
7395
7396 type = listing ,
7397     gender = m ,
7398     Name-sg = Listato ,
7399     name-sg = listato ,
7400     Name-pl = Listati ,
7401     name-pl = listati ,
7402
7403 type = exercise ,
7404     gender = m ,
7405     Name-sg = Esercizio ,
7406     name-sg = esercizio ,
7407     Name-pl = Esercizi ,
7408     name-pl = esercizi ,
7409
7410 type = solution ,
7411     gender = f ,
7412     Name-sg = Soluzione ,
7413     name-sg = soluzione ,
7414     Name-pl = Soluzioni ,
7415     name-pl = soluzioni ,
7416 </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	
<code>\A</code>	1885
<code>\AddToHook</code>	108, 2031, 2075, 2098, 2129, 2131, 2169, 2242, 2284, 2447, 2460, 2468, 5327, 5363, 5380, 5382, 5387, 5424, 5442, 5476, 5527, 5552
<code>\AddToHookWithArguments</code>	2422
<code>\alph</code>	5457, 5543
<code>\appendix</code>	2, 127, 129, 136
<code>\appendixname</code>	127, 138
<code>\AtEndOfPackage</code>	2458
B	
<code>\babelname</code>	2085
<code>\babelprovide</code>	30, 55
bool commands:	
<code>\bool_gset_false:N</code>	541
<code>\bool_gset_true:N</code>	534, 2434
<code>\bool_if:NTF</code> ...	385, 462, 500, 558, 1761, 1812, 2035, 2039, 2424, 2470, 3444, 3847, 3988, 4115, 4151, 4185, 4252, 4265, 4277, 4328, 4345, 4355, 4360, 4406, 4410, 4466, 4491, 4498, 4504, 4515, 4521, 4549, 4594, 4616, 4645, 4794, 4947, 4949, 5479, 5555
<code>\bool_if:nTF</code>	76, 3558, 3568, 3592, 3609, 3624, 3689, 3697, 4338, 4715, 4756, 4837, 4854
<code>\bool_if_exist:NTF</code>	339, 349, 373, 383, 433, 450, 460, 485, 488, 496, 498, 512, 515, 522, 525, 532, 539
<code>\bool_lazy_all:nTF</code>	4434, 4959
<code>\bool_lazy_and:nnTF</code>	49, 54, 2591, 3415, 3436, 4219, 4290, 4668, 4932, 4974
<code>\bool_lazy_any:nTF</code>	5111, 5120
<code>\bool_lazy_or:nnTF</code>	1378, 1411, 1961, 2540, 2804, 3303, 3328, 3419, 4920
<code>\bool_new:N</code>	340, 350, 375, 434, 452, 490, 513, 516, 523, 526, 533, 540, 800, 1575, 1576, 1603, 1627, 1979, 1986, 1993, 2006, 2007, 2177, 2178, 2179, 2180, 2181, 2277, 2278, 2415, 2427, 3452, 3467, 3729, 3730, 3741, 3742, 3749, 3751, 3752, 3765, 3766, 3767, 3779, 3780, 5441, 5486, 5526
<code>\bool_set:N</code>	3412
<code>\bool_set_eq:NN</code>	548
<code>\bool_set_false:N</code> ..	374, 451, 487, 489, 524, 1588, 1592, 1764, 1862, 1907, 1949, 2014, 2023, 2024, 2041, 2048, 2189, 2193, 2200, 2208, 2209, 2210, 2302, 2314, 3457, 3550, 3796, 3797, 3814, 3853, 3864, 4264, 4456, 4457, 4464, 4465, 4698, 5118, 5135
<code>\bool_set_true:N</code>	341, 351, 435, 514, 517, 527, 1582, 1583, 1587, 1593, 1786, 1808, 1871, 1873, 1911, 1913, 1922, 1924, 1953, 1955, 1968, 1970, 2013, 2018, 2019, 2187, 2194, 2199, 2216, 2218, 2220, 2223, 2224, 2225, 2290, 2295, 3564, 3574, 3578, 3600, 3615, 3630, 3654, 3822, 3848, 3854, 3858, 3865, 4011, 4022, 4033, 4083, 4119, 4155, 4189, 4206, 4287, 4321, 4493, 4553, 4599, 4621, 4650, 4908, 4915, 5037, 5096, 5134, 5171, 5178, 5179, 5197, 5214, 5216, 5455, 5493, 5540
<code>\bool_until_do:Nn</code>	1863, 1914, 1956, 3590, 3815
<code>\bool_while_do:nn</code>	5585
<code>\l_tmpa_bool</code>	1764, 1786, 1808, 1812, 1862, 1863, 1871, 1873, 1907, 1911, 1913, 1914, 1922, 1924, 1949, 1953, 1955, 1956, 1968, 1970
C	
<code>\chaptername</code>	138
clist commands:	
<code>\clist_map_inline:nn</code>	633, 681, 698, 1017, 2211, 2363, 2436, 2940, 5458, 5545
<code>\counterwithin</code>	5
<code>\crefstripprefix</code>	45
cs commands:	
<code>\cs_generate_variant:Nn</code>	73, 282, 288, 295, 306, 317, 328, 343, 353, 360, 367, 378, 402, 412, 437, 444, 455, 493, 519, 529, 536, 543, 971, 1845, 1881, 1932, 1978, 2599, 4750, 4788, 5154, 5258, 5291, 5324
<code>\cs_if_exist:NTF</code>	24, 27, 66, 86, 5351, 5357
<code>\cs_if_exist_p:N</code>	51, 56, 4221, 4292, 4670, 5587
<code>\cs_if_exist_use:N</code>	5423, 5625

<code>\cs_new:Npn</code>	64, 74, 84, 95, 283, 289, 291, 293, 296, 307, 318, 330, 332, 720, 4711, 4751, 4789, 5144	1895, 1937, 2864, 3409, 3423, 3455, 4362, 4378, 4727, 4739, 4768, 4779, 4810, 4817, 4828, 4843, 4874, 4889
<code>\cs_new_protected:Npn</code>	277, 334, 344, 354, 361, 368, 393, 403, 424, 426, 428, 438, 445, 483, 510, 520, 530, 537, 813, 915, 1524, 1543, 1747, 1846, 1891, 1933, 2466, 2597, 3407, 3471, 3513, 3524, 3537, 3667, 3719, 3781, 3995, 4460, 4707, 4709, 4903, 5138, 5155, 5226, 5259, 5292, 5494	<code>\group_end:</code> 748, 969, 1827, 1876, 1927, 1973, 2892, 3426, 3449, 3459, 4375, 4381, 4731, 4743, 4771, 4782, 4813, 4822, 4831, 4846, 4879, 4894
<code>\cs_set_eq:NN</code>	722	
<code>\cs_to_str:N</code>	333	
D		
<code>\d</code>	1885	
<code>\def</code>	3	
E		
exp commands:		
<code>\exp_args:NNe</code>	36, 39	
<code>\exp_args:NNNo</code>	279	
<code>\exp_args:NNNV</code>	1826, 1875, 1926, 1972	
<code>\exp_args:NNo</code>	279, 285	
<code>\exp_args:No</code>	285	
<code>\exp_args:Nx</code>	925, 5499, 5507	
<code>\exp_args:Nxx</code>	1765, 1777, 1789, 1799, 1865, 1916, 1958, 5157, 5163, 5185, 5189, 5204	
<code>\exp_not:N</code>	113, 4362, 4375, 4378, 4381, 4727, 4731, 4739, 4743, 4768, 4771, 4779, 4782, 4810, 4813, 4817, 4822, 4828, 4831, 4843, 4846, 4874, 4879, 4889, 4894	
