

The zref-clever package*

Code documentation

Gustavo Barros[†]

2022-01-28

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	7
4.1	Auxiliary	7
4.2	Messages	8
4.3	Data extraction	10
4.4	Option infra	11
4.5	Reference format	16
4.6	Languages	19
4.7	Language files	24
4.8	Options	34
5	Configuration	50
5.1	\zcsetup	50
5.2	\zcRefTypeSetup	51
5.3	\zcLanguageSetup	54
6	User interface	62
6.1	\zceref	62
6.2	\zcpageref	64
7	Sorting	64
8	Typesetting	71

*This file describes v0.2.0-alpha, released 2022-01-28.

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

9	Compatibility	103
9.1	appendix	103
9.2	appendices	104
9.3	memoir	105
9.4	KOMA	107
9.5	amsmath	108
9.6	mathtools	111
9.7	breqn	111
9.8	listings	112
9.9	enumitem	113
9.10	subcaption	114
9.11	subfig	114
10	Language files	115
10.1	English	115
10.2	German	119
10.3	French	127
10.4	Portuguese	132
10.5	Spanish	136
10.6	Dutch	140
	Index	144

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`, even though I'd have loved to have used `\bool_case_true:...`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Finally, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (`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>). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
```

```

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

Identify the package.

14 \ProvidesExplPackage {zref-clever} {2022-01-28} {0.2.0-alpha}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

Required packages. Besides these, `zref-hyperref`, `zref-titleref`, and `zref-check` may also be loaded depending on user options.

```

16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \RequirePackage { zref-abspage }
19 \RequirePackage { l3keys2e }
20 \RequirePackage { ifdraft }

```

3 zref setup

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

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

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

```

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

```

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

```

23 \zref@newprop { thecounter }
24 {
25   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
26   { \use:c { the \l__zrefclever_current_counter_tl } }
27   {
28     \cs_if_exist:cT { c@ \@currentcounter }

```

```

29         { \use:c { the \@currentcounter } }
30     }
31 }
32 \zref@addprop \ZREF@mainlist { thecounter }

```

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

```

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

```

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

```

44 \zref@newprop { zc@cntval } [0]
45 {
46     \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
47     { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
48     {
49         \cs_if_exist:cT { c@ \@currentcounter }
50         { \int_use:c { c@ \@currentcounter } }
51     }
52 }
53 \zref@addprop \ZREF@mainlist { zc@cntval }
54 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
55 \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 `ltcounts.dtx` in `texdoc source2e`). Besides, there may be a chain of resetting counters, which must be taken into account: if `counterC` gets reset by `counterB`, and `counterB` gets reset by `counterA`, stepping the latter affects all three of them.

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

`__zrefclever_get_enclosing_counters_value:n`

Recursively generate a *sequence* of “enclosing counters” values, for a given `⟨counter⟩` 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⟩}

56 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
57 {
58   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }

```

```

59   {
60     { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
61     \_zrefclever_get_enclosing_counters_value:e
62     { \_zrefclever_counter_reset_by:n {#1} }
63   }
64 }

```

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).

```

65 \cs_generate_variant:Nn \_zrefclever_get_enclosing_counters_value:n { e }

```

(End definition for _zrefclever_get_enclosing_counters_value:n.)

_zrefclever_counter_reset_by:n

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

```

\_zrefclever_counter_reset_by:n { \langle counter \rangle }

66 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
67   {
68     \bool_if:nTF
69       { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
70       { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
71       {
72         \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
73         { \_zrefclever_counter_reset_by_aux:nn {#1} }
74       }
75   }
76 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
77   {
78     \cs_if_exist:cT { c@ #2 }
79     {
80       \tl_if_empty:cF { c1@ #2 }
81       {
82         \tl_map_tokens:cn { c1@ #2 }
83         { \_zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
84       }
85     }
86   }
87 \cs_new:Npn \_zrefclever_counter_reset_by_auxi:nnn #1#2#3
88   {
89     \str_if_eq:nnT {#2} {#3}
90     { \tl_map_break:n { \seq_map_break:n {#1} } }
91   }

```

(End definition for _zrefclever_counter_reset_by:n.)

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

```

92 \zref@newprop { zc@enclval }
93 {

```

```

94     \__zrefclever_get_enclosing_counters_value:e
95     \l__zrefclever_current_counter_tl
96   }
97   \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

98 \tl_new:N \g__zrefclever_page_format_tl
99 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
100 \AddToHook { shipout / before }
101 {
102   \group_begin:
103   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
104   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
105   \group_end:
106 }
107 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
108 \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.

```

109 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
110 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
111 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
112 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

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

4.2 Messages

```

113 \msg_new:nnn { zref-clever } { option-not-type-specific }
114 {
115   Option~'#1'~is-not-type-specific~\msg_line_context:~
116   Set-it-in~'\iow_char:N\zcLanguageSetup'~before-first~'type'~
117   switch-or-as~package-option.
118 }
119 \msg_new:nnn { zref-clever } { option-only-type-specific }
120 {
121   No~type~specified-for~option~'#1'~\msg_line_context:~
122   Set-it-after~'type'~switch.
123 }
124 \msg_new:nnn { zref-clever } { key-requires-value }
125 { The~'#1'~key~'#2'~requires-a~value~\msg_line_context:~
126 \msg_new:nnn { zref-clever } { language-declared }
127 { Language~'#1'~is-already-declared~\msg_line_context:~Nothing-to-do. }
128 \msg_new:nnn { zref-clever } { unknown-language-alias }
129 {
130   Language~'#1'~is-unknown~\msg_line_context:~Can't~alias~to~it.~
131   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
132   '\iow_char:N\zcDeclareLanguageAlias'.
133 }
134 \msg_new:nnn { zref-clever } { unknown-language-setup }
135 {
136   Language~'#1'~is-unknown~\msg_line_context:~Can't~set~it~up.~
137   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
138   '\iow_char:N\zcDeclareLanguageAlias'.
139 }
140 \msg_new:nnn { zref-clever } { unknown-language-opt }
141 {
142   Language~'#1'~is-unknown~\msg_line_context:~
143   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
144   '\iow_char:N\zcDeclareLanguageAlias'.
145 }
146 \msg_new:nnn { zref-clever } { unknown-language-decl }
147 {
148   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
149   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
150   '\iow_char:N\zcDeclareLanguageAlias'.
151 }
152 \msg_new:nnn { zref-clever } { language-no-decl-ref }
153 {
154   Language~'#1'~has-no-declared-declension~cases~\msg_line_context:~
155   Nothing-to-do-with~option~'d=#2'.
156 }
157 \msg_new:nnn { zref-clever } { language-no-gender }
158 {
159   Language~'#1'~has-no-declared-gender~\msg_line_context:~
160   Nothing-to-do-with~option~'#2=#3'.
161 }
162 \msg_new:nnn { zref-clever } { language-no-decl-setup }

```



```

163 {
164   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
165   Nothing~to~do~with~option~'case=#2'.
166 }
167 \msg_new:nnn { zref-clever } { unknown-decl-case }
168 {
169   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:~
170   Using~default~declension~case.
171 }
172 \msg_new:nnn { zref-clever } { nudge-multitype }
173 {
174   Reference~with~multiple~types~\msg_line_context:~
175   You~may~wish~to~separate~them~or~review~language~around~it.
176 }
177 \msg_new:nnn { zref-clever } { nudge-comptosing }
178 {
179   Multiple~labels~have~been~compressed~into~singular~type~name~
180   for~type~'#1'~\msg_line_context:.
181 }
182 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
183 {
184   Option~'sg'~signals~that~a~singular~type~name~was~expected~
185   \msg_line_context:~But~type~'#1'~has~plural~type~name.
186 }
187 \msg_new:nnn { zref-clever } { gender-not-declared }
188 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:. }
189 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
190 {
191   Gender~mismatch~for~type~'#1'~\msg_line_context:~
192   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
193 }
194 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
195 {
196   You've~specified~'g=#1'~\msg_line_context:~
197   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
198 }
199 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
200 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
201 \msg_new:nnn { zref-clever } { option-document-only }
202 { Option~'#1'~is~only~available~after~\iow_char:N\begin\{document\}. }
203 \msg_new:nnn { zref-clever } { langfile-loaded }
204 { Loaded~'#1'~language~file. }
205 \msg_new:nnn { zref-clever } { zref-property-undefined }
206 {
207   Option~'ref=#1'~requested~\msg_line_context:~
208   But~the~property~'#1'~is~not~declared,~falling~back~to~'default'.
209 }
210 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
211 {
212   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:~
213   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
214   '\iow_char:N\zcref'.
215 }
216 \msg_new:nnn { zref-clever } { missing-hyperref }

```

```

217 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
218 \msg_new:nnn { zref-clever } { titleref-preamble-only }
219 {
220   Option~'titleref'~only~available~in~the~preamble~\msg_line_context:..
221   Did~you~mean~'ref=title'?.
222 }
223 \msg_new:nnn { zref-clever } { option-preamble-only }
224 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:.. }
225 \msg_new:nnn { zref-clever } { unknown-compat-module }
226 {
227   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
228   Nothing~to~do.
229 }
230 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
231 {
232   The~value~of~option~'#1'~must~be~a~comma~separated~list~
233   of~four~items.~We~received~'#2'~items~\msg_line_context:..
234   Option~not~set.
235 }
236 \msg_new:nnn { zref-clever } { missing-zref-check }
237 {
238   Option~'check'~requested~\msg_line_context:..
239   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
240 }
241 \msg_new:nnn { zref-clever } { missing-type }
242 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
243 \msg_new:nnn { zref-clever } { missing-property }
244 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:.. }
245 \msg_new:nnn { zref-clever } { missing-name }
246 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:.. }
247 \msg_new:nnn { zref-clever } { single-element-range }
248 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
249 \msg_new:nnn { zref-clever } { compat-package }
250 { Loaded~support~for~'#1'~package. }
251 \msg_new:nnn { zref-clever } { compat-class }
252 { Loaded~support~for~'#1'~documentclass. }
253 \msg_new:nnn { zref-clever } { option-deprecated }
254 {
255   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
256   Use~'#2'~instead.
257 }

```

4.3 Data extraction

`_zrefclever_extract_default:Nnnn`

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

```

\__zrefclever_extract_default:Nnnn {\langle tl var \rangle}
  {\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

258 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
259 {
260   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
261   { \zref@extractdefault {#2} {#3} {#4} }

```

```

262 }
263 \cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn }

```

(End definition for __zrefclever_extract_default:Nnnn.)

__zrefclever_extract_unexp:nnn Extract property $\langle prop \rangle$ from $\langle label \rangle$. Ensure that, in the context of an x expansion, $\text{\zref@extractdefault}$ 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}

264 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
265 {
266   \exp_args:NNo \exp_args:No
267   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
268 }
269 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvnn , Vvn }

```

(End definition for __zrefclever_extract_unexp:nnn.)

__zrefclever_extract:nnn An internal version for $\text{\zref@extractdefault}$.

```

\__zrefclever_extract:nnn{\label}{\prop}{\default}

270 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
271 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for __zrefclever_extract:nnn.)

4.4 Option infra

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

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

__zrefclever_opt_varname_general:nn Defines, and leaves in the input stream, the csname of the variable used to store the general $\langle option \rangle$. The data type of the variable must be specified (\tt t1 , \tt seq , \tt bool , etc.).

```

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

```

```

272 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
273 { l__zrefclever_opt_general_ #1 _ #2 }

```

(End definition for _zrefclever_opt_varname_general:nn.)

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

```

    \_zrefclever_opt_varname_type:nnn {\ref type} {\option} {\data type}

```

```

274 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
275 { l__zrefclever_opt_type_ #1 _ #2 _ #3 }
276 \cs_generate_variant:Nn \_zrefclever_opt_varname_type:nnn { enn , een }

```

(End definition for _zrefclever_opt_varname_type:nnn.)

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

```

    \_zrefclever_opt_varname_language:nnn {\lang} {\option} {\data type}

```

```

277 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
278 {
279   \_zrefclever_language_if_declared:nTF {#1}
280   {
281     g__zrefclever_opt_language_
282     \tl_use:c { \_zrefclever_language_varname:n {#1} }
283     _ #2 _ #3
284   }
285   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
286 }
287 \cs_generate_variant:Nn \_zrefclever_opt_varname_language:nnn { enn }

```

(End definition for _zrefclever_opt_varname_language:nnn.)

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

```

    \_zrefclever_opt_varname_lang_default:nnn {\lang} {\option} {\data type}

```

```

288 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
289 {
290   \_zrefclever_language_if_declared:nTF {#1}
291   {
292     g__zrefclever_opt_lang_
293     \tl_use:c { \_zrefclever_language_varname:n {#1} }
294     _default_ #2 _ #3
295   }
296   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
297 }
298 \cs_generate_variant:Nn \_zrefclever_opt_varname_lang_default:nnn { enn }

```

(End definition for `_zrefclever_opt_varname_lang_default:nnn`.)

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

```

    \_zrefclever_opt_varname_lang_type:nnnn {\langle lang \rangle} {\langle ref type \rangle}
    {\langle option \rangle} {\langle data type \rangle}
299 \cs_new:Npn \_zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
300 {
301   \_zrefclever_language_if_declared:nTF {#1}
302   {
303     g\_zrefclever_opt_lang_
304     \tl_use:c { \_zrefclever_language_varname:n {#1} }
305     _type_ #2 _ #3 _ #4
306   }
307   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
308 }
309 \cs_generate_variant:Nn
310 \_zrefclever_opt_varname_lang_type:nnnn { eenn , eenen }

```

(End definition for `_zrefclever_opt_varname_lang_type:nnnn`.)

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

```

    \_zrefclever_opt_varname_fallback:nn {\langle option \rangle} {\langle data type \rangle}
311 \cs_new:Npn \_zrefclever_opt_varname_fallback:nn #1#2
312 { c\_zrefclever_opt_fallback_ #1 _ #2 }

```

(End definition for `_zrefclever_opt_varname_fallback:nn`.)

`_zrefclever_opt_tl_unset:N` Unset $\langle option tl \rangle$. These functions *define* what means to be unset for an option token list, and it must match what the conditional `_zrefclever_opt_tl_if_set:N` tests for.

`_zrefclever_opt_tl_gunset:N`

```

    \_zrefclever_opt_tl_unset:N {\langle option tl \rangle}
    \_zrefclever_opt_tl_gunset:N {\langle option tl \rangle}
313 \cs_new_protected:Npn \_zrefclever_opt_tl_unset:N #1
314 { \tl_set_eq:NN #1 \c_novalue_tl }
315 \cs_new_protected:Npn \_zrefclever_opt_tl_gunset:N #1
316 { \tl_gset_eq:NN #1 \c_novalue_tl }
317 \cs_generate_variant:Nn \_zrefclever_opt_tl_unset:N { c }
318 \cs_generate_variant:Nn \_zrefclever_opt_tl_gunset:N { c }

```

(End definition for `_zrefclever_opt_tl_unset:N` and `_zrefclever_opt_tl_gunset:N`.)