<code>\exp_not:n</code>	286, 4027, 4050, 4058, 4076, 4089, 4093, 4128, 4137, 4159, 4167, 4174, 4198, 4212, 4229, 4239, 4281, 4304, 4314, 4363, 4374, 4379, 4380, 4537, 4560, 4572, 4606, 4628, 4637, 4657, 4678, 4688, 4728, 4740, 4769, 4770, 4780, 4781, 4811, 4812, 4814, 4818, 4829, 4830, 4832, 4840, 4844, 4845, 4848, 4875, 4890	
<code>\ExplSyntaxOn</code>	30, 931	
F		
<code>\figurename</code>	138	
file commands:		
<code>\file_get:nnNTF</code>	925	
<code>\fmtversion</code>	5	
<code>\footnote</code>	2	
G		
group commands:		
<code>\group_begin:</code>	736, 917, 1763, 1850,	
<code>\hyperlink</code>		5141
H		
I		
<code>\IfBooleanT</code>		3456
<code>\IfClassLoadedTF</code>		121
<code>\ifdraft</code>		2192
<code>\IfFormatAtLeastTF</code>		5, 6
<code>\ifoptionfinal</code>		2198
<code>\IfPackageAtLeastTF</code>		2288
<code>\IfPackageLoadedTF</code>		119
<code>\input</code>		30
int commands:		
<code>\int_case:nnTF</code>	3998, 4040, 4102, 4413, 4528, 4585	
<code>\int_compare:nNnTF</code>	3601, 3616, 3631, 3643, 3655, 3675, 3677, 3721, 3895, 4018, 4046, 4068, 4123, 4193, 4398, 4400, 4477, 4507, 4566, 5167, 5173, 5193, 5199, 5603	
<code>\int_compare_p:nNn</code>	1381, 1414, 1962, 1963, 2542, 2806, 3306, 3331, 3691, 3699, 4438, 4924, 4935, 4964, 5131	
<code>\int_gincr:N</code>		112
<code>\int_incr:N</code>	4453, 4485, 4497, 4499, 4514, 4516, 4520, 4522, 4534, 4557, 4569, 4603, 4625, 4634, 4654, 4705, 5601	
<code>\int_new:N</code>	106, 3468, 3469, 3731, 3732, 3733, 3746, 3747	
<code>\int_rand:n</code>		304, 315, 326
<code>\int_set:Nn</code>	3676, 3678, 3682, 3685, 5584	
<code>\int_to_roman:n</code>	5588, 5595, 5596, 5599	
<code>\int_use:N</code>		52, 57, 62, 68, 116
<code>\int_zero:N</code>	3669, 3670, 3791, 3792, 3793, 3794, 3795, 4451, 4452, 4454, 4455, 4700, 4701	
<code>\l_tmpa_int</code>	5584, 5588, 5595, 5596, 5599, 5601, 5603	
iow commands:		
<code>\iow_char:N</code>	125, 140, 141, 146, 147, 152, 153, 158, 159, 211, 228, 275, 2259, 2268	
<code>\iow_newline:</code>		269

K	
keys commands:	
\l_keys_choice_tl	805
\keys_define:nn	
. 20, 641, 687, 704, 765, 972, 1061,	
1086, 1114, 1323, 1362, 1440, 1550,	
1577, 1604, 1613, 1628, 1637, 1980,	
1987, 1994, 2000, 2008, 2043, 2052,	
2066, 2094, 2133, 2164, 2171, 2183,	
2244, 2251, 2253, 2262, 2272, 2279,	
2291, 2303, 2315, 2323, 2330, 2359,	
2385, 2409, 2416, 2429, 2449, 2478,	
2496, 2526, 2560, 2583, 2609, 2621,	
2645, 2757, 2787, 2827, 2895, 2966,	
2990, 3017, 3223, 3253, 3293, 3355	
\keys_set:nn 27, 30, 57, 59, 84, 745,	
893, 956, 2296, 2598, 2603, 2889, 3410	
keyval commands:	
\keyval_parse:nnn	1529, 2334, 2389
L	
\label	2, 60, 63, 132
\labelformat	3
\languagename	24, 138, 2079
M	
\mainbabelname	24, 2086
MH commands:	
\MH_if_boolean:nTF	5491
msg commands:	
\msg_info:nnn	959,
1012, 1077, 1270, 1276, 1330, 5401,	
5434, 5483, 5518, 5559, 5577, 5604	
\msg_info:nnnn	
. 985, 992, 1022, 1394, 1428	
\msg_info:nnnnn	1006
\msg_line_context:	124, 130,
134, 136, 139, 145, 151, 157, 163,	
168, 173, 178, 183, 189, 194, 197,	
200, 205, 209, 216, 221, 226, 233,	
242, 247, 252, 256, 258, 260, 262, 269	
\msg_new:nnn	122, 128,
133, 135, 137, 143, 149, 155, 161,	
166, 171, 176, 181, 186, 191, 196,	
198, 203, 208, 210, 212, 214, 219,	
224, 230, 232, 234, 239, 245, 250,	
255, 257, 259, 261, 263, 265, 267, 272	
\msg_warning:nn	
. 2040, 2046, 2318, 2594, 4440	
\msg_warning:nnn	
. 740, 761, 1556, 1563, 1719,	
1725, 2118, 2155, 2167, 2229, 2240,	
2282, 2307, 2391, 2453, 2463, 2613,	
2727, 2733, 2891, 2935, 2981, 3173,	
3179, 3260, 3879, 4259, 4953, 4969	
\msg_warning:nnnn	
. 829, 846, 880, 898, 2071, 2258,	
2267, 2336, 2501, 2507, 2513, 2519,	
2549, 2762, 2768, 2774, 2780, 2816,	
2908, 2915, 2945, 3228, 3234, 3240,	
3246, 3319, 3344, 3887, 5038, 5098	
\msg_warning:nnnnn 866, 905, 2929, 4988	
\msg_warning:nnnnnn	4995
N	
\NeedsTeXFormat	4
\newcounter	5, 5347, 5348
\NewDocumentCommand	
. 734, 751, 2595, 2600, 2862, 3405, 3453	
\newfloat	129
\NewHook	1636
\newsubfloat	129
\newtheorem	137
\nobreakspace	1537, 5661, 5662,
5664, 5665, 5667, 5669, 5866, 5867,	
5869, 5870, 5872, 5874, 6313, 6314,	
6316, 6317, 6319, 6321, 6530, 6531,	
6533, 6534, 6536, 6538, 6756, 6757,	
6759, 6760, 6762, 6764, 6970, 6971,	
6973, 6974, 6976, 6978, 7188, 7189,	
7191, 7192, 7194, 7196, 7197, 7322	
\NumCheckSetup	45
\NumsCheckSetup	45
P	
\PackageError	9
\pagename	138
\pagenote	130
\pagenumbering	7
\pageref	85
\PagesCheckSetup	45
\paragraph	61
\part	137
\partname	138
prg commands:	
\prg_generate_conditional_-	
variant:Nnn	422,
470, 481, 508, 553, 561, 730, 1889	
\prg_new_conditional:Npnn	
. 118, 120, 379, 456, 494, 555, 724	
\prg_new_protected_conditional:Npnn	
. 413, 472, 544, 1882	
\prg_return_false:	
. 119, 121, 387, 391, 420, 464, 468,	
479, 502, 506, 551, 558, 559, 728, 1887	
\prg_return_true:	
. 119, 121, 386, 389, 418, 463,	
466, 477, 501, 504, 549, 558, 727, 1886	
\prg_set_eq_conditional:NNn	732

prop commands:	
\prop_if_in:NnTF	36
\prop_if_in_p:Nn	77
\prop_item:Nn	39, 78
\prop_new:N	2329, 2384
\prop_put:Nnn	1547
\prop_remove:Nn	1546
\providecommand	5
\ProvidesExplPackage	14
\ProvidesFile	30
	R
\refstepcounter	3, 4, 130, 131, 133, 134
regex commands:	
\regex_match:nnTF	1885
\renewlist	134
\RequirePackage	16, 17, 18, 19, 2036
\roman	7
	S
\scantokens	127
seq commands:	
\seq_clear:N	449, 824, 861, 942, 955, 1016, 1624, 2537, 2801, 2875, 2888, 2939, 3473, 5285
\seq_const_from_clist:Nn	21
\seq_count:N	1382, 1396, 1415, 1430, 2542, 2551, 2806, 2818, 3307, 3321, 3332, 3346
\seq_gclear:N	1375, 1408, 3300, 3325
\seq_gconcat:NNN	626, 630
\seq_get_left:NN	839, 850, 946, 994, 2879, 2917, 3825
\seq_gput_right:Nn	957, 963, 2440
\seq_gremove_all:Nn	2472
\seq_gset_eq:NN	442, 1044
\seq_gset_from_clist:Nn	569, 578, 590, 605, 613, 774, 789
\seq_gset_split:Nnn	427
\seq_if_empty:NTF	825, 862, 943, 983, 1004, 2876, 2906, 2927, 3819, 4986
\seq_if_exist:NTF	430, 440, 447, 458
\seq_if_in:NnTF	843, 877, 921, 989, 1019, 2365, 2438, 2471, 2912, 2942, 3517, 4982
\seq_item:Nn	4721, 4726, 4732, 4734, 4737, 4738, 4744, 4745, 4762, 4767, 4772, 4774, 4777, 4778, 4783, 4784, 4815, 4816, 4823, 4825, 4860, 4872, 4880, 4883, 4887, 4888, 4895, 4896
\seq_map_break:n	98, 3710, 3713
\seq_map_function:NN	3476
\seq_map_indexed_inline:Nn	44, 3671
\seq_map_inline:Nn	1058, 1083, 1320, 1359, 1437, 2462, 2475, 2523, 2557, 2606, 2618, 2784, 2824, 2963, 2987, 3250, 3290, 3352, 3707, 5497
\seq_map_tokens:Nn	80
\seq_new:N	431, 441, 566, 567, 568, 577, 589, 604, 612, 625, 629, 769, 784, 914, 1036, 1612, 2358, 2428, 3451, 3470, 3728, 3745, 3768, 3769, 3770, 3771, 3772, 3773, 3774, 3775, 3776, 3777, 3778
\seq_pop_left:NN	3817
\seq_put_right:Nn	1020, 2367, 2943, 3520
\seq_reverse:N	1618
\seq_set_eq:NN	432, 476, 3783, 4009, 4020, 4031, 4081, 4117, 4153, 4187, 4203, 4285, 4319, 4330, 4551, 4596, 4618, 4647
\seq_set_from_clist:Nn	1617, 3411
\seq_set_split:Nnn	425
\seq_sort:Nn	87, 3479
\seq_use:Nn	5000
\g_tmpa_seq	1375, 1377, 1382, 1391, 1396, 1408, 1410, 1415, 1425, 1430, 3300, 3302, 3307, 3316, 3321, 3325, 3327, 3332, 3341, 3346
\l_tmpa_seq	1016, 1020, 1052, 2537, 2539, 2542, 2546, 2551, 2801, 2803, 2806, 2813, 2818, 2939, 2943, 2958
\setcounter	5349, 5350, 5366, 5381, 5385
\sidefootnote	130
sort commands:	
\sort_return_same:	88, 92, 3486, 3491, 3565, 3585, 3606, 3621, 3635, 3660, 3695, 3710, 3726
\sort_return_swapped:	88, 92, 3499, 3575, 3584, 3605, 3620, 3636, 3659, 3703, 3713, 3725
\space	11
\stepcounter	133, 5365, 5384
str commands:	
\str_case:nn	45
\str_case:nnTF	1119, 1641, 2100, 2137, 2213, 2649, 3022
\str_compare:nNnTF	3581
\str_if_eq:nnTF	97, 4753
\str_if_eq_p:nn	5116, 5122, 5124, 5128
\str_new:N	2051
\str_set:Nn	2056, 2058, 2060, 2062
\string	5504, 5512
\subparagraph	136
\subref	135
\subsections	136

<code>\subsubsection</code>	61	<code>\zrefclever@required@kernel</code>	3, 4, 6, 11
<code>\subsubsections</code>	136	<code>\textendash</code>	1541, 5716, 5980, 6592, 6814, 7028, 7251
<code>\subsubsubsection</code>	61	<code>\thechapter</code>	127
T			
<code>\tablename</code>	138	<code>\thelstnumber</code>	134
<code>\tag</code>	124, 131, 133	<code>\thepage</code>	7, 110, 113
TeX and L ^A T _E X 2 _ε commands:			
<code>\@Alph</code>	127	<code>\thesection</code>	127
<code>\@addtoreset</code>	5	tl commands:	
<code>\@auxout</code>	5503, 5511	<code>\c_novalue_tl</code>	689, 690, 691, 692, 693, 694, 695, 2480, 2528, 2623, 2789
<code>\@bsphack</code>	918, 5496	<code>\tl_clear:N</code>	348, 372, 833, 871, 884, 901, 910, 935, 944, 977, 2604, 2867, 2877, 2900, 3785, 3786, 3787, 3788, 3789, 3790, 3821, 4446, 4447, 4448, 4449, 4450, 4496, 4513, 4907, 4914, 4944, 5036, 5095, 5252
<code>\@capttype</code>	5423, 5625	<code>\tl_const:Nn</code>	1526
<code>\@chapapp</code>	127	<code>\tl_gclear:N</code>	365, 409
<code>\@currentcounter</code>	2–5, 63, 130, 131, 134, 27, 28, 55, 56, 57, 2412, 2413	<code>\tl_gset:Nn</code>	358, 399, 743, 758
<code>\@currentlabel</code>	3, 124, 134	<code>\tl_gset_eq:NN</code>	113
<code>\@elt</code>	5	<code>\tl_head:N</code>	3619, 3632, 3644, 3646, 3656, 3658
<code>\@esphack</code>	968, 5516	<code>\tl_head:n</code>	1866, 1917, 1959, 1963
<code>\@ifl@t@r</code>	5	<code>\tl_if_empty:NTF</code>	34, 88, 827, 837, 864, 875, 896, 903, 1010, 1066, 1091, 1123, 1158, 1195, 1232, 1280, 1328, 1334, 1367, 1445, 1483, 1749, 1870, 1921, 2933, 2971, 2995, 3026, 3061, 3098, 3135, 3183, 3258, 3264, 3298, 3360, 3381, 3427, 4254, 4847, 4929, 4951, 5009, 5020, 5063
<code>\@onlypreamble</code>	750, 764, 2894	<code>\tl_if_empty:nTF</code>	737, 753, 976, 1268, 1545, 1554, 1717, 2433, 2725, 2899, 3171, 5140
<code>\bbl@loaded</code>	55	<code>\tl_if_empty_p:N</code>	50, 55, 2593, 4220, 4291, 4669, 4962, 4976, 5115, 5125, 5129
<code>\bbl@main@language</code>	24, 2080	<code>\tl_if_empty_p:n</code>	1379, 1412, 2541, 2805, 3304, 3329, 3560, 3561, 3570, 3571, 3596, 3597, 3612, 3627
<code>\c@lstnumber</code>	134	<code>\tl_if_eq:NNTF</code>	110, 3531, 3554, 3861
<code>\c@page</code>	7	<code>\tl_if_eq:NnTF</code>	1775, 3474, 3506, 3681, 3684, 3709, 3712, 3829, 3877, 4911, 5161
<code>\caption@subtypelink</code>	5626	<code>\tl_if_eq:nnTF</code>	1765, 1777, 1789, 1799, 1865, 1916, 1958, 3673, 5157, 5163, 5185, 5189, 5204, 5499, 5507
<code>\hyper@@link</code>	113, 122	<code>\tl_if_exist:NTF</code>	336, 346, 356, 363, 370, 381, 397, 407, 726
<code>\hyper@linkfile</code>	5142	<code>\tl_if_exist_p:N</code>	2592
<code>\lst@AddToHook</code>	5575	<code>\tl_if_novalue:nTF</code>	2483, 2531, 2626, 2792