`_zrefclever_opt_tl_if_set:NTF`

```

    \_zrefclever_opt_tl_if_set:N(TF) {\langle option tl \rangle} {\langle true \rangle} {\langle false \rangle}
319 \prg_new_conditional:Npnn \_zrefclever_opt_tl_if_set:N #1 { F , TF }
320 {
321   \bool_lazy_and:nnTF
322   { \tl_if_exist_p:N #1 }
323   { ! \tl_if_no_value_p:n {#1} }
324   { \prg_return_true: }
325   { \prg_return_false: }
326 }

```

(End definition for `_zrefclever_opt_tl_if_set:NTF`.)

```
\_zrefclever_opt_tl_gset_if_new:Nn      \_zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
327 \cs_new_protected:Npn \_zrefclever_opt_tl_gset_if_new:Nn #1#2
328 {
329   \_zrefclever_opt_tl_if_set:NF #1
330   { \tl_gset:Nn #1 {#2} }
331 }
332 \cs_generate_variant:Nn \_zrefclever_opt_tl_gset_if_new:Nn { cn }
```

(End definition for `_zrefclever_opt_tl_gset_if_new:Nn`.)

```
\_zrefclever_opt_tl_get:NNTF      \_zrefclever_opt_tl_get:NN(TF) {<option tl to get>} {<tl var to set>}
                                   {<true>} {<false>}
333 \prg_new_protected_conditional:Npnn \_zrefclever_opt_tl_get:NN #1#2 { F }
334 {
335   \_zrefclever_opt_tl_if_set:NTF #1
336   {
337     \tl_set_eq:NN #2 #1
338     \prg_return_true:
339   }
340   { \prg_return_false: }
341 }
342 \prg_generate_conditional_variant:Nnn \_zrefclever_opt_tl_get:NN { cn } { F }
343
```

(End definition for `_zrefclever_opt_tl_get:NNTF`.)

```
\_zrefclever_opt_seq_set_clist_split:Nn \_zrefclever_opt_seq_set_clist_split:Nn {<option seq>} {<value>}
\_zrefclever_opt_seq_gset_clist_split:Nn \_zrefclever_opt_seq_gset_clist_split:Nn {<option seq>} {<value>}
344 \cs_new_protected:Npn \_zrefclever_opt_seq_set_clist_split:Nn #1#2
345 { \seq_set_split:Nnn #1 { , } {#2} }
346 \cs_new_protected:Npn \_zrefclever_opt_seq_gset_clist_split:Nn #1#2
347 { \seq_gset_split:Nnn #1 { , } {#2} }
```

(End definition for `_zrefclever_opt_seq_set_clist_split:Nn` and `_zrefclever_opt_seq_gset_clist_split:Nn`.)

`_zrefclever_opt_seq_unset:N` Unset `<option seq>`. These functions *define* what means to be unset for an option sequence, and it must match what the conditional `_zrefclever_opt_seq_if_set:N` tests for.

```
\_zrefclever_opt_seq_unset:N {<option seq>}
\_zrefclever_opt_seq_gunset:N {<option seq>}
348 \cs_new_protected:Npn \_zrefclever_opt_seq_unset:N #1
349 { \cs_set_eq:NN #1 \scan_stop: }
350 \cs_new_protected:Npn \_zrefclever_opt_seq_gunset:N #1
351 { \cs_gset_eq:NN #1 \scan_stop: }
352 \cs_generate_variant:Nn \_zrefclever_opt_seq_unset:N { c }
353 \cs_generate_variant:Nn \_zrefclever_opt_seq_gunset:N { c }
```

(End definition for `_zrefclever_opt_seq_unset:N` and `_zrefclever_opt_seq_gunset:N`.)

```

\__zrefclever_opt_seq_if_set:N $\overline{TF}$ 
\__zrefclever_opt_seq_if_set:N(TF) {\langle option seq \rangle} {\langle true \rangle} {\langle false \rangle}
354 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
355 { \seq_if_exist:NTF #1 { \prg_return_true: } { \prg_return_false: } }
356 \prg_generate_conditional_variant:Nnn
357 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

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

```

\__zrefclever_opt_seq_get:N $\overline{NTF}$ 
\__zrefclever_opt_seq_get:N(TF) {\langle option seq to get \rangle} {\langle seq var to set \rangle}
{\langle true \rangle} {\langle false \rangle}
358 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:N #1#2 { F }
359 {
360 \__zrefclever_opt_seq_if_set:NTF #1
361 {
362 \seq_set_eq:NN #2 #1
363 \prg_return_true:
364 }
365 { \prg_return_false: }
366 }
367 \prg_generate_conditional_variant:Nnn
368 \__zrefclever_opt_seq_get:N { cN } { F }

```

(End definition for __zrefclever_opt_seq_get:N \overline{NTF} .)

__zrefclever_opt_bool_unset:N Unset $\langle option bool \rangle$. These functions *define* what means to be unset for an option boolean, and it must match what the conditional __zrefclever_opt_bool_if_set:N tests for. The particular definition we are employing here has some relevant implications. Setting the boolean variable to \scan_stop: (aka, \relax) means we can *never* test the variable without first testing if it is *set*. __zrefclever_opt_bool_if:N does this conveniently.

```

\__zrefclever_opt_bool_unset:N {\langle option bool \rangle}
\__zrefclever_opt_bool_gunset:N {\langle option bool \rangle}
369 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
370 { \cs_set_eq:NN #1 \scan_stop: }
371 \cs_new_protected:Npn \__zrefclever_opt_bool_gunset:N #1
372 { \cs_gset_eq:NN #1 \scan_stop: }
373 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }
374 \cs_generate_variant:Nn \__zrefclever_opt_bool_gunset:N { c }

```

(End definition for __zrefclever_opt_bool_unset:N and __zrefclever_opt_bool_gunset:N.)

```

\__zrefclever_opt_bool_if_set:N $\overline{TF}$ 
\__zrefclever_opt_bool_if_set:N(TF) {\langle option bool \rangle} {\langle true \rangle} {\langle false \rangle}
375 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
376 { \bool_if_exist:NTF #1 { \prg_return_true: } { \prg_return_false: } }
377 \prg_generate_conditional_variant:Nnn
378 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_bool_if_set:N \overline{TF} .)

```

\__zrefclever_opt_bool_get:NNTF      \__zrefclever_opt_bool_get:NN(TF) {\langle option bool to get \rangle} {\langle bool var to set \rangle}
                                     {\langle true \rangle} {\langle false \rangle}

379 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
380 {
381   \__zrefclever_opt_bool_if_set:NTF #1
382   {
383     \bool_set_eq:NN #2 #1
384     \prg_return_true:
385   }
386   { \prg_return_false: }
387 }
388 \prg_generate_conditional_variant:Nnn
389   \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End definition for `__zrefclever_opt_bool_get:NTF`.)

```

\__zrefclever_opt_bool_if:NTF      \__zrefclever_opt_bool_if:N(TF) {\langle option bool \rangle} {\langle true \rangle} {\langle false \rangle}

390 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
391 {
392   \__zrefclever_opt_bool_if_set:NTF #1
393   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
394   { \prg_return_false: }
395 }
396 \prg_generate_conditional_variant:Nnn
397   \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

(End definition for `__zrefclever_opt_bool_if:NTF`.)

4.5 Reference format

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

The fact that we have multiple scopes to set reference format options has some implications for how we handle these options, and for the resulting UI. Since there is a clear precedence rule between the different levels, setting an option at a high priority level shadows everything below it. Hence, it may be relevant to be able to “unset” these options too, so as to be able go back to the lower precedence level of the language-specific options at any given point. However, since many of these options are token lists, or clists, for which “empty” is a legitimate value, we cannot rely on emptiness to distinguish that particular intention. How to deal with it, depends on the kind of option (its data type, to be precise). For token lists and clists/sequences, we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit in `\keys_define:nn` by means of the `.default:x` 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>. 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. In the language files the “unsetting” behavior is less meaningful, since they only change any variable if it is unset to start with, so that unsetting an unset variable would be redundant. However, for UI symmetry also in the language files keys with no value should not be considered “empty” and boolean `unset` values should exist. They are just no-op.

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

```
\l__zrefclever_setup_language_tl
\l__zrefclever_lang_decl_case_tl
\l__zrefclever_lang_declension_seq
\l__zrefclever_lang_gender_seq
398 \tl_new:N \l__zrefclever_setup_type_tl
399 \tl_new:N \l__zrefclever_setup_language_tl
400 \tl_new:N \l__zrefclever_lang_decl_case_tl
401 \seq_new:N \l__zrefclever_lang_declension_seq
402 \seq_new:N \l__zrefclever_lang_gender_seq
```

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

`zrefclever_rf_opts_tl_not_type_specific_seq` Lists of reference format options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent.

```
\c__zrefclever_rf_opts_seq_refbounds_seq
\c__zrefclever_rf_opts_tl_type_names_seq
\c__zrefclever_rf_opts_tl_font_seq
\c__zrefclever_rf_opts_tl_typesetup_seq
\c__zrefclever_rf_opts_tl_reference_seq
403 \seq_const_from_clist:Nn
404 \c__zrefclever_rf_opts_tl_not_type_specific_seq
405 {
406   tpairsep ,
407   tlistsep ,
408   tlastsep ,
409   notesep ,
410 }
411 \seq_const_from_clist:Nn
412 \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
413 {
414   namesep ,
415   pairsep ,
416   listsep ,
417   lastsep ,
418   rangesep ,
419   namefont ,
420   reffont ,
421 }
422 \seq_const_from_clist:Nn
423 \c__zrefclever_rf_opts_seq_refbounds_seq
424 {
425   refbounds-first ,
426   refbounds-first-sg ,
427   refbounds-first-pb ,
428   refbounds-first-rb ,
429   refbounds-mid ,
430   refbounds-mid-rb ,
431   refbounds-mid-re ,
432   refbounds-last ,
433   refbounds-last-pe ,
```

```

434     refbounds-last-re ,
435   }
436 \seq_const_from_clist:Nn
437   \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
438   {
439     cap ,
440     abbrev ,
441   }

```

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:`.

```

442 \seq_const_from_clist:Nn
443   \c__zrefclever_rf_opts_tl_type_names_seq
444   {
445     Name-sg ,
446     name-sg ,
447     Name-pl ,
448     name-pl ,
449     Name-sg-ab ,
450     name-sg-ab ,
451     Name-pl-ab ,
452     name-pl-ab ,
453   }

```

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

```

454 \seq_new:N \c__zrefclever_rf_opts_tl_typesetup_seq
455 \seq_gconcat:NNN \c__zrefclever_rf_opts_tl_typesetup_seq
456   \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
457   \c__zrefclever_rf_opts_tl_type_names_seq
458 \seq_new:N \c__zrefclever_rf_opts_tl_reference_seq
459 \seq_gconcat:NNN \c__zrefclever_rf_opts_tl_reference_seq
460   \c__zrefclever_rf_opts_tl_not_type_specific_seq
461   \c__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

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

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

```

462 \clist_map_inline:nn
463   {
464     reference ,
465     typesetup ,
466     langsetup ,
467     langfile ,
468   }
469   {
470     \keys_define:nn { zref-clever/ #1 }
471     {
472       +refbounds-first .meta:n =
473       {
474         refbounds-first = {##1} ,
475         refbounds-first-sg = {##1} ,
476         refbounds-first-pb = {##1} ,
477         refbounds-first-rb = {##1} ,

```

```

478     } ,
479     +refbounds-first .default:x = \c_novalue_tl ,
480     +refbounds-mid .meta:n =
481     {
482         refbounds-mid = {##1} ,
483         refbounds-mid-rb = {##1} ,
484         refbounds-mid-re = {##1} ,
485     } ,
486     +refbounds-mid .default:x = \c_novalue_tl ,
487     +refbounds-last .meta:n =
488     {
489         refbounds-last = {##1} ,
490         refbounds-last-pe = {##1} ,
491         refbounds-last-re = {##1} ,
492     } ,
493     +refbounds-last .default:x = \c_novalue_tl ,
494     +refbounds-rb .meta:n =
495     {
496         refbounds-first-rb = {##1} ,
497         refbounds-mid-rb = {##1} ,
498     } ,
499     +refbounds-rb .default:x = \c_novalue_tl ,
500     +refbounds-re .meta:n =
501     {
502         refbounds-mid-re = {##1} ,
503         refbounds-last-re = {##1} ,
504     } ,
505     +refbounds-re .default:x = \c_novalue_tl ,
506     +refbounds .meta:n =
507     {
508         +refbounds-first = {##1} ,
509         +refbounds-mid = {##1} ,
510         +refbounds-last = {##1} ,
511     } ,
512     +refbounds .default:x = \c_novalue_tl ,
513     refbounds .meta:n = { +refbounds = {##1} } ,
514     refbounds .default:x = \c_novalue_tl ,
515 }
516 }

```

4.6 Languages

`_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}
```

```

517 \cs_new:Npn \_zrefclever_language_varname:n #1
518 { g\_zrefclever_declared_language_ #1 _tl }

```

(End definition for `_zrefclever_language_varname:n`.)

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

```

519 \cs_set_eq:NN \zrefclever_language_varname:n
520 \_zrefclever_language_varname:n

```

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

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

```

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

521 \prg_new_conditional:Npnn \__zrefclever_language_if_declared:n #1 { T , F , TF }
522 {
523   \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
524   { \prg_return_true: }
525   { \prg_return_false: }
526 }
527 \prg_generate_conditional_variant:Nnn
528 \__zrefclever_language_if_declared:n { x } { T , F , TF }

```

(End definition for `__zrefclever_language_if_declared:nTF`.)

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

```

529 \prg_set_eq_conditional:Nnn \zrefclever_language_if_declared:n
530 \__zrefclever_language_if_declared:n { TF }

```

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

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

```

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

531 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
532 {
533   \group_begin:
534   \tl_if_empty:nF {#2}
535   {
536     \__zrefclever_language_if_declared:nTF {#2}
537     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
538     {
539       \tl_new:c { \__zrefclever_language_varname:n {#2} }
540       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
541       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
542       \keys_set:nn { zref-clever/declarelang } {#1}

```

```

543     }
544   }
545   \group_end:
546 }
547 \@onlypreamble \zcDeclareLanguage

```

(End 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}

548 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
549 {
550   \tl_if_empty:nF {#1}
551   {
552     \__zrefclever_language_if_declared:nTF {#2}
553     {
554       \tl_gset:cx { \__zrefclever_language_varname:n {#1} }
555       { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
556     }
557     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
558   }
559 }
560 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

561 \keys_define:nn { zref-clever/declarelang }
562 {
563   declension .code:n =
564   {
565     \seq_gset_from_clist:cn
566     {
567       \__zrefclever_opt_varname_language:enn
568       { \l__zrefclever_setup_language_tl } { declension } { seq }
569     }
570     {#1}
571   } ,
572   declension .value_required:n = true ,
573   gender .code:n =
574   {
575     \seq_gset_from_clist:cn
576     {
577       \__zrefclever_opt_varname_language:enn
578       { \l__zrefclever_setup_language_tl } { gender } { seq }
579     }
580     {#1}
581   } ,
582   gender .value_required:n = true ,
583   allcaps .choices:nn =
584   { true , false }
585   {
586     \use:c { bool_gset_ \l_keys_choice_tl :c }

```

```

587     {
588         \__zrefclever_opt_varname_language:enn
589         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
590     }
591 },
592 allcaps .default:n = true ,
593 }

```

`__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.

```

594 \cs_new_protected:Npn \__zrefclever_process_language_settings:
595 {
596     \__zrefclever_language_if_declared:xTF
597     { \l__zrefclever_ref_language_tl }
598     {

```

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.