<code>\ltx@gobble</code>	63	<code>\tl_map_break:n</code>	98
<code>\MT@newlabel</code>	5504, 5512		
<code>\protected@write</code>	5503, 5511		
<code>\zref@addprop</code>	21, 31, 46, 61, 63, 105, 117		
<code>\zref@default</code>	113, 4708, 4710		
<code>\zref@extractdefault</code>	11, 12, 122, 280, 286, 290, 1561, 1723, 2731, 3177, 5146, 5542		
<code>\zref@ifpropundefined</code>	43, 1274, 1751, 1759, 1818, 1836, 1848, 1893, 1935, 3882, 4713, 4796, 4850, 5149		
<code>\zref@ifrefcontainsprop</code>	43, 45, 1751, 1759, 1818, 1836, 1848, 1893, 1935, 3882, 4713, 4796, 4850, 5149		
<code>\zref@ifrefundefined</code>	3481, 3483, 3495, 3850, 3852, 3857, 3874, 4256, 4267, 4468, 4791, 4905		
<code>\zref@label</code>	63, 2425		
<code>\zref@localaddprop</code>	5425, 5480, 5556, 5627		
<code>\ZREF@mainlist</code>	21, 31, 46, 61, 63, 105, 117, 5425, 5480, 5556, 5627		
<code>\zref@newprop</code>	6, 7, 20, 22, 32, 47, 62, 100, 116, 5422, 5457, 5543, 5624		
<code>\zref@refused</code>	3872		
<code>\zref@wrapper@babel</code>	63, 84, 2425, 3406		

<code>\tl_map_tokens:Nn</code>	90	<code>\zcDeclareLanguage</code> .	13, 25, <u>734</u> , 5651, 5856, 6309, 6524, 6753, 6967, 7185
<code>\tl_new:N</code>	107, 337, 347, 357, 364, 398, 408, 563, 564, 565, 715, 716, 717, 718, 742, 757, 1549, 2163, 2182, 2250, 2271, 2322, 2408, 3461, 3462, 3463, 3464, 3465, 3466, 3734, 3735, 3736, 3737, 3738, 3739, 3740, 3743, 3744, 3748, 3750, 3753, 3754, 3755, 3756, 3757, 3758, 3759, 3760, 3761, 3762, 3763, 3764	<code>\zcDeclareLanguageAlias</code> 26, <u>751</u> , 5652, 5653, 5654, 5655, 5656, 5657, 5658, 5859, 5860, 5861, 5862, 5863, 6310, 6525, 6526, 6527
<code>\tl_put_left:Nn</code> .	4341, 4348, 4391, 5022, 5023, 5065, 5067, 5069, 5071	<code>\zcLanguageSetup</code> 21, 30, 68, 73, 74, 138, <u>2862</u>
<code>\tl_put_right:Nn</code> .	4025, 4048, 4056, 4074, 4087, 4126, 4135, 4157, 4165, 4172, 4196, 4210, 4227, 4237, 4535, 4558, 4570, 4604, 4626, 4635, 4655, 4676, 4686, 4930, 4931, 4942, 5626	<code>\zcpageref</code>	85, <u>3453</u>
<code>\tl_reverse:N</code>	3541, 3544	<code>\zcref</code>	21, 65, 67, 83–87, 93, 95, 132, <u>3405</u> , 3458
<code>\tl_set:Nn</code> .	279, 338, 719, 744, 934, 978, 990, 1558, 1565, 1567, 1756, 1824, 1828, 1841, 1852, 1857, 1868, 1869, 1877, 1879, 1897, 1902, 1919, 1920, 1928, 1930, 1939, 1944, 1965, 1966, 1974, 1976, 2079, 2080, 2085, 2086, 2089, 2090, 2104, 2110, 2115, 2141, 2147, 2152, 2602, 2868, 2901, 2913, 3648, 3650, 3831, 3832, 4005, 4007, 4279, 4302, 4312, 4358, 4481, 4483, 4494, 4511, 4926, 4927, 4940	<code>\zcRefTypeSetup</code>	21, 68, <u>2600</u>
<code>\tl_set_eq:NN</code>	417, 4444	<code>\zcsetup</code>	21, 55, 65, 67, 68, <u>2595</u>
<code>\tl_show:N</code>	4407	<code>\zlabel</code>	2, 63, <u>129</u> , <u>130</u> , <u>132</u>
<code>\tl_tail:N</code>	3649, 3651	zrefcheck commands:	
<code>\tl_tail:n</code> 1868, 1869, 1919, 1920, 1965, 1966	<code>\zrefcheck_zcref_beg_label:</code> ..	3418
<code>\tl_use:N</code>	301, 312, 323, 759, 923, 928, 958, 960, 964	<code>\zrefcheck_zcref_end_label_-</code> maybe:	3440
<code>\l_tmpa_tl</code>	932, 956, 1852, 1866, 1868, 1897, 1908, 1917, 1919, 1939, 1950, 1959, 1965, 3432, 3433	<code>\zrefcheck_zcref_run_checks_on_-</code> labels:n	3441
<code>\l_tmpb_tl</code> .	1814, 1821, 1824, 1828, 1857, 1866, 1869, 1870, 1877, 1902, 1910, 1917, 1920, 1921, 1928, 1944, 1952, 1959, 1962, 1963, 1966, 1974	zrefclever commands:	
		<code>\zrefclever_language_if_declared:n</code>	732
		<code>\zrefclever_language_if_declared:nTF</code>	<u>732</u>
		<code>\zrefclever_language_varname:n</code> <u>722</u> , <u>722</u>
		<code>\l_zrefclever_ref_language_tl</code> ..	<u>718</u>
		zrefclever internal commands:	
		<code>\l_zrefclever_abbrev_bool</code> <u>3753</u> , 3940, 4933
		<code>\l_zrefclever_amsmath_subequations_</code> <code>bool</code>	5441, 5455, 5479
		<code>\l_zrefclever_breqn_dgroup_bool</code>	5526, 5540, 5555
		<code>\l_zrefclever_cap_bool</code> <u>3753</u> , 3936, 4921
		<code>\l_zrefclever_capfirst_bool</code> 1986, 1989, 4923
		<code>_zrefclever_compat_module:nn</code> 64, <u>2466</u> , 2466, 5325, 5343, 5404, 5437, 5487, 5522, 5562, 5580, 5607, 5630
		<code>_zrefclever_counter_reset_by:n</code> ..	6, 62, 66, 68, 70, <u>74</u> , 74, 5449, 5534
		<code>_zrefclever_counter_reset_by_-</code> <code>aux:nn</code>	81, 84
		<code>_zrefclever_counter_reset_by_-</code> <code>auxi:nnn</code>	91, 95
		<code>\l_zrefclever_counter_resetby_-</code> <code>prop</code>	5, 62, 77, 78, 2384, 2396
		<code>\l_zrefclever_counter_resetters_-</code> <code>seq</code> ..	5, 6, 62, 80, 2358, 2365, 2368
U			
use commands:			
<code>\use:N</code>	25, 28, 805, 4223, 4298, 4672, 5318		
<code>\UseHook</code>	1851, 1896, 1938		
V			
<code>\value</code>	5366, 5385		
Z			
<code>\Z</code>	1885		

\l_zrefclever_counter_type_prop
..... 4, 61, 36, 39, 2329, 2341
\l_zrefclever_current_counter_
tl 3, 5, 63, 20, 24, 25,
37, 40, 42, 50, 51, 52, 103, 2408, 2411
\l_zrefclever_current_language_
tl 24,
55, 716, 2079, 2085, 2089, 2105, 2142
\l_zrefclever_endrangefunc_tl ..
... 3753, 3928, 4220, 4221, 4223,
4291, 4292, 4298, 4669, 4670, 4672
\l_zrefclever_endrangeprop_tl ..
..... 48, 1749, 1759, 3753, 3932
_zrefclever_extract:nnn
..... 12, 289, 289, 1854, 1859,
1899, 1904, 1941, 1946, 3602, 3604,
3617, 3634, 3722, 3724, 5168, 5170,
5174, 5176, 5194, 5196, 5200, 5202
_zrefclever_extract_default:Nnn
... 11, 277, 277, 282, 1753, 1814,
1821, 1831, 1838, 3515, 3526, 3528,
3539, 3542, 3545, 3547, 3835, 3838
_zrefclever_extract_unexp:nnn .
..... 11, 122,
283, 283, 288, 1767, 1771, 1779,
1783, 1791, 1795, 1801, 1805, 4370,
4724, 4729, 4741, 4765, 4806, 4819,
4868, 4876, 4891, 5147, 5150, 5151,
5158, 5159, 5164, 5165, 5186, 5187,
5190, 5191, 5206, 5210, 5500, 5508
_zrefclever_extract_url_
unexp:n 4366, 4723,
4764, 4802, 4864, 5144, 5144, 5154
_zrefclever_get_enclosing_
counters_value:n
..... 6, 64, 64, 69, 73, 102
_zrefclever_get_endrange_
pagecomp:nnN 1891, 1932
_zrefclever_get_endrange_
pagecomptwo:nnN 1933, 1978
_zrefclever_get_endrange_
property:nnN 45, 1747, 1845
_zrefclever_get_endrange_
stripprefix:nnN 1846, 1881
_zrefclever_get_ref:nN
. 113, 114, 4028, 4051, 4059, 4077,
4090, 4094, 4129, 4138, 4160, 4168,
4175, 4199, 4213, 4240, 4282, 4315,
4350, 4538, 4561, 4573, 4607, 4629,
4638, 4658, 4689, 4711, 4711, 4750
_zrefclever_get_ref_endrange:nnN
..... 45,
114, 4230, 4305, 4679, 4751, 4751, 4788
_zrefclever_get_ref_first: ...
113, 114, 118, 4342, 4392, 4789, 4789
_zrefclever_get_rf_opt_bool:nN 126
_zrefclever_get_rf_opt_
bool:nnnN 20,
3933, 3937, 3941, 5292, 5292, 5324
_zrefclever_get_rf_opt_
seq:nnnN .. 20, 125, 3945, 3949,
3953, 3957, 3961, 3965, 3969, 3973,
3977, 3981, 4978, 5259, 5259, 5291
_zrefclever_get_rf_opt_tl:nnnN
..... 20,
22, 45, 125, 3429, 3800, 3804, 3808,
3897, 3901, 3905, 3909, 3913, 3917,
3921, 3925, 3929, 5226, 5226, 5258
_zrefclever_hyperlink:nnn
..... 122, 4364,
4722, 4763, 4800, 4862, 5138, 5138
\l_zrefclever_hyperlink_bool ...
2006, 2013, 2018, 2023, 2035, 2041,
2048, 3457, 4717, 4758, 4856, 5113
\l_zrefclever_hyperref_warn_
bool ... 2007, 2014, 2019, 2024, 2039
_zrefclever_if_class_loaded:n .
..... 118, 120
_zrefclever_if_class_loaded:nTF
..... 5406
_zrefclever_if_package_
loaded:n 118, 118
_zrefclever_if_package_
loaded:nTF 2033,
2077, 2083, 2286, 5345, 5439,
5489, 5524, 5564, 5582, 5609, 5632
_zrefclever_is_integer_rgx:n ..
..... 1882, 1883, 1890
_zrefclever_is_integer_rgx:nTF
..... 1908, 1910, 1950, 1952
\l_zrefclever_label_a_tl
. 92, 3734, 3818, 3837, 3850, 3872,
3874, 3880, 3883, 3889, 4006, 4028,
4051, 4059, 4094, 4160, 4175, 4225,
4231, 4240, 4272, 4282, 4300, 4306,
4315, 4468, 4472, 4482, 4495, 4512,
4538, 4574, 4638, 4674, 4680, 4689
\l_zrefclever_label_b_tl
..... 92, 3734,
3821, 3826, 3840, 3852, 3857, 4472
\l_zrefclever_label_count_int ..
..... 93, 3731, 3791,
3895, 3998, 4451, 4477, 4705, 4965
\l_zrefclever_label_enclval_a_
tl 3461, 3539, 3541, 3596,
3612, 3632, 3644, 3648, 3649, 3656

`\l_zrefclever_label_enclval_b_`
`tl` 3461, 3542, 3544, 3597,
3619, 3627, 3646, 3650, 3651, 3658
`\l_zrefclever_label_extdoc_a_tl`
. 3461, 3545, 3555, 3560, 3570, 3583
`\l_zrefclever_label_extdoc_b_tl`
. 3461, 3547, 3556, 3561, 3571, 3582
`\l_zrefclever_label_type_a_tl` . .
. 3430, 3461, 3516, 3518,
3521, 3527, 3532, 3681, 3709, 3801,
3805, 3809, 3831, 3836, 3862, 3877,
3898, 3902, 3906, 3910, 3914, 3918,
3922, 3926, 3930, 3934, 3938, 3942,
3946, 3950, 3954, 3958, 3962, 3966,
3970, 3974, 3978, 3982, 4008, 4484
`\l_zrefclever_label_type_b_tl` . .
. 3461, 3529,
3533, 3684, 3712, 3832, 3839, 3863
`_zrefclever_label_type_put_`
`new_right:n` 86, 87, 3477, 3513, 3513
`\l_zrefclever_label_types_seq` . .
. 87, 3470, 3473, 3517, 3520, 3707
`\l_zrefclever_labelhook_bool` . . .
. 2415, 2418, 2424
`_zrefclever_labels_in_sequence:n`
. 48, 93, 123, 4270, 4471, 5155, 5155
`\l_zrefclever_lang_decl_case_tl`
. 563, 944, 947, 990, 995, 1334, 1351,
2877, 2880, 2913, 2918, 3264, 3281
`\l_zrefclever_lang_declension_`
`seq` 563,
823, 824, 825, 839, 843, 850, 941,
942, 943, 946, 983, 989, 994, 2874,
2875, 2876, 2879, 2906, 2912, 2917
`\l_zrefclever_lang_gender_seq` . .
. 563, 860, 861, 862, 877, 954,
955, 1004, 1019, 2887, 2888, 2927, 2942
`_zrefclever_language_if_`
`declared:n` 25, 724, 731, 733
`_zrefclever_language_if_`
`declared:n(TF)` 25
`_zrefclever_language_if_`
`declared:nTF` 298, 309, 320, 724,
739, 755, 815, 919, 2116, 2153, 2865
`_zrefclever_language_varname:n`
. 24, 25, 301, 312,
323, 720, 720, 723, 726, 742, 743,
757, 758, 759, 923, 928, 958, 960, 964
`\l_zrefclever_last_of_type_bool`
. 93, 3728, 3848,
3853, 3854, 3858, 3864, 3865, 3988
`\l_zrefclever_lastsep_tl` . 3753,
3912, 4058, 4093, 4137, 4174, 4212
`\l_zrefclever_link_star_bool` . . .
. 3412, 3451, 4718, 4759, 4857, 5114
`\l_zrefclever_listsep_tl`
. 3753, 3908, 4089, 4167, 4537,
4560, 4572, 4606, 4628, 4637, 4657
`\g_zrefclever_loaded_langfiles_`
`seq` 914, 922, 957, 963
`\l_zrefclever_main_language_tl` .
. 24,
55, 717, 2080, 2086, 2090, 2111, 2148
`_zrefclever_mathtools_showonlyrefs:n`
. 3446, 5494
`\l_zrefclever_mathtools_`
`showonlyrefs_bool` 3444, 5486, 5493
`_zrefclever_name_default:`
. 4707, 4709, 4839
`\l_zrefclever_name_format_`
`fallback_tl` 3740, 4940,
4944, 5009, 5058, 5070, 5072, 5090
`\l_zrefclever_name_format_tl` . . .