```

599     \__zrefclever_opt_seq_get:cNF
600     {
601         \__zrefclever_opt_varname_language:enn
602         { \l__zrefclever_ref_language_tl } { declension } { seq }
603     }
604     \l__zrefclever_lang_declension_seq
605     { \seq_clear:N \l__zrefclever_lang_declension_seq }
606     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
607     {
608         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
609         {
610             \msg_warning:nxxx { zref-clever }
611             { language-no-decl-ref }
612             { \l__zrefclever_ref_language_tl }
613             { \l__zrefclever_ref_decl_case_tl }
614             \tl_clear:N \l__zrefclever_ref_decl_case_tl
615         }
616     }
617     {
618         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
619         {
620             \seq_get_left:NN \l__zrefclever_lang_declension_seq
621             \l__zrefclever_ref_decl_case_tl
622         }
623     }

```

```

624         \seq_if_in:NVF \l__zrefclever_lang_declension_seq
625         \l__zrefclever_ref_decl_case_tl
626         {
627             \msg_warning:nxxx { zref-clever }
628             { unknown-decl-case }
629             { \l__zrefclever_ref_decl_case_tl }
630             { \l__zrefclever_ref_language_tl }
631             \seq_get_left:NN \l__zrefclever_lang_declension_seq
632             \l__zrefclever_ref_decl_case_tl
633         }
634     }
635 }

```

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.

```

636     \__zrefclever_opt_seq_get:cNF
637     {
638         \__zrefclever_opt_varname_language:enn
639         { \l__zrefclever_ref_language_tl } { gender } { seq }
640     }
641     \l__zrefclever_lang_gender_seq
642     { \seq_clear:N \l__zrefclever_lang_gender_seq }
643     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
644     {
645         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
646         {
647             \msg_warning:nxxxx { zref-clever }
648             { language-no-gender }
649             { \l__zrefclever_ref_language_tl }
650             { g }
651             { \l__zrefclever_ref_gender_tl }
652             \tl_clear:N \l__zrefclever_ref_gender_tl
653         }
654     }
655     {
656         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
657         {
658             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
659             \l__zrefclever_ref_gender_tl
660             {
661                 \msg_warning:nxxx { zref-clever }
662                 { gender-not-declared }
663                 { \l__zrefclever_ref_language_tl }
664                 { \l__zrefclever_ref_gender_tl }
665                 \tl_clear:N \l__zrefclever_ref_gender_tl
666             }
667         }
668     }

```

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

```

669     \__zrefclever_opt_bool_if:cT
670     {
671         \__zrefclever_opt_varname_language:enn

```

```

672         { \l__zrefclever_ref_language_tl } { allcaps } { bool }
673     }
674     { \keys_set:nn { zref-clever/reference } { cap = true } }
675 }
676 {

```

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

```

677     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
678     {
679         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
680         { \l__zrefclever_ref_decl_case_tl }
681         { \l__zrefclever_ref_language_tl }
682         \tl_clear:N \l__zrefclever_ref_decl_case_tl
683     }
684     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
685     {
686         \msg_warning:nxxx { zref-clever }
687         { language-no-gender }
688         { \l__zrefclever_ref_language_tl }
689         { g }
690         { \l__zrefclever_ref_gender_tl }
691         \tl_clear:N \l__zrefclever_ref_gender_tl
692     }
693 }
694 }

```

(End definition for `__zrefclever_process_language_settings:`)

4.7 Language files

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

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

This design decision has also implications to the *form* the language files assumed. As far as my somewhat impressionistic sampling goes, dictionary or localization files of the most common packages in this area of functionality, are usually a set of commands which perform the relevant definitions and assignments in the preamble or at `begindocument`. This includes `translator`, `translations`, but also `babel`’s `.ldf` files, and `biblatex`’s `.ltx` files. I’m not really well acquainted with this machinery, but as far as I grasp, they all rely on some variation of `\ProvidesFile` and `\input`. And they can be safely `\input` without generating spurious content, because they rely on being loaded before the document has actually started. As far as I can tell, `babel`’s “on the fly” functionality is not based on the `.ldf` files, but on the `.ini` files, and on `\babelprovide`. And the `.ini` files are not in this form, but actually resemble “configuration files” of sorts, which means they are read

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

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

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

```
695 \seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(End definition for `\g__zrefclever_loaded_langfiles_seq`.)

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

```

\__zrefclever_provide_langfile:n {\langle language \rangle}

696 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
697 {
698   \group_begin:
699   \@bsphack
700   \__zrefclever_language_if_declared:nT {#1}
701   {
702     \seq_if_in:NxF
703     \g__zrefclever_loaded_langfiles_seq
704     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
705     {
706       \exp_args:Nx \file_get:nnNTF
707       {
708         zref-clever-
709         \tl_use:c { \__zrefclever_language_varname:n {#1} }
710         .lang
711       }
712       { \ExplSyntaxOn }
713       \l_tmpa_tl
714       {
715         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
716         \tl_clear:N \l__zrefclever_setup_type_tl
717         \__zrefclever_opt_seq_get:cNF
718         {
719           \__zrefclever_opt_varname_language:nnn
720           {#1} { declension } { seq }
721         }
722         \l__zrefclever_lang_declension_seq
723         { \seq_clear:N \l__zrefclever_lang_declension_seq }
724         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq

```

```

725         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
726         {
727             \seq_get_left:NN \l__zrefclever_lang_declension_seq
728             \l__zrefclever_lang_decl_case_tl
729         }
730     \__zrefclever_opt_seq_get:cNF
731     {
732         \__zrefclever_opt_varname_language:nnn
733         {#1} { gender } { seq }
734     }
735     \l__zrefclever_lang_gender_seq
736     { \seq_clear:N \l__zrefclever_lang_gender_seq }
737     \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
738     \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
739     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
740     \msg_info:nnx { zref-clever } { langfile-loaded }
741     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
742 }
743 {

```

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.

```

744         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
745         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
746     }
747 }
748 }
749 \@esphack
750 \group_end:
751 }
752 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }

```

(End definition for `__zrefclever_provide_langfile:n`.)

The set of keys for `zref-clever/langfile`, which is used to process the language files in `__zrefclever_provide_langfile:n`. The no-op cases for each category have their messages sent to “info”. These messages should not occur, as long as the language files are well formed, but they're placed there nevertheless, and can be leveraged in regression tests.

```

753 \keys_define:nn { zref-clever/langfile }
754 {
755     type .code:n =
756     {
757         \tl_if_empty:nTF {#1}
758         { \tl_clear:N \l__zrefclever_setup_type_tl }
759         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
760     } ,
761
762     case .code:n =
763     {
764         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
765         {
766             \msg_info:nnxx { zref-clever } { language-no-decl-setup }
767             { \l__zrefclever_setup_language_tl } {#1}

```

```

768     }
769     {
770         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
771         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
772         {
773             \msg_info:nxxx { zref-clever } { unknown-decl-case }
774             {#1} { \l__zrefclever_setup_language_tl }
775             \seq_get_left:NN \l__zrefclever_lang_declension_seq
776             \l__zrefclever_lang_decl_case_tl
777         }
778     }
779 },
780 case .value_required:n = true ,
781
782 gender .default:x = \c_novalue_tl ,
783 gender .code:n =
784 {
785     \seq_if_empty:NnTF \l__zrefclever_lang_gender_seq
786     {
787         \msg_info:nxxxx { zref-clever } { language-no-gender }
788         { \l__zrefclever_setup_language_tl } { gender } {#1}
789     }
790     {
791         \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
792         {
793             \msg_info:nnn { zref-clever }
794             { option-only-type-specific } { gender }
795         }
796         {
797             \tl_if_novalue:nF {#1}
798             {
799                 \seq_clear:N \l_tmpa_seq
800                 \clist_map_inline:nn {#1}
801                 {
802                     \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
803                     { \seq_put_right:Nn \l_tmpa_seq {##1} }
804                     {
805                         \msg_info:nxxx { zref-clever }
806                         { gender-not-declared }
807                         { \l__zrefclever_setup_language_tl } {##1}
808                     }
809                 }
810             }
811             \__zrefclever_opt_seq_if_set:cF
812             {
813                 \__zrefclever_opt_varname_lang_type:eenn
814                 { \l__zrefclever_setup_language_tl }
815                 { \l__zrefclever_setup_type_tl }
816                 { gender }
817                 { seq }
818             }
819             {
820                 \seq_gset_eq:cN
821                 {
822                     \__zrefclever_opt_varname_lang_type:eenn

```

```

822             { \l__zrefclever_setup_language_tl }
823             { \l__zrefclever_setup_type_tl }
824             { gender }
825             { seq }
826         }
827         \l_tmpa_seq
828     }
829 }
830 }
831 }
832 } ,
833 }
834 \seq_map_inline:Nn
835   \c__zrefclever_rf_opts_tl_not_type_specific_seq
836   {
837     \keys_define:nn { zref-clever/langfile }
838     {
839       #1 .default:x = \c_novalue_tl ,
840       #1 .code:n =
841       {
842         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
843         {
844           \tl_if_novalue:nF {##1}
845           {
846             \__zrefclever_opt_tl_gset_if_new:cn
847             {
848               \__zrefclever_opt_varname_lang_default:enn
849               { \l__zrefclever_setup_language_tl }
850               {#1} { tl }
851             }
852             {##1}
853           }
854         }
855         {
856           \msg_info:nnn { zref-clever }
857           { option-not-type-specific } {#1}
858         }
859       } ,
860     }
861   }
862 \seq_map_inline:Nn
863   \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
864   {
865     \keys_define:nn { zref-clever/langfile }
866     {
867       #1 .default:x = \c_novalue_tl ,
868       #1 .code:n =
869       {
870         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
871         {
872           \tl_if_novalue:nF {##1}
873           {
874             \__zrefclever_opt_tl_gset_if_new:cn
875             {

```

```

876         \_zrefclever_opt_varname_lang_default:enn
877         { \l__zrefclever_setup_language_tl }
878         {#1} { tl }
879     }
880     {##1}
881 }
882 }
883 {
884     \tl_if_novalue:nF {##1}
885     {
886         \_zrefclever_opt_tl_gset_if_new:cn
887         {
888             \_zrefclever_opt_varname_lang_type:eenn
889             { \l__zrefclever_setup_language_tl }
890             { \l__zrefclever_setup_type_tl }
891             {#1} { tl }
892         }
893         {##1}
894     }
895 } ,
896 }
897 }
898 }
899 \seq_map_inline:Nn
900 \c__zrefclever_rf_opts_tl_type_names_seq
901 {
902     \keys_define:nn { zref-clever/langfile }
903     {
904         #1 .default:x = \c_novalue_tl ,
905         #1 .code:n =
906         {
907             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
908             {
909                 \msg_info:nnn { zref-clever }
910                 { option-only-type-specific } {#1}
911             }
912             {
913                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
914                 {
915                     \tl_if_novalue:nF {##1}
916                     {
917                         \_zrefclever_opt_tl_gset_if_new:cn
918                         {
919                             \_zrefclever_opt_varname_lang_type:eenn
920                             { \l__zrefclever_setup_language_tl }
921                             { \l__zrefclever_setup_type_tl }
922                             {#1} { tl }
923                         }
924                         {##1}
925                     }
926                 }
927             }
928             \tl_if_novalue:nF {##1}
929             {

```

```

930         \_zrefclever_opt_tl_gset_if_new:cn
931         {
932             \_zrefclever_opt_varname_lang_type:een
933             { \l__zrefclever_setup_language_tl }
934             { \l__zrefclever_setup_type_tl }
935             { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
936         }
937         {##1}
938     }
939 }
940 }
941 } ,
942 }
943 }
944 \seq_map_inline:Nn
945   \c__zrefclever_rf_opts_seq_refbounds_seq
946   {
947     \keys_define:nn { zref-clever/langfile }
948     {
949       #1 .default:x = \c_novalue_tl ,
950       #1 .code:n =
951       {
952         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
953         {
954           \tl_if_novalue:nF {##1}
955           {
956             \_zrefclever_opt_seq_if_set:cF
957             {
958               \_zrefclever_opt_varname_lang_default:enn
959               { \l__zrefclever_setup_language_tl } {##1} { seq }
960             }
961             {
962               \seq_gclear:N \g_tmpa_seq
963               \_zrefclever_opt_seq_gset_clist_split:Nn
964               \g_tmpa_seq {##1}
965               \bool_lazy_or:nnTF
966               { \tl_if_empty_p:n {##1} }
967               {
968                 \int_compare_p:nNn
969                 { \seq_count:N \g_tmpa_seq } = { 4 }
970               }
971               {
972                 \seq_gset_eq:cN
973                 {
974                   \_zrefclever_opt_varname_lang_default:enn
975                   { \l__zrefclever_setup_language_tl }
976                   {##1} { seq }
977                 }
978                 \g_tmpa_seq
979               }
980             }
981             \msg_info:nxxx { zref-clever }
982             { refbounds-must-be-four }
983             {##1} { \seq_count:N \g_tmpa_seq }

```

```

984         }
985     }
986 }
987 {
988     \tl_if_novalue:nF {##1}
989     {
990         \__zrefclever_opt_seq_if_set:cF
991         {
992             \__zrefclever_opt_varname_lang_type:eenn
993             { \l__zrefclever_setup_language_tl }
994             { \l__zrefclever_setup_type_tl } {#1} { seq }
995         }
996     }
997     {
998         \seq_gclear:N \g_tmpa_seq
999         \__zrefclever_opt_seq_gset_clist_split:Nn
1000         \g_tmpa_seq {##1}
1001         \bool_lazy_or:nnTF
1002         { \tl_if_empty_p:n {##1} }
1003         {
1004             \int_compare_p:nNn
1005             { \seq_count:N \g_tmpa_seq } = { 4 }
1006         }
1007         {
1008             \seq_gset_eq:cN
1009             {
1010                 \__zrefclever_opt_varname_lang_type:eenn
1011                 { \l__zrefclever_setup_language_tl }
1012                 { \l__zrefclever_setup_type_tl }
1013                 {#1} { seq }
1014             }
1015             \g_tmpa_seq
1016         }
1017     }
1018     \msg_info:nnxx { zref-clever }
1019     { refbounds-must-be-four }
1020     {#1} { \seq_count:N \g_tmpa_seq }
1021 }
1022 }
1023 }
1024 }
1025 } ,
1026 }
1027 }
1028 \seq_map_inline:Nn
1029 \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
1030 {
1031     \keys_define:nn { zref-clever/langfile }
1032     {
1033         #1 .choice: ,
1034         #1 / true .code:n =
1035         {
1036             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1037             {

```

```

1038         \__zrefclever_opt_bool_if_set:cF
1039         {
1040             \__zrefclever_opt_varname_lang_default:enn
1041             { \l__zrefclever_setup_language_tl }
1042             {#1} { bool }
1043         }
1044         {
1045             \bool_gset_true:c
1046             {
1047                 \__zrefclever_opt_varname_lang_default:enn
1048                 { \l__zrefclever_setup_language_tl }
1049                 {#1} { bool }
1050             }
1051         }
1052     }
1053     {
1054         \__zrefclever_opt_bool_if_set:cF
1055         {
1056             \__zrefclever_opt_varname_lang_type:enn
1057             { \l__zrefclever_setup_language_tl }
1058             { \l__zrefclever_setup_type_tl }
1059             {#1} { bool }
1060         }
1061         {
1062             \bool_gset_true:c
1063             {
1064                 \__zrefclever_opt_varname_lang_type:enn
1065                 { \l__zrefclever_setup_language_tl }
1066                 { \l__zrefclever_setup_type_tl }
1067                 {#1} { bool }
1068             }
1069         }
1070     }
1071 },
1072 #1 / false .code:n =
1073 {
1074     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1075     {
1076         \__zrefclever_opt_bool_if_set:cF
1077         {
1078             \__zrefclever_opt_varname_lang_default:enn
1079             { \l__zrefclever_setup_language_tl }
1080             {#1} { bool }
1081         }
1082         {
1083             \bool_gset_false:c
1084             {
1085                 \__zrefclever_opt_varname_lang_default:enn
1086                 { \l__zrefclever_setup_language_tl }
1087                 {#1} { bool }
1088             }
1089         }
1090     }
1091     {