. 3740, 4926, 4927, 4930, 4931,
4941, 4942, 5015, 5022, 5023, 5031,
5039, 5049, 5066, 5067, 5080, 5100
`\l_zrefclever_name_in_link_bool`
. 115,
118, 3740, 4360, 4794, 5118, 5134, 5135
`\l_zrefclever_namefont_tl` 3753,
3920, 4363, 4379, 4811, 4829, 4844
`\l_zrefclever_nameinlink_str` . . .
. 2051, 2056, 2058,
2060, 2062, 5116, 5122, 5124, 5128
`\l_zrefclever_namesep_tl`
. 3753, 3900, 4814, 4832, 4840, 4848
`\l_zrefclever_next_is_same_bool`
. 93, 123, 3746,
4465, 4498, 4515, 4521, 5179, 5217
`\l_zrefclever_next_maybe_range_`
`bool`
. 93, 123, 3746, 4264, 4277, 4464,
4491, 4504, 5171, 5178, 5197, 5215
`\l_zrefclever_noabbrev_first_`
`bool` 1993, 1996, 4937
`\g_zrefclever_nocompat_bool` . . .
. 2427, 2434, 2470
`\l_zrefclever_nocompat_bool` . . . 64
`\g_zrefclever_nocompat_modules_`
`seq` 2428, 2438, 2441, 2462, 2471, 2472
`\l_zrefclever_nocompat_modules_`
`seq` 64
`\l_zrefclever_nudge_comptosing_`
`bool` 2179, 2209, 2218, 2224, 4961
`\l_zrefclever_nudge_enabled_`
`bool` 2177, 2187, 2189,
2193, 2194, 2199, 2200, 4436, 4947

```

\l_zrefclever_nudge_gender_bool
..... 2181, 2210, 2220, 2225, 4975
\l_zrefclever_nudge_multitype_-
bool ... 2178, 2208, 2216, 2223, 4437
\l_zrefclever_nudge_singular_-
bool ..... 2180, 2236, 4949
\__zrefclever_opt_bool_get:NN ...
..... 544, 554
\__zrefclever_opt_bool_get:NN(TF)
..... 19
\__zrefclever_opt_bool_get:NNTF .
... 544, 5295, 5300, 5305, 5310, 5315
\__zrefclever_opt_bool_gset_-
false:N ..... 19,
510, 537, 543, 1492, 1509, 3383, 3391
\__zrefclever_opt_bool_gset_-
true:N ..... 19,
510, 530, 536, 1454, 1471, 3362, 3370
\__zrefclever_opt_bool_if:N 555, 562
\__zrefclever_opt_bool_if:N(TF) . 20
\__zrefclever_opt_bool_if:NTF ...
..... 555, 888
\__zrefclever_opt_bool_if_set:N .
..... 494, 509
\__zrefclever_opt_bool_if_-
set:N(TF) ..... 18
\__zrefclever_opt_bool_if_-
set:NTF ..... 494,
546, 557, 1447, 1463, 1485, 1501
\__zrefclever_opt_bool_set_-
false:N 19, 510, 520, 529, 2570, 2841
\__zrefclever_opt_bool_set_-
true:N 19, 510, 510, 519, 2565, 2832
\__zrefclever_opt_bool_unset:N ..
..... 18, 483, 483, 493, 2575, 2850
\__zrefclever_opt_seq_get:NN 472, 482
\__zrefclever_opt_seq_get:NN(TF) 18
\__zrefclever_opt_seq_get:NNTF ..
... 472, 818, 855, 936, 949, 2869,
2882, 5262, 5267, 5272, 5277, 5282
\__zrefclever_opt_seq_gset_-
clist_split:Nn ..... 16,
424, 426, 1376, 1409, 3301, 3326
\__zrefclever_opt_seq_gset_eq:NN
..... 16, 424,
438, 444, 1385, 1418, 2950, 3310, 3335
\__zrefclever_opt_seq_if_set:N ..
..... 456, 471
\__zrefclever_opt_seq_if_-
set:N(TF) ..... 17
\__zrefclever_opt_seq_if_set:NTF
..... 456, 474, 1027, 1369, 1401
\__zrefclever_opt_opt_seq_set_clist_-
split:Nn .. 16, 424, 424, 2538, 2802
\__zrefclever_opt_seq_set_eq:NN .
..... 16, 424, 428, 437, 2544, 2808
\__zrefclever_opt_seq_unset:N ...
..... 17, 445, 445, 455, 2533, 2794
\__zrefclever_opt_tl_clear:N ...
..... 14, 334,
344, 353, 1645, 1650, 1665, 1680,
1695, 2653, 2658, 2673, 2688, 2703
\__zrefclever_opt_tl_cset_-
fallback:n ..... 1524, 1531
\__zrefclever_opt_tl_gclear:N ...
..... 14, 334,
361, 367, 3028, 3034, 3042, 3049,
3070, 3086, 3107, 3123, 3144, 3160
\__zrefclever_opt_tl_gclear_if_-
new:N ..... 16, 393,
403, 412, 1125, 1131, 1139, 1146,
1167, 1183, 1204, 1220, 1241, 1257
\__zrefclever_opt_tl_get:NN 413, 423
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF ...
413, 5011, 5026, 5045, 5054, 5075,
5085, 5229, 5234, 5239, 5244, 5249
\__zrefclever_opt_tl_gset:N .... 14
\__zrefclever_opt_tl_gset:Nn ...
.. 334, 354, 360, 2973, 2997, 3005,
3063, 3078, 3100, 3115, 3137, 3152,
3185, 3192, 3201, 3209, 3266, 3276
\__zrefclever_opt_tl_gset_if_-
new:Nn ..... 16,
393, 393, 402, 1068, 1093, 1102,
1160, 1175, 1197, 1212, 1234, 1249,
1282, 1289, 1298, 1306, 1336, 1346
\__zrefclever_opt_tl_if_set:N .. 379
\__zrefclever_opt_tl_if_set:N(TF)
..... 15
\__zrefclever_opt_tl_if_set:NTF .
..... 379, 395, 405, 415
\__zrefclever_opt_tl_set:N ..... 14
\__zrefclever_opt_tl_set:Nn ....
..... 334, 334, 343,
1659, 1674, 1689, 1729, 1735, 2489,
2635, 2667, 2682, 2697, 2737, 2744
\__zrefclever_opt_tl_unset:N ...
..... 15, 368, 368,
378, 1704, 1709, 2485, 2628, 2712, 2717
\__zrefclever_opt_var_set_bool:n
..... 14, 332, 332, 339,
340, 341, 349, 350, 351, 373, 374,
375, 383, 385, 433, 434, 435, 450,
451, 452, 460, 462, 488, 489, 490,
498, 500, 515, 516, 517, 525, 526, 527
\__zrefclever_opt_varname_-
fallback:n ..... 14,

```

[330](#), [330](#), [1527](#), [5250](#), [5283](#), [5316](#)
`__zrefclever_opt_varname_-`
 `general:nn` [12](#),
 [291](#), [291](#), [1647](#), [1652](#), [1661](#), [1667](#),
 [1676](#), [1682](#), [1691](#), [1697](#), [1706](#), [1711](#),
 [1731](#), [1737](#), [2486](#), [2490](#), [2534](#), [2545](#),
 [2566](#), [2571](#), [2576](#), [5230](#), [5263](#), [5296](#)
`__zrefclever_opt_varname_lang_-`
 `default:nnn` .. [13](#), [307](#), [307](#), [317](#),
 [1070](#), [1095](#), [1127](#), [1133](#), [1162](#), [1169](#),
 [1199](#), [1206](#), [1236](#), [1243](#), [1284](#), [1291](#),
 [1371](#), [1387](#), [1449](#), [1456](#), [1487](#), [1494](#),
 [2975](#), [2999](#), [3030](#), [3036](#), [3065](#), [3072](#),
 [3102](#), [3109](#), [3139](#), [3146](#), [3187](#), [3194](#),
 [3312](#), [3364](#), [3385](#), [5245](#), [5278](#), [5311](#)
`__zrefclever_opt_varname_lang_-`
 `type:nnnn` [13](#),
 [318](#), [318](#), [329](#), [1029](#), [1038](#), [1046](#),
 [1104](#), [1141](#), [1148](#), [1177](#), [1185](#), [1214](#),
 [1222](#), [1251](#), [1259](#), [1300](#), [1308](#), [1338](#),
 [1348](#), [1403](#), [1420](#), [1465](#), [1473](#), [1503](#),
 [1511](#), [2952](#), [3007](#), [3044](#), [3051](#), [3080](#),
 [3088](#), [3117](#), [3125](#), [3154](#), [3162](#), [3203](#),
 [3211](#), [3268](#), [3278](#), [3337](#), [3372](#), [3393](#),
 [5028](#), [5077](#), [5087](#), [5240](#), [5273](#), [5306](#)
`__zrefclever_opt_varname_-`
 `language:nnn` [13](#), [296](#),
 [296](#), [306](#), [771](#), [776](#), [786](#), [791](#), [802](#),
 [807](#), [820](#), [857](#), [890](#), [938](#), [951](#), [2871](#), [2884](#)
`__zrefclever_opt_varname_-`
 `type:nnn` [12](#), [293](#), [293](#), [295](#), [2630](#),
 [2637](#), [2655](#), [2660](#), [2669](#), [2675](#), [2684](#),
 [2690](#), [2699](#), [2705](#), [2714](#), [2719](#), [2739](#),
 [2746](#), [2796](#), [2810](#), [2834](#), [2843](#), [2852](#),
 [5013](#), [5047](#), [5056](#), [5235](#), [5268](#), [5301](#)
`\g__zrefclever_page_format_int` ..