```



```

1092 \__zrefclever_opt_bool_if_set:cF
1093 {
1094   \__zrefclever_opt_varname_lang_type:eenn
1095   { \l__zrefclever_setup_language_tl }
1096   { \l__zrefclever_setup_type_tl }
1097   {#1} { bool }
1098 }
1099 {
1100   \bool_gset_false:c
1101   {
1102     \__zrefclever_opt_varname_lang_type:eenn
1103     { \l__zrefclever_setup_language_tl }
1104     { \l__zrefclever_setup_type_tl }
1105     {#1} { bool }
1106   }
1107 }
1108 }
1109 } ,
1110 #1 / unset .code:n = { } ,
1111 #1 .default:n = true ,
1112 no #1 .meta:n = { #1 = false } ,
1113 no #1 .value_forbidden:n = true ,
1114 }
1115 }

```

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.

```

1116 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1117 {
1118   \tl_const:cn
1119   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1120 }
1121 \keyval_parse:nnn
1122 { }
1123 { \__zrefclever_opt_tl_cset_fallback:nn }
1124 {
1125   tpairsep = {,~} ,
1126   tlistsep = {,~} ,
1127   tlastsep = {,~} ,
1128   notesep = {~} ,
1129   namesep = {\nobreakspace} ,
1130   pairsep = {,~} ,
1131   listsep = {,~} ,
1132   lastsep = {,~} ,
1133   rangesep = {\textendash} ,
1134 }

```

4.8 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}

1135 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1136 {
1137   \tl_if_empty:nTF {#3}
1138     { \prop_remove:Nn #1 {#2} }
1139     { \prop_put:Nnn #1 {#2} {#3} }
1140 }
```

(End definition for `_zrefclever_prop_put_non_empty:Nnn`.)

ref option

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

```

1141 \tl_new:N \l__zrefclever_ref_property_tl
1142 \keys_define:nn { zref-clever/reference }
1143 {
1144   ref .code:n =
1145   {
1146     \zref@ifpropundefined {#1}
1147     {
1148       \msg_warning:nnn { zref-clever } { zref-property-undefined } {#1}
1149       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1150     }
1151     { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1152   } ,
1153   ref .initial:n = default ,
1154   ref .value_required:n = true ,
1155   page .meta:n = { ref = page },
1156   page .value_forbidden:n = true ,
1157 }
```

typeset option

```

1158 \bool_new:N \l__zrefclever_typeset_ref_bool
1159 \bool_new:N \l__zrefclever_typeset_name_bool
1160 \keys_define:nn { zref-clever/reference }
1161 {
1162   typeset .choice: ,
1163   typeset / both .code:n =
```

```

1164     {
1165         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1166         \bool_set_true:N \l__zrefclever_typeset_name_bool
1167     } ,
1168     typeset / ref .code:n =
1169     {
1170         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1171         \bool_set_false:N \l__zrefclever_typeset_name_bool
1172     } ,
1173     typeset / name .code:n =
1174     {
1175         \bool_set_false:N \l__zrefclever_typeset_ref_bool
1176         \bool_set_true:N \l__zrefclever_typeset_name_bool
1177     } ,
1178     typeset .initial:n = both ,
1179     typeset .value_required:n = true ,
1180
1181     noname .meta:n = { typeset = ref } ,
1182     noname .value_forbidden:n = true ,
1183     noref .meta:n = { typeset = name } ,
1184     noref .value_forbidden:n = true ,
1185 }

```

sort option

```

1186 \bool_new:N \l__zrefclever_typeset_sort_bool
1187 \keys_define:nn { zref-clever/reference }
1188 {
1189     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1190     sort .initial:n = true ,
1191     sort .default:n = true ,
1192     nosort .meta:n = { sort = false } ,
1193     nosort .value_forbidden:n = true ,
1194 }

```

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.

```

1195 \seq_new:N \l__zrefclever_typesort_seq
1196 \keys_define:nn { zref-clever/reference }
1197 {
1198     typesort .code:n =
1199     {
1200         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1201         \seq_reverse:N \l__zrefclever_typesort_seq
1202     } ,
1203     typesort .initial:n =
1204     { part , chapter , section , paragraph } ,
1205     typesort .value_required:n = true ,
1206     notypesort .code:n =
1207     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1208     notypesort .value_forbidden:n = true ,

```

```
1209 }
```

comp option

```
1210 \bool_new:N \l__zrefclever_typeset_compress_bool
1211 \keys_define:nn { zref-clever/reference }
1212 {
1213   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1214   comp .initial:n = true ,
1215   comp .default:n = true ,
1216   nocomp .meta:n = { comp = false },
1217   nocomp .value_forbidden:n = true ,
1218 }
```

range option

```
1219 \bool_new:N \l__zrefclever_typeset_range_bool
1220 \keys_define:nn { zref-clever/reference }
1221 {
1222   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1223   range .initial:n = false ,
1224   range .default:n = true ,
1225 }
```

cap and capfirst options

The cap option is currently being handled with other reference format option booleans at `\c__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```
1226 \bool_new:N \l__zrefclever_capfirst_bool
1227 \keys_define:nn { zref-clever/reference }
1228 {
1229   capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1230   capfirst .initial:n = false ,
1231   capfirst .default:n = true ,
1232 }
```

abbrev and noabbrevfirst options

The abbrev option is currently being handled with other reference format option booleans at `\c__zrefclever_rf_opts_bool_maybe_type_specific_seq`.

```
1233 \bool_new:N \l__zrefclever_noabbrev_first_bool
1234 \keys_define:nn { zref-clever/reference }
1235 {
1236   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1237   noabbrevfirst .initial:n = false ,
1238   noabbrevfirst .default:n = true ,
1239 }
```

S option

```
1240 \keys_define:nn { zref-clever/reference }
1241 {
1242   S .meta:n =
1243     { capfirst = {#1} , noabbrevfirst = {#1} },
1244   S .default:n = true ,
```

```
1245 }
```

hyperref option

```
1246 \bool_new:N \l__zrefclever_hyperlink_bool
1247 \bool_new:N \l__zrefclever_hyperref_warn_bool
1248 \keys_define:nn { zref-clever/reference }
1249 {
1250   hyperref .choice: ,
1251   hyperref / auto .code:n =
1252   {
1253     \bool_set_true:N \l__zrefclever_hyperlink_bool
1254     \bool_set_false:N \l__zrefclever_hyperref_warn_bool
1255   } ,
1256   hyperref / true .code:n =
1257   {
1258     \bool_set_true:N \l__zrefclever_hyperlink_bool
1259     \bool_set_true:N \l__zrefclever_hyperref_warn_bool
1260   } ,
1261   hyperref / false .code:n =
1262   {
1263     \bool_set_false:N \l__zrefclever_hyperlink_bool
1264     \bool_set_false:N \l__zrefclever_hyperref_warn_bool
1265   } ,
1266   hyperref .initial:n = auto ,
1267   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.

```
1268   nohyperref .meta:n = { hyperref = false } ,
1269   nohyperref .value_forbidden:n = true ,
1270 }
1271 \AddToHook { begindocument }
1272 {
1273   \__zrefclever_if_package_loaded:nTF { hyperref }
1274   {
1275     \bool_if:NT \l__zrefclever_hyperlink_bool
1276     { \RequirePackage { zref-hyperref } }
1277   }
1278   {
1279     \bool_if:NT \l__zrefclever_hyperref_warn_bool
1280     { \msg_warning:nn { zref-clever } { missing-hyperref } }
1281     \bool_set_false:N \l__zrefclever_hyperlink_bool
1282   }
1283   \keys_define:nn { zref-clever/reference }
1284   {
1285     hyperref .code:n =
1286     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
1287     nohyperref .code:n =
1288     { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
1289   }
1290 }
```

nameinlink option

```
1291 \str_new:N \l__zrefclever_nameinlink_str
1292 \keys_define:nn { zref-clever/reference }
1293 {
1294     nameinlink .choice: ,
1295     nameinlink / true .code:n =
1296         { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
1297     nameinlink / false .code:n =
1298         { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
1299     nameinlink / single .code:n =
1300         { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
1301     nameinlink / tsingle .code:n =
1302         { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
1303     nameinlink .initial:n = tsingle ,
1304     nameinlink .default:n = true ,
1305 }
```

preposinlink option (deprecated)

```
1306 \keys_define:nn { zref-clever/reference }
1307 {
1308     preposinlink .code:n =
1309     {
1310         % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
1311         \msg_warning:nnnn { zref-clever } { option-deprecated }
1312         { preposinlink } { refbounds }
1313     } ,
1314 }
```

lang option

`\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.

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`.

```

1315 \tl_new:N \l__zrefclever_ref_language_tl
1316 \tl_new:N \l__zrefclever_current_language_tl
1317 \tl_new:N \l__zrefclever_main_language_tl
1318 \AddToHook { begindocument }
1319 {
1320   \__zrefclever_if_package_loaded:nTF { babel }
1321   {
1322     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
1323     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
1324   }
1325   {
1326     \__zrefclever_if_package_loaded:nTF { polyglossia }
1327     {
1328       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
1329       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
1330     }
1331     {
1332       \tl_set:Nn \l__zrefclever_current_language_tl { english }
1333       \tl_set:Nn \l__zrefclever_main_language_tl { english }
1334     }
1335   }
1336 }

```

`\l_zrefclever_ref_language_tl` A public version of `\l__zrefclever_ref_language_tl` for use in `zref-vario`.

```

1337 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```

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

```

1338 \keys_define:nn { zref-clever/reference }
1339 {
1340   lang .code:n =
1341   {
1342     \AddToHook { begindocument }
1343     {
1344       \str_case:nnF {#1}
1345       {
1346         { current }
1347         {
1348           \tl_set:Nn \l__zrefclever_ref_language_tl
1349           { \l__zrefclever_current_language_tl }
1350         }
1351
1352         { main }
1353         {
1354           \tl_set:Nn \l__zrefclever_ref_language_tl
1355           { \l__zrefclever_main_language_tl }
1356         }
1357       }
1358     }
1359     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}

```

```

1360         \__zrefclever_language_if_declared:nF {#1}
1361         {
1362             \msg_warning:nnn { zref-clever }
1363             { unknown-language-opt } {#1}
1364         }
1365     }
1366     \__zrefclever_provide_langfile:x
1367     { \l__zrefclever_ref_language_tl }
1368 }
1369 },
1370 lang .initial:n = current ,
1371 lang .value_required:n = true ,
1372 }
1373 \AddToHook { begindocument / before }
1374 {
1375     \AddToHook { begindocument }
1376     {

```

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.

```

1377     \keys_define:nn { zref-clever/reference }
1378     {
1379         lang .code:n =
1380         {
1381             \str_case:nnF {#1}
1382             {
1383                 { current }
1384                 {
1385                     \tl_set:Nn \l__zrefclever_ref_language_tl
1386                     { \l__zrefclever_current_language_tl }
1387                 }
1388             }
1389             { main }
1390             {
1391                 \tl_set:Nn \l__zrefclever_ref_language_tl
1392                 { \l__zrefclever_main_language_tl }
1393             }
1394         }
1395         {
1396             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1397             \__zrefclever_language_if_declared:nF {#1}
1398             {
1399                 \msg_warning:nnn { zref-clever }
1400                 { unknown-language-opt } {#1}
1401             }
1402         }
1403     },
1404 }
1405 }
1406 }

```


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.

```
1407 \tl_new:N \l__zrefclever_ref_decl_case_tl
1408 \keys_define:nn { zref-clever/reference }
1409 {
1410   d .code:n =
1411     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1412 }
1413 \AddToHook { begindocument }
1414 {
1415   \keys_define:nn { zref-clever/reference }
1416   {
```

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

```
1417     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1418     d .value_required:n = true ,
1419   }
1420 }
```

nudge & co. options

```
1421 \bool_new:N \l__zrefclever_nudge_enabled_bool
1422 \bool_new:N \l__zrefclever_nudge_multitype_bool
1423 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1424 \bool_new:N \l__zrefclever_nudge_singular_bool
1425 \bool_new:N \l__zrefclever_nudge_gender_bool
1426 \tl_new:N \l__zrefclever_ref_gender_tl
1427 \keys_define:nn { zref-clever/reference }
1428 {
1429   nudge .choice: ,
1430   nudge / true .code:n =
1431     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1432   nudge / false .code:n =
1433     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1434   nudge / ifdraft .code:n =
1435     {
1436       \ifdraft
1437         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1438         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1439     } ,
1440   nudge / iffinal .code:n =
1441     {
1442       \ifoptionfinal
1443         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1444         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1445     } ,
1446   nudge .initial:n = false ,
```

```

1447 nudge .default:n = true ,
1448 nonudge .meta:n = { nudge = false } ,
1449 nonudge .value_forbidden:n = true ,
1450 nudgeif .code:n =
1451 {
1452   \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1453   \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1454   \bool_set_false:N \l__zrefclever_nudge_gender_bool
1455   \clist_map_inline:nn {#1}
1456   {
1457     \str_case:nnF {##1}
1458     {
1459       { multitype }
1460       { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1461       { comptosing }
1462       { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1463       { gender }
1464       { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1465       { all }
1466       {
1467         \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1468         \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1469         \bool_set_true:N \l__zrefclever_nudge_gender_bool
1470       }
1471     }
1472     {
1473       \msg_warning:nnn { zref-clever }
1474       { nudgeif-unknown-value } {##1}
1475     }
1476   }
1477 },
1478 nudgeif .value_required:n = true ,
1479 nudgeif .initial:n = all ,
1480 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1481 sg .initial:n = false ,
1482 sg .default:n = true ,
1483 g .code:n =
1484 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1485 }
1486 \AddToHook { begindocument }
1487 {
1488   \keys_define:nn { zref-clever/reference }
1489   {

```

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

```

1490   g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1491   g .value_required:n = true ,
1492 }
1493 }
```

font option

`font` can't be used as a package option, since the options get expanded by L^AT_EX before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can

be set in `\zcref` and, for global settings, with `\zcsetup`. Note that, technically, the “raw” options are already available as `\@raw@opt@⟨package⟩.sty` (helpful comment by David Carlisle at <https://tex.stackexchange.com/a/618439>).

```

1494 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1495 \keys_define:nn { zref-clever/reference }
1496 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

1497 \keys_define:nn { zref-clever/reference }
1498 {
1499     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1500     titleref .value_forbidden:n = true ,
1501 }
1502 \AddToHook { begindocument }
1503 {
1504     \keys_define:nn { zref-clever/reference }
1505     {
1506         titleref .code:n =
1507             { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1508     }
1509 }

```

note option

```

1510 \tl_new:N \l__zrefclever_zcref_note_tl
1511 \keys_define:nn { zref-clever/reference }
1512 {
1513     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
1514     note .value_required:n = true ,
1515 }