 [106](#), [112](#), [116](#)
`\l__zrefclever_pairsep_tl`
 [3753](#), [3904](#), [4027](#),
 [4050](#), [4076](#), [4128](#), [4159](#), [4198](#), [4281](#)
`\g__zrefclever_prev_page_format_-`
 `tl` [7](#), [107](#), [110](#), [113](#)
`__zrefclever_process_language_-`
 `settings:` ... [57](#), [59](#), [813](#), [813](#), [3414](#)
`__zrefclever_prop_put_non_-`
 `empty:Nnn` [43](#), [1543](#), [1543](#), [2340](#), [2395](#)
`__zrefclever_provide_langfile:n`
 [21](#),
 [30](#), [32](#), [84](#), [915](#), [915](#), [971](#), [2122](#), [3413](#)
`\l__zrefclever_range_beg_is_-`
 `first_bool` [3746](#),
 [3796](#), [4115](#), [4151](#), [4185](#), [4456](#),
 [4493](#), [4549](#), [4594](#), [4616](#), [4645](#), [4698](#)
`\l__zrefclever_range_beg_label_-`
 `tl` [93](#),
 [3746](#), [3789](#), [4078](#), [4091](#), [4130](#), [4139](#),
 [4169](#), [4200](#), [4214](#), [4224](#), [4449](#), [4494](#),
 [4511](#), [4562](#), [4608](#), [4630](#), [4659](#), [4673](#)
`\l__zrefclever_range_count_int` ..
 [93](#),
 [3746](#), [3794](#), [4040](#), [4104](#), [4454](#), [4497](#),
 [4508](#), [4514](#), [4520](#), [4528](#), [4587](#), [4700](#)
`\l__zrefclever_range_end_ref_tl` .
 ... [3746](#), [3790](#), [4226](#), [4232](#), [4301](#),
 [4307](#), [4450](#), [4496](#), [4513](#), [4675](#), [4681](#)
`\l__zrefclever_range_same_count_-`
 `int` [93](#),
 [3746](#), [3795](#), [4018](#), [4069](#), [4105](#), [4455](#),
 [4499](#), [4516](#), [4522](#), [4567](#), [4588](#), [4701](#)
`\l__zrefclever_rangeseptl`
 [3753](#), [3916](#),
 [4229](#), [4239](#), [4304](#), [4314](#), [4678](#), [4688](#)
`\l__zrefclever_rangetopair_bool` .
 [3753](#), [3944](#), [4265](#)
`\l__zrefclever_ref_count_int` ...
 [3731](#), [3793](#),
 [4046](#), [4124](#), [4194](#), [4452](#), [4485](#), [4534](#),
 [4557](#), [4569](#), [4603](#), [4625](#), [4634](#), [4654](#)
`\l__zrefclever_ref_decl_case_tl` .
 [27](#), [827](#), [832](#), [833](#), [837](#), [840](#),
 [844](#), [848](#), [851](#), [896](#), [899](#), [901](#), [2163](#),
 [2173](#), [5020](#), [5024](#), [5063](#), [5068](#), [5073](#)
`__zrefclever_ref_default:` [4707](#),
 [4707](#), [4748](#), [4754](#), [4792](#), [4833](#), [4899](#)
`\l__zrefclever_ref_gender_tl` ...
 [28](#), [864](#), [870](#),
 [871](#), [875](#), [878](#), [883](#), [884](#), [903](#), [909](#),
 [910](#), [2182](#), [2246](#), [4976](#), [4984](#), [4990](#), [4998](#)
`\l__zrefclever_ref_language_tl` ..
 [24](#), [27](#), [55](#), [715](#), [719](#), [816](#),
 [821](#), [831](#), [849](#), [858](#), [868](#), [882](#), [891](#),
 [900](#), [907](#), [2104](#), [2110](#), [2115](#), [2123](#),
 [2141](#), [2147](#), [2152](#), [3413](#), [3431](#), [3802](#),
 [3806](#), [3810](#), [3899](#), [3903](#), [3907](#), [3911](#),
 [3915](#), [3919](#), [3923](#), [3927](#), [3931](#), [3935](#),
 [3939](#), [3943](#), [3947](#), [3951](#), [3955](#), [3959](#),
 [3963](#), [3967](#), [3971](#), [3975](#), [3979](#), [3983](#),
 [4980](#), [4992](#), [5003](#), [5029](#), [5078](#), [5088](#)
`\l__zrefclever_ref_property_tl` ..
 [43](#), [48](#), [1549](#),
 [1558](#), [1565](#), [1567](#), [1751](#), [1775](#), [1819](#),
 [1836](#), [1848](#), [1855](#), [1860](#), [1893](#), [1900](#),
 [1905](#), [1935](#), [1942](#), [1947](#), [3506](#), [3829](#),
 [3884](#), [3888](#), [4713](#), [4798](#), [4852](#), [5161](#)
`\l__zrefclever_ref_propserity_tl` [3474](#)
`\l__zrefclever_ref_typeset_font_-`
 `tl` [2250](#), [2252](#), [3424](#)

\l_zrefclever_refbounds_first_- pb_seq	3768, 3956, 4032, 4082, 4154, 4205, 4286	1388, 1404, 1421, 1450, 1457, 1466, 1474, 1488, 1495, 1504, 1512, 2868, 2909, 2916, 2930, 2947, 2953, 2976,
\l_zrefclever_refbounds_first_- rb_seq	3768, 3960, 4188, 4320, 4649	3000, 3008, 3031, 3037, 3045, 3052, 3066, 3073, 3081, 3089, 3103, 3110,
\l_zrefclever_refbounds_first_- seq	3768, 3948, 4331, 4552, 4598, 4620	3118, 3126, 3140, 3147, 3155, 3163, 3188, 3195, 3204, 3212, 3269, 3279,
\l_zrefclever_refbounds_first_- sg_seq	3768, 3952, 4010, 4021, 4118	3313, 3338, 3365, 3373, 3386, 3394
\l_zrefclever_refbounds_last_- pe_seq	3768, 3980, 4029, 4052, 4079, 4131, 4161, 4283	\l_zrefclever_setup_type_tl 563, 935, 977, 978, 1010, 1031, 1040, 1048, 1066, 1091, 1106, 1123, 1143, 1150, 1158, 1179, 1187, 1195, 1216, 1224, 1232, 1253, 1261, 1280, 1302, 1310, 1328, 1340, 1350, 1367, 1405, 1422, 1445, 1467, 1475, 1483, 1505, 1513, 2602, 2604, 2631, 2638, 2656, 2661, 2670, 2676, 2685, 2691, 2700, 2706, 2715, 2720, 2740, 2747, 2797, 2811, 2835, 2844, 2853, 2867, 2900, 2901, 2933, 2954, 2971, 2995, 3009, 3026, 3046, 3053, 3061, 3082, 3090, 3098, 3119, 3127, 3135, 3156, 3164, 3183, 3205, 3213, 3258, 3270, 3280, 3298, 3339, 3360, 3374, 3381, 3395
\l_zrefclever_refbounds_last_- re_seq	3768, 3984, 4233, 4241, 4308, 4316	\l_zrefclever_sort_decided_bool 3467, 3550, 3564, 3574, 3578, 3590, 3600, 3615, 3630, 3654
\l_zrefclever_refbounds_last_- seq	3768, 3976, 4060, 4095, 4140, 4176	_zrefclever_sort_default:nn 88, 3508, 3524, 3524
\l_zrefclever_refbounds_mid_rb_- seq	3768, 3968, 4201, 4215, 4660	_zrefclever_sort_default_- different_types:nn 44, 86, 91, 3535, 3667, 3667
\l_zrefclever_refbounds_mid_re_- seq	3768, 3972, 4682, 4690	_zrefclever_sort_default_same_- type:nn 86, 88, 3534, 3537, 3537
\l_zrefclever_refbounds_mid_seq	3768, 3964, 4092, 4170, 4539, 4563, 4575, 4609, 4631, 4639	_zrefclever_sort_labels: 86-88, 92, 3422, 3471, 3471
\l_zrefclever_reffont_tl	3753, 3924, 4728, 4740, 4769, 4780, 4818, 4875, 4890	_zrefclever_sort_page:nn 92, 3507, 3719, 3719
\l_zrefclever_reftype_override_- tl	34, 44, 2322, 2325	\l_zrefclever_sort_prior_a_int 3468, 3669, 3675, 3676, 3682, 3692, 3700
\g_zrefclever_rf_opts_bool_- maybe_type_specific_seq	52, 53, 568, 1438, 2558, 2825, 3353	\l_zrefclever_sort_prior_b_int 3468, 3670, 3677, 3678, 3685, 3693, 3701
\g_zrefclever_rf_opts_seq_- refbounds_seq	568, 1360, 2524, 2785, 3291	\l_zrefclever_tlastsep_tl 3753, 3811, 4430
\g_zrefclever_rf_opts_tl_maybe_- type_specific_seq	568, 1084, 2988	\l_zrefclever_tlistsep_tl 3753, 3807, 4401
\g_zrefclever_rf_opts_tl_not_- type_specific_seq	568, 1059, 2607, 2964	\l_zrefclever_tpairsep_tl 3753, 3803, 4423
\g_zrefclever_rf_opts_tl_- reference_seq	568, 2476	\l_zrefclever_type_count_int 93, 118, 3731, 3792, 4398, 4400, 4413, 4438, 4453, 4924, 4936, 5131
\g_zrefclever_rf_opts_tl_type_- names_seq	568, 1321, 3251	
\g_zrefclever_rf_opts_tl_- typesetup_seq	568, 2619	
\l_zrefclever_setup_language_tl	563, 744, 772, 777, 787, 792, 803, 808, 934, 986, 993, 1007, 1024, 1030, 1039, 1047, 1071, 1096, 1105, 1128, 1134, 1142, 1149, 1163, 1170, 1178, 1186, 1200, 1207, 1215, 1223, 1237, 1244, 1252, 1260, 1285, 1292, 1301, 1309, 1339, 1349, 1372,	

<ul style="list-style-type: none"> \l_zrefclever_type_first_label_- tl 93, 115, 3734, 3787, 4005, 4256, 4267, 4271, 4299, 4350, 4367, 4371, 4447, 4481, 4791, 4797, 4803, 4807, 4820, 4851, 4865, 4869, 4877, 4892, 4905 \l_zrefclever_type_first_label_- type_tl ... 93, 118, 3734, 3788, 4007, 4260, 4448, 4483, 4912, 4955, 4971, 4979, 4991, 4997, 5014, 5030, 5040, 5048, 5057, 5079, 5089, 5101 \l_zrefclever_type_first_- refbounds_seq 3768, 4009, 4020, 4031, 4081, 4117, 4153, 4187, 4204, 4285, 4319, 4330, 4351, 4551, 4597, 4619, 4648, 4815, 4816, 4823, 4825, 4861, 4873, 4881, 4884, 4887, 4888, 4895, 4896 \l_zrefclever_type_first_- refbounds_set_bool 3768, 3797, 4011, 4022, 4033, 4084, 4120, 4156, 4190, 4207, 4287, 4321, 4328, 4457, 4554, 4600, 4622, 4651 \l_zrefclever_type_name_gender_- seq ... 3740, 4981, 4983, 4986, 5001 \l_zrefclever_type_name_- missing_bool 3740, 4837, 4908, 4915, 5037, 5097 _zrefclever_type_name_setup: 20, 22, 115, 4337, 4903, 4903 \l_zrefclever_type_name_tl 115, 118, 3740, 4374, 4380, 4812, 4830, 4845, 4847, 4907, 4914, 5018, 5034, 5036, 5052, 5061, 5083, 5093, 5095, 5115 \l_zrefclever_typeset_compress_- bool 1627, 1630, 4466 \l_zrefclever_typeset_labels_- seq 92, 3728, 3783, 3817, 3819, 3825 \l_zrefclever_typeset_last_bool 93, 3728, 3814, 3815, 3822, 3847, 4410, 5130 \l_zrefclever_typeset_name_bool ... 1576, 1583, 1588, 1593, 4339, 4355 \l_zrefclever_typeset_queue_- curr_tl 93, 	<ul style="list-style-type: none"> 95, 113, 118, 3734, 3786, 4025, 4048, 4056, 4074, 4087, 4126, 4135, 4157, 4165, 4172, 4196, 4210, 4227, 4237, 4254, 4279, 4302, 4312, 4341, 4348, 4358, 4391, 4407, 4418, 4424, 4431, 4445, 4446, 4535, 4558, 4570, 4604, 4626, 4635, 4655, 4676, 4686, 4929, 4951, 4962, 5125, 5129 \l_zrefclever_typeset_queue_- prev_tl . 93, 3734, 3785, 4402, 4444 \l_zrefclever_typeset_range_- bool ... 1761, 1979, 1982, 3421, 4252 \l_zrefclever_typeset_ref_bool 1575, 1582, 1587, 1592, 4339, 4345 _zrefclever_typeset_refs: 92, 94, 95, 3425, 3781, 3781 _zrefclever_typeset_refs_last_- of_type: 99, 113, 115, 118, 3990, 3995, 3995 _zrefclever_typeset_refs_not_- last_of_type: 93, 100, 113, 123, 3992, 4460, 4460 \l_zrefclever_typeset_sort_bool 1603, 1606, 3420 \l_zrefclever_typeset_sort_seq 44, 91, 1612, 1617, 1618, 1624, 3671 \l_zrefclever_verbose_testing_- bool 3780, 4406 _zrefclever_zcref:nnn 27, 56, 3406, 3407 _zrefclever_zcref:nnnn 84, 86, 3407 \l_zrefclever_zcref_labels_seq 86, 87, 3411, 3442, 3447, 3451, 3476, 3479, 3784 \l_zrefclever_zcref_note_tl 2271, 2274, 3427, 3434 \l_zrefclever_zcref_with_check_- bool 2278, 2295, 3417, 3438 _zrefclever_zcsetup:n 68, 2596, 2597, 2597, 2599, 5329, 5353, 5359, 5367, 5389, 5408, 5426, 5444, 5478, 5529, 5554, 5566, 5576, 5591, 5611, 5634 \l_zrefclever_zrefcheck_- available_bool 2277, 2290, 2302, 2314, 3416, 3437
--	---