```

check option

Integration with `zref-check`.

```

1516 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1517 \bool_new:N \l__zrefclever_zcref_with_check_bool
1518 \keys_define:nn { zref-clever/reference }
1519 {
1520     check .code:n = { \RequirePackage { zref-check } } ,
1521     check .value_forbidden:n = true ,
1522 }
1523 \AddToHook { begindocument }
1524 {
1525     \__zrefclever_if_package_loaded:nTF { zref-check }
1526     {
1527         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1528         \keys_define:nn { zref-clever/reference }
1529         {
1530             check .code:n =
1531             {
1532                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
1533                 \keys_set:nn { zref-check / zcheck } {#1}
1534             } ,
1535             check .value_required:n = true ,

```

```

1536     }
1537   }
1538   {
1539     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1540     \keys_define:nn { zref-clever/reference }
1541     {
1542       check .value_forbidden:n = false ,
1543       check .code:n =
1544         { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1545     }
1546   }
1547 }

```

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`.

```

1548 \prop_new:N \l__zrefclever_counter_type_prop
1549 \keys_define:nn { zref-clever/label }
1550 {
1551   countertype .code:n =
1552   {
1553     \keyval_parse:nnn
1554     {
1555       \msg_warning:nnnn { zref-clever }
1556       { key-requires-value } { countertype }
1557     }
1558     {
1559       \__zrefclever_prop_put_non_empty:Nnn
1560       \l__zrefclever_counter_type_prop
1561     }
1562     {#1}
1563   } ,
1564   countertype .value_required:n = true ,
1565   countertype .initial:n =
1566   {
1567     subsection      = section ,
1568     subsubsection   = section ,
1569     subparagraph    = paragraph ,
1570     enumi           = item ,
1571     enumii          = item ,
1572     enumiii         = item ,
1573     enumiv          = item ,
1574     mpfootnote      = footnote ,
1575   } ,
1576 }

```

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_resetby: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.

```

1577 \seq_new:N \l__zrefclever_counter_resetters_seq
1578 \keys_define:nn { zref-clever/label }
1579 {
1580   counterresetters .code:n =
1581   {
1582     \clist_map_inline:nn {#1}
1583     {
1584       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1585       {
1586         \seq_put_right:Nn
1587           \l__zrefclever_counter_resetters_seq {##1}
1588       }
1589     }
1590   } ,
1591   counterresetters .initial:n =
1592   {
1593     part ,
1594     chapter ,
1595     section ,
1596     subsection ,
1597     subsubsection ,
1598     paragraph ,
1599     subparagraph ,
1600   } ,
1601   counterresetters .value_required:n = true ,
1602 }
```

counterresetby option

`\l__zrefclever_counter_resetby_prop` is used by `__zrefclever_counter_resetby:n` to populate the `zc@enclval` property, and stores a mapping from counters to the

counter which resets each of them. This mapping has precedence in `__zrefclever_counter_reset_by:n` over the search through `\l__zrefclever_counter_resettters_seq`.

```

1603 \prop_new:N \l__zrefclever_counter_resetby_prop
1604 \keys_define:nn { zref-clever/label }
1605 {
1606   counterresetby .code:n =
1607   {
1608     \keyval_parse:nnn
1609     {
1610       \msg_warning:nnn { zref-clever }
1611       { key-requires-value } { counterresetby }
1612     }
1613     {
1614       \__zrefclever_prop_put_non_empty:Nnn
1615       \l__zrefclever_counter_resetby_prop
1616     }
1617     {#1}
1618   } ,
1619   counterresetby .value_required:n = true ,
1620   counterresetby .initial:n =
1621   {

```

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.

```

1622   enumii = enumi ,
1623   enumiii = enumii ,
1624   enumiv = enumiii ,
1625 } ,
1626 }

```

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.

```

1627 \tl_new:N \l__zrefclever_current_counter_tl
1628 \keys_define:nn { zref-clever/label }
1629 {
1630   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1631   currentcounter .value_required:n = true ,
1632   currentcounter .initial:n = \@currentcounter ,
1633 }

```

nocompat option

```

1634 \bool_new:N \g__zrefclever_nocompat_bool
1635 \seq_new:N \g__zrefclever_nocompat_modules_seq
1636 \keys_define:nn { zref-clever/reference }
1637 {
1638   nocompat .code:n =

```

```

1639 {
1640   \tl_if_empty:nTF {#1}
1641   { \bool_gset_true:N \g__zrefclever_nocompat_bool }
1642   {
1643     \clist_map_inline:nn {#1}
1644     {
1645       \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
1646       {
1647         \seq_gput_right:Nn
1648         \g__zrefclever_nocompat_modules_seq {##1}
1649       }
1650     }
1651   }
1652 },
1653 }
1654 \AddToHook { begindocument }
1655 {
1656   \keys_define:nn { zref-clever/reference }
1657   {
1658     nocompat .code:n =
1659     {
1660       \msg_warning:nnn { zref-clever }
1661       { option-preamble-only } { nocompat }
1662     }
1663   }
1664 }
1665 \AtEndOfPackage
1666 {
1667   \AddToHook { begindocument }
1668   {
1669     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
1670     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
1671   }
1672 }

```

`_zrefclever_compat_module:nn`

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

```

\__zrefclever_compat_module:nn {\langle module \rangle} {\langle code \rangle}

1673 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
1674 {
1675   \AddToHook { begindocument }
1676   {
1677     \bool_if:NF \g__zrefclever_nocompat_bool
1678     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }

```

```

1679     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
1680   }
1681 }

```

(End definition for `__zrefclever_compat_module:nn`.)

Reference options

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

```

1682 \seq_map_inline:Nn
1683   \c__zrefclever_rf_opts_tl_reference_seq
1684   {
1685     \keys_define:nn { zref-clever/reference }
1686     {
1687       #1 .default:x = \c_novalue_tl ,
1688       #1 .code:n =
1689       {
1690         \tl_if_novalue:nTF {##1}
1691         {
1692           \__zrefclever_opt_tl_unset:c
1693           { \__zrefclever_opt_varname_general:nn {#1} { tl } }
1694         }
1695         {
1696           \tl_set:cn
1697           { \__zrefclever_opt_varname_general:nn {#1} { tl } }
1698           {##1}
1699         }
1700       } ,
1701     }
1702   }
1703 \keys_define:nn { zref-clever/reference }
1704   {
1705     refpre .code:n =
1706     {
1707       % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1708       \msg_warning:nnnn { zref-clever }{ option-deprecated }
1709       { refpre } { refbounds }
1710     } ,
1711     refpos .code:n =
1712     {
1713       % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1714       \msg_warning:nnnn { zref-clever }{ option-deprecated }
1715       { refpos } { refbounds }
1716     } ,
1717     preref .code:n =
1718     {
1719       % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
1720       \msg_warning:nnnn { zref-clever }{ option-deprecated }
1721       { preref } { refbounds }
1722     } ,
1723     postref .code:n =

```



```

1724     {
1725         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
1726         \msg_warning:nnnn { zref-clever }{ option-deprecated }
1727         { postref } { refbounds }
1728     } ,
1729 }
1730 \seq_map_inline:Nn
1731   \c__zrefclever_rf_opts_seq_refbounds_seq
1732   {
1733     \keys_define:nn { zref-clever/reference }
1734     {
1735       #1 .default:x = \c_novalue_tl ,
1736       #1 .code:n =
1737       {
1738         \tl_if_novalue:nTF {##1}
1739         {
1740           \__zrefclever_opt_seq_unset:c
1741           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
1742         }
1743         {
1744           \seq_clear:N \l_tmpa_seq
1745           \__zrefclever_opt_seq_set_clist_split:Nn
1746             \l_tmpa_seq {##1}
1747           \bool_lazy_or:nnTF
1748             { \tl_if_empty_p:n {##1} }
1749             { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
1750             {
1751               \seq_set_eq:cN
1752                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
1753                 \l_tmpa_seq
1754             }
1755             {
1756               \msg_warning:nxxx { zref-clever }
1757               { refbounds-must-be-four }
1758               {#1} { \seq_count:N \l_tmpa_seq }
1759             }
1760           }
1761         } ,
1762       }
1763     }
1764 \seq_map_inline:Nn
1765   \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
1766   {
1767     \keys_define:nn { zref-clever/reference }
1768     {
1769       #1 .choice: ,
1770       #1 / true .code:n =
1771       {
1772         \bool_set_true:c
1773         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
1774       } ,
1775       #1 / false .code:n =
1776       {
1777         \bool_set_false:c

```

```

1778         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
1779     } ,
1780     #1 / unset .code:n =
1781     {
1782         \_zrefclever_opt_bool_unset:c
1783         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
1784     } ,
1785     #1 .default:n = true ,
1786     no #1 .meta:n = { #1 = false } ,
1787     no #1 .value_forbidden:n = true ,
1788 }
1789 }

```

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 load-time package options and for `\zcsetup` we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```

1790 \keys_define:nn { }
1791 {
1792     zref-clever/zcsetup .inherit:n =
1793     {
1794         zref-clever/label ,
1795         zref-clever/reference ,
1796     }
1797 }

```

Process load-time package options (<https://tex.stackexchange.com/a/15840>).

```

1798 \ProcessKeysOptions { zref-clever/zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

```

```

1799 \NewDocumentCommand \zcsetup { m }
1800 { \_zrefclever_zcsetup:n {#1} }

```

(End definition for \zcsetup.)

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

```

\_zrefclever_zcsetup:n{<options>}

```

```

1801 \cs_new_protected:Npn \_zrefclever_zcsetup:n #1
1802 { \keys_set:nn { zref-clever/zcsetup } {#1} }
1803 \cs_generate_variant:Nn \_zrefclever_zcsetup:n { x }

```

(End definition for _zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

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

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

1804 \NewDocumentCommand \zcRefTypeSetup { m m }
1805 {
1806   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1807   \keys_set:nn { zref-clever/typesetup } {#2}
1808 }

(End definition for \zcRefTypeSetup.)

1809 \seq_map_inline:Nn
1810 \c__zrefclever_rf_opts_tl_not_type_specific_seq
1811 {
1812   \keys_define:nn { zref-clever/typesetup }
1813   {
1814     #1 .code:n =
1815     {
1816       \msg_warning:nnn { zref-clever }
1817       { option-not-type-specific } {#1}
1818     } ,
1819   }
1820 }
1821 \seq_map_inline:Nn
1822 \c__zrefclever_rf_opts_tl_typesetup_seq
1823 {
1824   \keys_define:nn { zref-clever/typesetup }
1825   {
1826     #1 .default:x = \c_novalue_tl ,
1827     #1 .code:n =
1828     {
1829       \tl_if_novalue:nTF {##1}
1830       {
1831         \__zrefclever_opt_tl_unset:c
1832         {
1833           \__zrefclever_opt_varname_type:enn
1834           { \l__zrefclever_setup_type_tl } {#1} { tl }
1835         }
1836       }
1837       {
1838         \tl_set:cn
1839         {
1840           \__zrefclever_opt_varname_type:enn
1841           { \l__zrefclever_setup_type_tl } {#1} { tl }
1842         }
1843         {##1}
1844       }
1845     }
1846   }
1847 }
```

```

1845     } ,
1846   }
1847 }
1848 \keys_define:nn { zref-clever/typesetup }
1849 {
1850   refpre .code:n =
1851   {
1852     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1853     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1854     { refpre } { refbounds }
1855   } ,
1856   refpos .code:n =
1857   {
1858     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1859     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1860     { refpos } { refbounds }
1861   } ,
1862   preref .code:n =
1863   {
1864     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
1865     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1866     { preref } { refbounds }
1867   } ,
1868   postref .code:n =
1869   {
1870     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
1871     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1872     { postref } { refbounds }
1873   } ,
1874 }
1875 \seq_map_inline:Nn
1876 \c__zrefclever_rf_opts_seq_refbounds_seq
1877 {
1878   \keys_define:nn { zref-clever/typesetup }
1879   {
1880     #1 .default:x = \c_novalue_tl ,
1881     #1 .code:n =
1882     {
1883       \tl_if_novalue:nTF {##1}
1884       {
1885         \__zrefclever_opt_seq_unset:c
1886         {
1887           \__zrefclever_opt_varname_type:enn
1888           { \l__zrefclever_setup_type_tl } {#1} { seq }
1889         }
1890       }
1891       {
1892         \seq_clear:N \l_tmpa_seq
1893         \__zrefclever_opt_seq_set_clist_split:Nn
1894         \l_tmpa_seq {##1}
1895         \bool_lazy_or:nnTF
1896         { \tl_if_empty_p:n {##1} }
1897         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
1898         {

```

```

1899         \seq_set_eq:cN
1900         {
1901             \__zrefclever_opt_varname_type:enn
1902             { \l__zrefclever_setup_type_tl } {#1} { seq }
1903         }
1904         \l_tmpa_seq
1905     }
1906     {
1907         \msg_warning:nxxx { zref-clever }
1908         { refbounds-must-be-four }
1909         {#1} { \seq_count:N \l_tmpa_seq }
1910     }
1911 } ,
1912 } ,
1913 }
1914 }
1915 \seq_map_inline:Nn
1916 \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
1917 {
1918     \keys_define:nn { zref-clever/typesetup }
1919     {
1920         #1 .choice: ,
1921         #1 / true .code:n =
1922         {
1923             \bool_set_true:c
1924             {
1925                 \__zrefclever_opt_varname_type:enn
1926                 { \l__zrefclever_setup_type_tl }
1927                 {#1} { bool }
1928             }
1929         } ,
1930         #1 / false .code:n =
1931         {
1932             \bool_set_false:c
1933             {
1934                 \__zrefclever_opt_varname_type:enn
1935                 { \l__zrefclever_setup_type_tl }
1936                 {#1} { bool }
1937             }
1938         } ,
1939         #1 / unset .code:n =
1940         {
1941             \__zrefclever_opt_bool_unset:c
1942             {
1943                 \__zrefclever_opt_varname_type:enn
1944                 { \l__zrefclever_setup_type_tl }
1945                 {#1} { bool }
1946             }
1947         } ,
1948         #1 .default:n = true ,
1949         no #1 .meta:n = { #1 = false } ,
1950         no #1 .value_forbidden:n = true ,
1951     }
1952 }

```

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>}

1953 \NewDocumentCommand \zcLanguageSetup { m m }
1954 {
1955   \group_begin:
1956   \__zrefclever_language_if_declared:nTF {#1}
1957   {
1958     \tl_clear:N \l__zrefclever_setup_type_tl
1959     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
1960     \__zrefclever_opt_seq_get:cNF
1961     {
1962       \__zrefclever_opt_varname_language:nnn
1963       {#1} { declension } { seq }
1964     }
1965     \l__zrefclever_lang_declension_seq
1966     { \seq_clear:N \l__zrefclever_lang_declension_seq }
1967     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1968     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
1969     {
1970       \seq_get_left:NN \l__zrefclever_lang_declension_seq
1971       \l__zrefclever_lang_decl_case_tl
1972     }
1973     \__zrefclever_opt_seq_get:cNF
1974     {
1975       \__zrefclever_opt_varname_language:nnn
1976       {#1} { gender } { seq }
1977     }
1978     \l__zrefclever_lang_gender_seq
1979     { \seq_clear:N \l__zrefclever_lang_gender_seq }
1980     \keys_set:nn { zref-clever/langsetup } {#2}
1981   }
1982   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1983   \group_end:
1984 }
1985 \@onlypreamble \zcLanguageSetup
```

(End definition for \zcLanguageSetup.)

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

```
1986 \keys_define:nn { zref-clever/langsetup }
1987 {
1988   type .code:n =
1989   {
1990     \tl_if_empty:nTF {#1}
```

```

1991         { \tl_clear:N \l__zrefclever_setup_type_tl }
1992         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1993     } ,
1994
1995     case .code:n =
1996     {
1997         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1998         {
1999             \msg_warning:nnxx { zref-clever } { language-no-decl-setup }
2000             { \l__zrefclever_setup_language_tl } {#1}
2001         }
2002         {
2003             \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2004             { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2005             {
2006                 \msg_warning:nnxx { zref-clever } { unknown-decl-case }
2007                 {#1} { \l__zrefclever_setup_language_tl }
2008                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
2009                 \l__zrefclever_lang_decl_case_tl
2010             }
2011         }
2012     } ,
2013     case .value_required:n = true ,
2014
2015     gender .default:x = \c_novalue_tl ,
2016     gender .code:n =
2017     {
2018         \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2019         {
2020             \msg_warning:nnxxx { zref-clever } { language-no-gender }
2021             { \l__zrefclever_setup_language_tl } { gender } {#1}
2022         }
2023         {
2024             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2025             {
2026                 \msg_warning:nnn { zref-clever }
2027                 { option-only-type-specific } { gender }
2028             }
2029             {
2030                 \tl_if_novalue:nTF {#1}
2031                 {
2032                     \__zrefclever_opt_seq_gunset:c
2033                     {
2034                         \__zrefclever_opt_varname_lang_type:eenn
2035                         { \l__zrefclever_setup_language_tl }
2036                         { \l__zrefclever_setup_type_tl }
2037                         { gender }
2038                         { seq }
2039                     }
2040                 }
2041                 {
2042                     \seq_clear:N \l_tmpa_seq
2043                     \clist_map_inline:nn {#1}
2044                     {

```

```

2045         \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2046         { \seq_put_right:Nn \l_tmpa_seq {##1} }
2047         {
2048             \msg_warning:nxxx { zref-clever }
2049             { gender-not-declared }
2050             { \l__zrefclever_setup_language_tl } {##1}
2051         }
2052     }
2053     \seq_gset_eq:cN
2054     {
2055         \__zrefclever_opt_varname_lang_type:enn
2056         { \l__zrefclever_setup_language_tl }
2057         { \l__zrefclever_setup_type_tl }
2058         { gender }
2059         { seq }
2060     }
2061     \l_tmpa_seq
2062 }
2063 }
2064 }
2065 },
2066 }
2067 \seq_map_inline:Nn
2068 \c__zrefclever_rf_opts_tl_not_type_specific_seq
2069 {
2070     \keys_define:nn { zref-clever/langsetup }
2071     {
2072         #1 .default:x = \c_novalue_tl ,
2073         #1 .code:n =
2074         {
2075             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2076             {
2077                 \tl_if_novalue:NTF {##1}
2078                 {
2079                     \__zrefclever_opt_tl_gunset:c
2080                     {
2081                         \__zrefclever_opt_varname_lang_default:enn
2082                         { \l__zrefclever_setup_language_tl } {##1} { tl }
2083                     }
2084                 }
2085                 {
2086                     \tl_gset:cn
2087                     {
2088                         \__zrefclever_opt_varname_lang_default:enn
2089                         { \l__zrefclever_setup_language_tl } {##1} { tl }
2090                     }
2091                     {##1}
2092                 }
2093             }
2094             {
2095                 \msg_warning:nnn { zref-clever }
2096                 { option-not-type-specific } {##1}
2097             }
2098         },

```



```

2099     }
2100   }
2101   \seq_map_inline:Nn
2102     \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
2103   {
2104     \keys_define:nn { zref-clever/langsetup }
2105     {
2106       #1 .value_required:n = true ,
2107       #1 .code:n =
2108       {
2109         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2110         {
2111           \tl_if_novalue:NTF {##1}
2112           {
2113             \__zrefclever_opt_tl_gunset:c
2114             {
2115               \__zrefclever_opt_varname_lang_default:enn
2116               { \l__zrefclever_setup_language_tl } {#1} { t1 }
2117             }
2118           }
2119           {
2120             \tl_gset:cn
2121             {
2122               \__zrefclever_opt_varname_lang_default:enn
2123               { \l__zrefclever_setup_language_tl } {#1} { t1 }
2124             }
2125             {##1}
2126           }
2127         }
2128         {
2129           \tl_if_novalue:NTF {##1}
2130           {
2131             \__zrefclever_opt_tl_gunset:c
2132             {
2133               \__zrefclever_opt_varname_lang_type:eenn
2134               { \l__zrefclever_setup_language_tl }
2135               { \l__zrefclever_setup_type_tl }
2136               {#1} { t1 }
2137             }
2138           }
2139           {
2140             \tl_gset:cn
2141             {
2142               \__zrefclever_opt_varname_lang_type:eenn
2143               { \l__zrefclever_setup_language_tl }
2144               { \l__zrefclever_setup_type_tl }
2145               {#1} { t1 }
2146             }
2147             {##1}
2148           }
2149         }
2150       } ,
2151     }
2152   }

```

```

2153 \keys_define:nn { zref-clever/langsetup }
2154 {
2155   refpre .code:n =
2156   {
2157     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2158     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2159     { refpre } { refbounds }
2160   } ,
2161   refpos .code:n =
2162   {
2163     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2164     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2165     { refpos } { refbounds }
2166   } ,
2167   preref .code:n =
2168   {
2169     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2170     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2171     { preref } { refbounds }
2172   } ,
2173   postref .code:n =
2174   {
2175     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2176     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2177     { postref } { refbounds }
2178   } ,
2179 }
2180 \seq_map_inline:Nn
2181 \c__zrefclever_rf_opts_tl_type_names_seq
2182 {
2183   \keys_define:nn { zref-clever/langsetup }
2184   {
2185     #1 .value_required:n = true ,
2186     #1 .code:n =
2187     {
2188       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2189       {
2190         \msg_warning:nnn { zref-clever }
2191         { option-only-type-specific } {#1}
2192       }
2193       {
2194         \tl_if_novalue:NTF {##1}
2195         {
2196           \__zrefclever_opt_tl_gunset:c
2197           {
2198             \__zrefclever_opt_varname_lang_type:eenn
2199             { \l__zrefclever_setup_language_tl }
2200             { \l__zrefclever_setup_type_tl }
2201             {#1} { tl }
2202           }
2203         }
2204         {
2205           \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
2206           {

```

```

2207         \tl_gset:cn
2208         {
2209             \__zrefclever_opt_varname_lang_type:een
2210             { \l__zrefclever_setup_language_tl }
2211             { \l__zrefclever_setup_type_tl }
2212             {#1} { tl }
2213         }
2214         {##1}
2215     }
2216     {
2217         \tl_gset:cn
2218         {
2219             \__zrefclever_opt_varname_lang_type:een
2220             { \l__zrefclever_setup_language_tl }
2221             { \l__zrefclever_setup_type_tl }
2222             { \l__zrefclever_lang_decl_case_tl - #1 }
2223             { tl }
2224         }
2225         {##1}
2226     }
2227 }
2228 } ,
2229 }
2230 }
2231 }
2232 \seq_map_inline:Nn
2233 \c__zrefclever_rf_opts_seq_refbounds_seq
2234 {
2235     \keys_define:nn { zref-clever/langsetup }
2236     {
2237         #1 .default:x = \c_novalue_tl ,
2238         #1 .code:n =
2239         {
2240             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2241             {
2242                 \tl_if_novalue:NTF {##1}
2243                 {
2244                     \__zrefclever_opt_seq_gunset:c
2245                     {
2246                         \__zrefclever_opt_varname_lang_default:enn
2247                         { \l__zrefclever_setup_language_tl }
2248                         {#1} { seq }
2249                     }
2250                 }
2251             }
2252             \seq_gclear:N \g_tmpa_seq
2253             \__zrefclever_opt_seq_gset_clist_split:Nn
2254             \g_tmpa_seq {##1}
2255             \bool_lazy_or:nnTF
2256             { \tl_if_empty_p:n {##1} }
2257             {
2258                 \int_compare_p:nNn
2259                 { \seq_count:N \g_tmpa_seq } = { 4 }
2260             }

```

```

2261         {
2262             \seq_gset_eq:cN
2263             {
2264                 \__zrefclever_opt_varname_lang_default:enn
2265                 { \l__zrefclever_setup_language_tl }
2266                 {#1} { seq }
2267             }
2268             \g_tmpa_seq
2269         }
2270         {
2271             \msg_warning:nxxx { zref-clever }
2272             { refbounds-must-be-four }
2273             {#1} { \seq_count:N \g_tmpa_seq }
2274         }
2275     }
2276 }
2277 {
2278     \tl_if_novalue:nTF {##1}
2279     {
2280         \__zrefclever_opt_seq_gunset:c
2281         {
2282             \__zrefclever_opt_varname_lang_type:eenn
2283             { \l__zrefclever_setup_language_tl }
2284             { \l__zrefclever_setup_type_tl } {#1} { seq }
2285         }
2286     }
2287     {
2288         \seq_gclear:N \g_tmpa_seq
2289         \__zrefclever_opt_seq_gset_clist_split:Nn
2290         \g_tmpa_seq {##1}
2291         \bool_lazy_or:nnTF
2292         { \tl_if_empty_p:n {##1} }
2293         {
2294             \int_compare_p:nNn
2295             { \seq_count:N \g_tmpa_seq } = { 4 }
2296         }
2297         {
2298             \seq_gset_eq:cN
2299             {
2300                 \__zrefclever_opt_varname_lang_type:eenn
2301                 { \l__zrefclever_setup_language_tl }
2302                 { \l__zrefclever_setup_type_tl } {#1} { seq }
2303             }
2304             \g_tmpa_seq
2305         }
2306         {
2307             \msg_warning:nxxx { zref-clever }
2308             { refbounds-must-be-four }
2309             {#1} { \seq_count:N \g_tmpa_seq }
2310         }
2311     }
2312 }
2313 } ,
2314 }

```

```

2315 }
2316 \seq_map_inline:Nn
2317   \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
2318   {
2319     \keys_define:nn { zref-clever/langsetup }
2320     {
2321       #1 .choice: ,
2322       #1 / true .code:n =
2323       {
2324         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2325         {
2326           \bool_gset_true:c
2327           {
2328             \__zrefclever_opt_varname_lang_default:enn
2329             { \l__zrefclever_setup_language_tl }
2330             {#1} { bool }
2331           }
2332         }
2333         {
2334           \bool_gset_true:c
2335           {
2336             \__zrefclever_opt_varname_lang_type:enn
2337             { \l__zrefclever_setup_language_tl }
2338             { \l__zrefclever_setup_type_tl }
2339             {#1} { bool }
2340           }
2341         }
2342       } ,
2343       #1 / false .code:n =
2344       {
2345         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2346         {
2347           \bool_gset_false:c
2348           {
2349             \__zrefclever_opt_varname_lang_default:enn
2350             { \l__zrefclever_setup_language_tl }
2351             {#1} { bool }
2352           }
2353         }
2354         {
2355           \bool_gset_false:c
2356           {
2357             \__zrefclever_opt_varname_lang_type:enn
2358             { \l__zrefclever_setup_language_tl }
2359             { \l__zrefclever_setup_type_tl }
2360             {#1} { bool }
2361           }
2362         }
2363       } ,
2364       #1 / unset .code:n =
2365       {
2366         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2367         {
2368           \__zrefclever_opt_bool_gunset:c

```

```

2369         {
2370             \_zrefclever_opt_varname_lang_default:enn
2371             { \l\_zrefclever_setup_language_tl }
2372             {#1} { bool }
2373         }
2374     }
2375     {
2376         \_zrefclever_opt_bool_gunset:c
2377         {
2378             \_zrefclever_opt_varname_lang_type:enn
2379             { \l\_zrefclever_setup_language_tl }
2380             { \l\_zrefclever_setup_type_tl }
2381             {#1} { bool }
2382         }
2383     }
2384 } ,
2385 #1 .default:n = true ,
2386 no #1 .meta:n = { #1 = false } ,
2387 no #1 .value_forbidden:n = true ,
2388 }
2389 }

```

6 User interface

6.1 \zcref

\zcref The main user command of the package.

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

```

2390 \NewDocumentCommand \zcref { s O { } m }
2391 { \zref@wrapper@babel \_zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for \zcref.)

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

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

```

2392 \cs_new_protected:Npn \_zrefclever_zcref:nnn #1#2#3
2393 {
2394     \group_begin:

```

Set options.

```
2395     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

2396     \seq_set_from_clist:Nn \l\_zrefclever_zcref_labels_seq {#1}
2397     \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.

```
2398 \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
2399 \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
2400 \bool_lazy_and:nnT
2401 { \l__zrefclever_zrefcheck_available_bool }
2402 { \l__zrefclever_zcref_with_check_bool }
2403 { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
2404 \bool_lazy_or:nnT
2405 { \l__zrefclever_typeset_sort_bool }
2406 { \l__zrefclever_typeset_range_bool }
2407 { \__zrefclever_sort_labels: }
```

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

```
2408 \group_begin:
2409 \l__zrefclever_ref_typeset_font_tl
2410 \__zrefclever_typeset_refs:
2411 \group_end:
```

Typeset note.

```
2412 \tl_if_empty:NF \l__zrefclever_zcref_note_tl
2413 {
2414   \__zrefclever_get_rf_opt_tl:nxxN { notesep }
2415   { \l__zrefclever_label_type_a_tl }
2416   { \l__zrefclever_ref_language_tl }
2417   \l_tmpa_tl
2418   \l_tmpa_tl
2419   \l__zrefclever_zcref_note_tl
2420 }
```

Integration with `zref-check`.

```
2421 \bool_lazy_and:nnT
2422 { \l__zrefclever_zrefcheck_available_bool }
2423 { \l__zrefclever_zcref_with_check_bool }
2424 {
2425   \zrefcheck_zcref_end_label_maybe:
2426   \zrefcheck_zcref_run_checks_on_labels:n
2427   { \l__zrefclever_zcref_labels_seq }
2428 }
```

Integration with `mathtools`.

```
2429 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
2430 {
2431   \__zrefclever_mathtools_showonlyrefs:n
2432   { \l__zrefclever_zcref_labels_seq }
2433 }
2434 \group_end:
2435 }
```

(End definition for `_zrefclever_zcref:nmmn`.)

`\l_zrefclever_zcref_labels_seq`
`\l_zrefclever_link_star_bool`

2436 `\seq_new:N \l__zrefclever_zcref_labels_seq`
 2437 `\bool_new:N \l__zrefclever_link_star_bool`

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

6.2 `\zcpageref`

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

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

2438 `\NewDocumentCommand \zcpageref { s O { } m }`
 2439 `{`
 2440 `\group_begin:`
 2441 `\IfBooleanT {#1}`
 2442 `{ \bool_set_false:N \l__zrefclever_hyperlink_bool }`
 2443 `\zcref [#2, ref = page] {#3}`
 2444 `\group_end:`
 2445 `}`

(End definition for `\zcpageref`.)

7 Sorting

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

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

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

2446 `\tl_new:N \l__zrefclever_label_type_a_tl`
 2447 `\tl_new:N \l__zrefclever_label_type_b_tl`
 2448 `\tl_new:N \l__zrefclever_label_enclval_a_tl`
 2449 `\tl_new:N \l__zrefclever_label_enclval_b_tl`
 2450 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`
 2451 `\tl_new:N \l__zrefclever_label_extdoc_b_tl`

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

`\l_zrefclever_sort_decided_bool`

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

2452 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End definition for \l__zrefclever_sort_decided_bool.)

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

```
2453 \int_new:N \l__zrefclever_sort_prior_a_int
2454 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for \l__zrefclever_sort_prior_a_int and \l__zrefclever_sort_prior_b_int.)

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

```
2455 \seq_new:N \l__zrefclever_label_types_seq
```

(End definition for \l__zrefclever_label_types_seq.)

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

```
2456 \cs_new_protected:Npn \__zrefclever_sort_labels:
2457 {
```

Store label types sequence.

```
2458 \seq_clear:N \l__zrefclever_label_types_seq
2459 \tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }
2460 {
2461 \seq_map_function:NN \l__zrefclever_zcref_labels_seq
2462 \__zrefclever_label_type_put_new_right:n
2463 }
```

Sort.

```
2464 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
2465 {
2466 \zref@ifrefundefined {##1}
2467 {
2468 \zref@ifrefundefined {##2}
2469 {
2470 % Neither label is defined.
2471 \sort_return_same:
2472 }
2473 {
2474 % The second label is defined, but the first isn't, leave the
2475 % undefined first (to be more visible).
2476 \sort_return_same:
2477 }
2478 }
2479 {
2480 \zref@ifrefundefined {##2}
2481 {
2482 % The first label is defined, but the second isn't, bring the
```

```

2483             % second forward.
2484             \sort_return_swapped:
2485         }
2486     {
2487         % The interesting case: both labels are defined.  References
2488         % to the "default" property or to the "page" are quite
2489         % different with regard to sorting, so we branch them here to
2490         % specialized functions.
2491         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2492             { \__zrefclever_sort_page:nn {##1} {##2} }
2493             { \__zrefclever_sort_default:nn {##1} {##2} }
2494     }
2495 }
2496 }
2497 }

```

(End definition for __zrefclever_sort_labels:.)

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

```

\__zrefclever_label_type_put_new_right:n {\label}

2498 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
2499 {
2500     \__zrefclever_extract_default:Nnnn
2501     \l__zrefclever_label_type_a_tl {#1} {zc@type} {}
2502     \seq_if_in:NVF \l__zrefclever_label_types_seq
2503     \l__zrefclever_label_type_a_tl
2504     {
2505         \seq_put_right:NV \l__zrefclever_label_types_seq
2506         \l__zrefclever_label_type_a_tl
2507     }
2508 }

```

(End 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}

2509 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
2510 {
2511     \__zrefclever_extract_default:Nnnn
2512     \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}

```

```

2513 \__zrefclever_extract_default:Nnnn
2514 \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
2515
2516 \tl_if_eq:NNTF
2517 \l__zrefclever_label_type_a_tl
2518 \l__zrefclever_label_type_b_tl
2519 { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
2520 { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
2521 }

```

(End definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
2522 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
2523 {
2524 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
2525 {#1} {zc@enclval} { }
2526 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
2527 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
2528 {#2} {zc@enclval} { }
2529 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
2530 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
2531 {#1} {externaldocument} { }
2532 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
2533 {#2} {externaldocument} { }
2534
2535 \bool_set_false:N \l__zrefclever_sort_decided_bool
2536
2537 % First we check if there's any "external document" difference (coming
2538 % from 'zref-xr') and, if so, sort based on that.
2539 \tl_if_eq:NNTF
2540 \l__zrefclever_label_extdoc_a_tl
2541 \l__zrefclever_label_extdoc_b_tl
2542 {
2543 \bool_if:nTF
2544 {
2545 \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
2546 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
2547 }
2548 {
2549 \bool_set_true:N \l__zrefclever_sort_decided_bool
2550 \sort_return_same:
2551 }
2552 {
2553 \bool_if:nTF
2554 {
2555 ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
2556 \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
2557 }
2558 {
2559 \bool_set_true:N \l__zrefclever_sort_decided_bool
2560 \sort_return_swapped:
2561 }
2562 {

```

```

2563         \bool_set_true:N \l__zrefclever_sort_decided_bool
2564         % Two different "external documents": last resort, sort by the
2565         % document name itself.
2566         \str_compare:eNeTF
2567         { \l__zrefclever_label_extdoc_b_tl } <
2568         { \l__zrefclever_label_extdoc_a_tl }
2569         { \sort_return_swapped: }
2570         { \sort_return_same: }
2571     }
2572 }
2573 }
2574
2575 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
2576 {
2577     \bool_if:nTF
2578     {
2579         % Both are empty: neither label has any (further) "enclosing
2580         % counters" (left).
2581         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
2582         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
2583     }
2584     {
2585         \bool_set_true:N \l__zrefclever_sort_decided_bool
2586         \int_compare:nNnTF
2587         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2588         >
2589         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2590         { \sort_return_swapped: }
2591         { \sort_return_same: }
2592     }
2593     {
2594         \bool_if:nTF
2595         {
2596             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
2597             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
2598         }
2599         {
2600             \bool_set_true:N \l__zrefclever_sort_decided_bool
2601             \int_compare:nNnTF
2602             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
2603             >
2604             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2605             { \sort_return_swapped: }
2606             { \sort_return_same: }
2607         }
2608         {
2609             \bool_if:nTF
2610             {
2611                 % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
2612                 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
2613             }
2614             {
2615                 \bool_set_true:N \l__zrefclever_sort_decided_bool
2616                 \int_compare:nNnTF

```

```

2617         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2618         <
2619         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
2620         { \sort_return_same:    }
2621         { \sort_return_swapped: }
2622     }
2623     {
2624         % Neither is empty: we can compare the values of the
2625         % current enclosing counter in the loop, if they are
2626         % equal, we are still in the loop, if they are not, a
2627         % sorting decision can be made directly.
2628         \int_compare:nNnTF
2629         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2630         =
2631         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2632         {
2633             \tl_set:Nx \l__zrefclever_label_enclval_a_tl
2634             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
2635             \tl_set:Nx \l__zrefclever_label_enclval_b_tl
2636             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
2637         }
2638         {
2639             \bool_set_true:N \l__zrefclever_sort_decided_bool
2640             \int_compare:nNnTF
2641             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2642             >
2643             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2644             { \sort_return_swapped: }
2645             { \sort_return_same:    }
2646         }
2647     }
2648 }
2649 }
2650 }
2651 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

```

__zrefclever_sort_default_different_types:nn
2652 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
2653 {

```

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

```

2654     \int_zero:N \l__zrefclever_sort_prior_a_int
2655     \int_zero:N \l__zrefclever_sort_prior_b_int
2656     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
2657     {
2658         \tl_if_eq:nnTF {##2} {{othertypes}}
2659         {
2660             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
2661             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - #1 } }
2662             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }

```

```

2663         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
2664     }
2665     {
2666         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
2667         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
2668         {
2669             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
2670             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
2671         }
2672     }
2673 }

```

Then do the actual sorting.

```

2674 \bool_if:nTF
2675 {
2676     \int_compare_p:nNn
2677     { \l__zrefclever_sort_prior_a_int } <
2678     { \l__zrefclever_sort_prior_b_int }
2679 }
2680 { \sort_return_same: }
2681 {
2682     \bool_if:nTF
2683     {
2684         \int_compare_p:nNn
2685         { \l__zrefclever_sort_prior_a_int } >
2686         { \l__zrefclever_sort_prior_b_int }
2687     }
2688     { \sort_return_swapped: }
2689     {
2690         % Sort priorities are equal: the type that occurs first in
2691         % ‘labels’, as given by the user, is kept (or brought) forward.
2692         \seq_map_inline:Nn \l__zrefclever_label_types_seq
2693         {
2694             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
2695             { \seq_map_break:n { \sort_return_same: } }
2696             {
2697                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
2698                 { \seq_map_break:n { \sort_return_swapped: } }
2699             }
2700         }
2701     }
2702 }
2703 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

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

```

2704 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
2705 {
2706   \int_compare:nNnTF
2707     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
2708     >
2709     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
2710     { \sort_return_swapped: }
2711     { \sort_return_same: }
2712 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

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

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

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

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

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

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

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

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

```
\l__zrefclever_typeset_labels_seq
\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool
2713 \seq_new:N \l__zrefclever_typeset_labels_seq
2714 \bool_new:N \l__zrefclever_typeset_last_bool
2715 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

```
\l__zrefclever_type_count_int
\l__zrefclever_label_count_int
\l__zrefclever_ref_count_int
2716 \int_new:N \l__zrefclever_type_count_int
2717 \int_new:N \l__zrefclever_label_count_int
2718 \int_new:N \l__zrefclever_ref_count_int
```

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

`\l__zrefclever_label_a_tl`
`\l__zrefclever_label_b_tl`
`\l__zrefclever_typeset_queue_prev_tl`
`\l__zrefclever_typeset_queue_curr_tl`
`\l__zrefclever_type_first_label_tl`
`\l__zrefclever_type_first_label_type_tl`

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

`\tl_new:N \l__zrefclever_label_a_tl`
`\tl_new:N \l__zrefclever_label_b_tl`
`\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
`\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
`\tl_new:N \l__zrefclever_type_first_label_tl`
`\tl_new:N \l__zrefclever_type_first_label_type_tl`

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

`\l__zrefclever_type_name_tl`
`\l__zrefclever_name_in_link_bool`
`\l__zrefclever_type_name_missing_bool`
`\l__zrefclever_name_format_tl`
`\l__zrefclever_name_format_fallback_tl`
`\l__zrefclever_type_name_gender_seq`

Auxiliary variables for `__zrefclever_typeset_refs`: type name handling.

`\tl_new:N \l__zrefclever_type_name_tl`
`\bool_new:N \l__zrefclever_name_in_link_bool`
`\bool_new:N \l__zrefclever_type_name_missing_bool`
`\tl_new:N \l__zrefclever_name_format_tl`
`\tl_new:N \l__zrefclever_name_format_fallback_tl`
`\seq_new:N \l__zrefclever_type_name_gender_seq`

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

`\l__zrefclever_range_count_int`
`\l__zrefclever_range_same_count_int`
`\l__zrefclever_range_beg_label_tl`
`\l__zrefclever_next_maybe_range_bool`
`\l__zrefclever_next_is_same_bool`

Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

`\int_new:N \l__zrefclever_range_count_int`
`\int_new:N \l__zrefclever_range_same_count_int`
`\tl_new:N \l__zrefclever_range_beg_label_tl`
`\bool_new:N \l__zrefclever_next_maybe_range_bool`
`\bool_new:N \l__zrefclever_next_is_same_bool`

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

`\l__zrefclever_tpairsep_tl`
`\l__zrefclever_tlistsep_tl`
`\l__zrefclever_tlastsep_tl`
`\l__zrefclever_namesep_tl`
`\l__zrefclever_pairsep_tl`
`\l__zrefclever_listsep_tl`
`\l__zrefclever_lastsep_tl`
`\l__zrefclever_rangesep_tl`
`\l__zrefclever_namefont_tl`
`\l__zrefclever_reffont_tl`
`\l__zrefclever_cap_bool`
`\l__zrefclever_abbrev_bool`

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

`\tl_new:N \l__zrefclever_tpairsep_tl`
`\tl_new:N \l__zrefclever_tlistsep_tl`
`\tl_new:N \l__zrefclever_tlastsep_tl`
`\tl_new:N \l__zrefclever_namesep_tl`
`\tl_new:N \l__zrefclever_pairsep_tl`
`\tl_new:N \l__zrefclever_listsep_tl`
`\tl_new:N \l__zrefclever_lastsep_tl`
`\tl_new:N \l__zrefclever_rangesep_tl`
`\tl_new:N \l__zrefclever_namefont_tl`
`\tl_new:N \l__zrefclever_reffont_tl`
`\bool_new:N \l__zrefclever_cap_bool`
`\bool_new:N \l__zrefclever_abbrev_bool`

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

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

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

`\seq_new:N \l__zrefclever_refbounds_first_seq`
`\seq_new:N \l__zrefclever_refbounds_first_sg_seq`
`\seq_new:N \l__zrefclever_refbounds_first_pb_seq`
`\seq_new:N \l__zrefclever_refbounds_first_rb_seq`
`\seq_new:N \l__zrefclever_refbounds_mid_seq`

```

2753 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
2754 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
2755 \seq_new:N \l__zrefclever_refbounds_last_seq
2756 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
2757 \seq_new:N \l__zrefclever_refbounds_last_re_seq
2758 \seq_new:N \l__zrefclever_type_first_refbounds_seq
2759 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

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

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

```

2760 \bool_new:N \l__zrefclever_verbose_testing_bool

```

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```

2761 \cs_new_protected:Npn \__zrefclever_typeset_refs:
2762 {
2763   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
2764   \l__zrefclever_zceref_labels_seq
2765   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
2766   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2767   \tl_clear:N \l__zrefclever_type_first_label_tl
2768   \tl_clear:N \l__zrefclever_type_first_label_type_tl
2769   \tl_clear:N \l__zrefclever_range_beg_label_tl
2770   \int_zero:N \l__zrefclever_label_count_int
2771   \int_zero:N \l__zrefclever_type_count_int
2772   \int_zero:N \l__zrefclever_ref_count_int
2773   \int_zero:N \l__zrefclever_range_count_int
2774   \int_zero:N \l__zrefclever_range_same_count_int
2775   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
2776
2777   % Get type block options (not type-specific).
2778   \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
2779   { \l__zrefclever_label_type_a_tl }
2780   { \l__zrefclever_ref_language_tl }
2781   \l__zrefclever_tpairsep_tl
2782   \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
2783   { \l__zrefclever_label_type_a_tl }
2784   { \l__zrefclever_ref_language_tl }
2785   \l__zrefclever_tlistsep_tl
2786   \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
2787   { \l__zrefclever_label_type_a_tl }
2788   { \l__zrefclever_ref_language_tl }
2789   \l__zrefclever_tlastsep_tl
2790
2791   % Process label stack.
2792   \bool_set_false:N \l__zrefclever_typeset_last_bool
2793   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
2794   {

```

```

2795 \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
2796 \l__zrefclever_label_a_tl
2797 \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
2798 {
2799   \tl_clear:N \l__zrefclever_label_b_tl
2800   \bool_set_true:N \l__zrefclever_typeset_last_bool
2801 }
2802 {
2803   \seq_get_left:NN \l__zrefclever_typeset_labels_seq
2804   \l__zrefclever_label_b_tl
2805 }
2806
2807 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2808 {
2809   \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
2810   \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
2811 }
2812 {
2813   \__zrefclever_extract_default:NVnn
2814   \l__zrefclever_label_type_a_tl
2815   \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
2816   \__zrefclever_extract_default:NVnn
2817   \l__zrefclever_label_type_b_tl
2818   \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
2819 }
2820
2821 % First, we establish whether the "current label" (i.e. 'a') is the
2822 % last one of its type. This can happen because the "next label"
2823 % (i.e. 'b') is of a different type (or different definition status),
2824 % or because we are at the end of the list.
2825 \bool_if:NTF \l__zrefclever_typeset_last_bool
2826 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2827 {
2828   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2829   {
2830     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2831     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2832     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2833   }
2834   {
2835     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2836     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2837     {
2838       % Neither is undefined, we must check the types.
2839       \tl_if_eq:NNTF
2840         \l__zrefclever_label_type_a_tl
2841         \l__zrefclever_label_type_b_tl
2842         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2843         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2844     }
2845   }
2846 }
2847
2848 % Handle warnings in case of reference or type undefined.

```

```

2849 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
2850 \zref@refused { \l__zrefclever_label_a_tl }
2851 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
2852 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2853 {}
2854 {
2855   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
2856   {
2857     \msg_warning:nxx { zref-clever } { missing-type }
2858     { \l__zrefclever_label_a_tl }
2859   }
2860   \zref@ifrefcontainsprop
2861   { \l__zrefclever_label_a_tl }
2862   { \l__zrefclever_ref_property_tl }
2863   { }
2864   {
2865     \msg_warning:nxxx { zref-clever } { missing-property }
2866     { \l__zrefclever_ref_property_tl }
2867     { \l__zrefclever_label_a_tl }
2868   }
2869 }
2870
2871 % Get possibly type-specific separators, refbounds, font and other
2872 % options, once per type.
2873 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
2874 {
2875   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
2876   { \l__zrefclever_label_type_a_tl }
2877   { \l__zrefclever_ref_language_tl }
2878   \l__zrefclever_namesep_tl
2879   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
2880   { \l__zrefclever_label_type_a_tl }
2881   { \l__zrefclever_ref_language_tl }
2882   \l__zrefclever_pairsep_tl
2883   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
2884   { \l__zrefclever_label_type_a_tl }
2885   { \l__zrefclever_ref_language_tl }
2886   \l__zrefclever_listsep_tl
2887   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
2888   { \l__zrefclever_label_type_a_tl }
2889   { \l__zrefclever_ref_language_tl }
2890   \l__zrefclever_lastsep_tl
2891   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
2892   { \l__zrefclever_label_type_a_tl }
2893   { \l__zrefclever_ref_language_tl }
2894   \l__zrefclever_rangesep_tl
2895   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
2896   { \l__zrefclever_label_type_a_tl }
2897   { \l__zrefclever_ref_language_tl }
2898   \l__zrefclever_namefont_tl
2899   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
2900   { \l__zrefclever_label_type_a_tl }
2901   { \l__zrefclever_ref_language_tl }
2902   \l__zrefclever_reffont_tl

```

```

2903 \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
2904 { \l__zrefclever_label_type_a_tl }
2905 { \l__zrefclever_ref_language_tl }
2906 \l__zrefclever_cap_bool
2907 \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
2908 { \l__zrefclever_label_type_a_tl }
2909 { \l__zrefclever_ref_language_tl }
2910 \l__zrefclever_abbrev_bool
2911 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
2912 { \l__zrefclever_label_type_a_tl }
2913 { \l__zrefclever_ref_language_tl }
2914 \l__zrefclever_refbounds_first_seq
2915 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }
2916 { \l__zrefclever_label_type_a_tl }
2917 { \l__zrefclever_ref_language_tl }
2918 \l__zrefclever_refbounds_first_sg_seq
2919 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
2920 { \l__zrefclever_label_type_a_tl }
2921 { \l__zrefclever_ref_language_tl }
2922 \l__zrefclever_refbounds_first_pb_seq
2923 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
2924 { \l__zrefclever_label_type_a_tl }
2925 { \l__zrefclever_ref_language_tl }
2926 \l__zrefclever_refbounds_first_rb_seq
2927 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
2928 { \l__zrefclever_label_type_a_tl }
2929 { \l__zrefclever_ref_language_tl }
2930 \l__zrefclever_refbounds_mid_seq
2931 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
2932 { \l__zrefclever_label_type_a_tl }
2933 { \l__zrefclever_ref_language_tl }
2934 \l__zrefclever_refbounds_mid_rb_seq
2935 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
2936 { \l__zrefclever_label_type_a_tl }
2937 { \l__zrefclever_ref_language_tl }
2938 \l__zrefclever_refbounds_mid_re_seq
2939 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
2940 { \l__zrefclever_label_type_a_tl }
2941 { \l__zrefclever_ref_language_tl }
2942 \l__zrefclever_refbounds_last_seq
2943 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
2944 { \l__zrefclever_label_type_a_tl }
2945 { \l__zrefclever_ref_language_tl }
2946 \l__zrefclever_refbounds_last_pe_seq
2947 \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
2948 { \l__zrefclever_label_type_a_tl }
2949 { \l__zrefclever_ref_language_tl }
2950 \l__zrefclever_refbounds_last_re_seq
2951 }
2952
2953 % Here we send this to a couple of auxiliary functions.
2954 \bool_if:NTF \l__zrefclever_last_of_type_bool
2955 % There exists no next label of the same type as the current.
2956 { \__zrefclever_typeset_refs_last_of_type: }

```

```

2957         % There exists a next label of the same type as the current.
2958         { \_zrefclever_typeset_refs_not_last_of_type: }
2959     }
2960 }

```

(End definition for _zrefclever_typeset_refs:.)

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

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

```

2961 \cs_new_protected:Npn \_zrefclever_typeset_refs_last_of_type:
2962 {
2963     % Process the current label to the current queue.
2964     \int_case:nnF { \l_zrefclever_label_count_int }
2965     {
2966         % It is the last label of its type, but also the first one, and that's
2967         % what matters here: just store it.
2968         % Test: 'zc-typeset01.lvt': "Last of type: single"
2969         { 0 }
2970     {
2971         \tl_set:NV \l_zrefclever_type_first_label_tl
2972         \l_zrefclever_label_a_tl
2973         \tl_set:NV \l_zrefclever_type_first_label_type_tl
2974         \l_zrefclever_label_type_a_tl
2975         \seq_set_eq:NN \l_zrefclever_type_first_refbounds_seq
2976         \l_zrefclever_refbounds_first_sg_seq
2977         \bool_set_true:N \l_zrefclever_type_first_refbounds_set_bool
2978     }
2979
2980     % The last is the second: we have a pair (if not repeated).
2981     % Test: 'zc-typeset01.lvt': "Last of type: pair"
2982     { 1 }
2983     {
2984         \int_compare:nNnTF { \l_zrefclever_range_same_count_int } = { 1 }
2985         {
2986             \seq_set_eq:NN \l_zrefclever_type_first_refbounds_seq
2987             \l_zrefclever_refbounds_first_sg_seq
2988             \bool_set_true:N \l_zrefclever_type_first_refbounds_set_bool
2989         }
2990         {
2991             \tl_put_right:Nx \l_zrefclever_typeset_queue_curr_tl
2992             {
2993                 \exp_not:V \l_zrefclever_pairsep_tl
2994                 \_zrefclever_get_ref:VN \l_zrefclever_label_a_tl
2995                 \l_zrefclever_refbounds_last_pe_seq
2996             }
2997             \seq_set_eq:NN \l_zrefclever_type_first_refbounds_seq

```

```

2998         \l__zrefclever_refbounds_first_pb_seq
2999         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3000     }
3001 }
3002 }
3003 % Last is third or more of its type: without repetition, we'd have the
3004 % last element on a list, but control for possible repetition.
3005 {
3006     \int_case:nnF { \l__zrefclever_range_count_int }
3007     {
3008         % There was no range going on.
3009         % Test: 'zc-typeset01.lvt': "Last of type: not range"
3010         { 0 }
3011         {
3012             \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
3013             {
3014                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3015                 {
3016                     \exp_not:V \l__zrefclever_pairsep_tl
3017                     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3018                     \l__zrefclever_refbounds_last_pe_seq
3019                 }
3020             }
3021             {
3022                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3023                 {
3024                     \exp_not:V \l__zrefclever_lastsep_tl
3025                     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3026                     \l__zrefclever_refbounds_last_seq
3027                 }
3028             }
3029         }
3030         % Last in the range is also the second in it.
3031         % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
3032         { 1 }
3033         {
3034             \int_compare:nNnTF
3035             { \l__zrefclever_range_same_count_int } = { 1 }
3036             {
3037                 % We know 'range_beg_label' is not empty, since this is the
3038                 % second element in the range, but the third or more in the
3039                 % type list.
3040                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3041                 {
3042                     \exp_not:V \l__zrefclever_pairsep_tl
3043                     \__zrefclever_get_ref:VN
3044                     \l__zrefclever_range_beg_label_tl
3045                     \l__zrefclever_refbounds_last_pe_seq
3046                 }
3047                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3048                 \l__zrefclever_refbounds_first_pb_seq
3049                 \bool_set_true:N
3050                 \l__zrefclever_type_first_refbounds_set_bool
3051             }

```

```

3052     {
3053         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3054         {
3055             \exp_not:V \l__zrefclever_listsep_tl
3056             \__zrefclever_get_ref:VN
3057             \l__zrefclever_range_beg_label_tl
3058             \l__zrefclever_refbounds_mid_seq
3059             \exp_not:V \l__zrefclever_lastsep_tl
3060             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3061             \l__zrefclever_refbounds_last_seq
3062         }
3063     }
3064 }
3065 }
3066 % Last in the range is third or more in it.
3067 {
3068     \int_case:nnF
3069     {
3070         \l__zrefclever_range_count_int -
3071         \l__zrefclever_range_same_count_int
3072     }
3073     {
3074         % Repetition, not a range.
3075         % Test: 'zc-typeset01.lvt': "Last of type: range to one"
3076         { 0 }
3077         {
3078             % If 'range_beg_label' is empty, it means it was also the
3079             % first of the type, and hence its typesetting was already
3080             % handled, and we just have to set refbounds.
3081             \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3082             {
3083                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3084                 \l__zrefclever_refbounds_first_sg_seq
3085                 \bool_set_true:N
3086                 \l__zrefclever_type_first_refbounds_set_bool
3087             }
3088             {
3089                 \int_compare:nNnTF
3090                 { \l__zrefclever_ref_count_int } < { 2 }
3091                 {
3092                     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3093                     {
3094                         \exp_not:V \l__zrefclever_pairsep_tl
3095                         \__zrefclever_get_ref:VN
3096                         \l__zrefclever_range_beg_label_tl
3097                         \l__zrefclever_refbounds_last_pe_seq
3098                     }
3099                 }
3100                 {
3101                     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3102                     {
3103                         \exp_not:V \l__zrefclever_lastsep_tl
3104                         \__zrefclever_get_ref:VN
3105                         \l__zrefclever_range_beg_label_tl

```



```

3106         \l__zrefclever_refbounds_last_seq
3107     }
3108 }
3109 }
3110 }
3111 % A 'range', but with no skipped value, treat as pair if range
3112 % started with first of type, otherwise as list.
3113 % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
3114 { 1 }
3115 {
3116     % Ditto.
3117     \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3118     {
3119         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3120         \l__zrefclever_refbounds_first_pb_seq
3121         \bool_set_true:N
3122         \l__zrefclever_type_first_refbounds_set_bool
3123         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3124         {
3125             \exp_not:V \l__zrefclever_pairsep_tl
3126             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3127             \l__zrefclever_refbounds_last_pe_seq
3128         }
3129     }
3130     {
3131         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3132         {
3133             \exp_not:V \l__zrefclever_listsep_tl
3134             \__zrefclever_get_ref:VN
3135             \l__zrefclever_range_beg_label_tl
3136             \l__zrefclever_refbounds_mid_seq
3137         }
3138         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3139         {
3140             \exp_not:V \l__zrefclever_lastsep_tl
3141             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3142             \l__zrefclever_refbounds_last_seq
3143         }
3144     }
3145 }
3146 }
3147 {
3148     % An actual range.
3149     % Test: 'zc-typeset01.lvt': "Last of type: range"
3150     % Ditto.
3151     \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3152     {
3153         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3154         \l__zrefclever_refbounds_first_rb_seq
3155         \bool_set_true:N
3156         \l__zrefclever_type_first_refbounds_set_bool
3157     }
3158     {
3159         \int_compare:nNnTF

```

```

3160         { \l__zrefclever_ref_count_int } < { 2 }
3161     {
3162         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3163         {
3164             \exp_not:V \l__zrefclever_pairsep_tl
3165             \__zrefclever_get_ref:VN
3166             \l__zrefclever_range_beg_label_tl
3167             \l__zrefclever_refbounds_mid_rb_seq
3168         }
3169         \seq_set_eq:NN
3170         \l__zrefclever_type_first_refbounds_seq
3171         \l__zrefclever_refbounds_first_pb_seq
3172         \bool_set_true:N
3173         \l__zrefclever_type_first_refbounds_set_bool
3174     }
3175     {
3176         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3177         {
3178             \exp_not:V \l__zrefclever_lastsep_tl
3179             \__zrefclever_get_ref:VN
3180             \l__zrefclever_range_beg_label_tl
3181             \l__zrefclever_refbounds_mid_rb_seq
3182         }
3183     }
3184 }
3185 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3186 {
3187     \exp_not:V \l__zrefclever_rangesep_tl
3188     \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3189     \l__zrefclever_refbounds_last_re_seq
3190 }
3191 }
3192 }
3193 }
3194
3195 % Handle "range" option. The idea is simple: if the queue is not empty,
3196 % we replace it with the end of the range (or pair). We can still
3197 % retrieve the end of the range from 'label_a' since we know to be
3198 % processing the last label of its type at this point.
3199 \bool_if:NT \l__zrefclever_typeset_range_bool
3200 {
3201     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
3202     {
3203         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
3204         { }
3205         {
3206             \msg_warning:nxx { zref-clever } { single-element-range }
3207             { \l__zrefclever_type_first_label_type_tl }
3208         }
3209     }
3210     {
3211         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
3212         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
3213         { }

```

```

3214         {
3215             \__zrefclever_labels_in_sequence:nn
3216             { \l__zrefclever_type_first_label_tl }
3217             { \l__zrefclever_label_a_tl }
3218         }
3219         % Test: 'zc-typeset01.lvt': "Last of type: option range"
3220         % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
3221         \bool_if:NTF \l__zrefclever_next_maybe_range_bool
3222         {
3223             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
3224             {
3225                 \exp_not:V \l__zrefclever_pairsep_tl
3226                 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3227                 \l__zrefclever_refbounds_last_pe_seq
3228             }
3229             \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3230             \l__zrefclever_refbounds_first_pb_seq
3231             \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3232         }
3233         {
3234             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
3235             {
3236                 \exp_not:V \l__zrefclever_rangesep_tl
3237                 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3238                 \l__zrefclever_refbounds_last_re_seq
3239             }
3240             \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3241             \l__zrefclever_refbounds_first_rb_seq
3242             \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3243         }
3244     }
3245 }
3246
3247 % If none of the special cases for the first of type refbounds have been
3248 % set, do it.
3249 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
3250 {
3251     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3252     \l__zrefclever_refbounds_first_seq
3253 }
3254
3255 % Now that the type block is finished, we can add the name and the first
3256 % ref to the queue. Also, if "typeset" option is not "both", handle it
3257 % here as well.
3258 \__zrefclever_type_name_setup:
3259 \bool_if:nTF
3260 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
3261 {
3262     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
3263     { \__zrefclever_get_ref_first: }
3264 }
3265 {
3266     \bool_if:NTF \l__zrefclever_typeset_ref_bool
3267     {

```

```

3268 % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
3269 \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
3270 {
3271   \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
3272   \l__zrefclever_type_first_refbounds_seq
3273 }
3274 }
3275 {
3276   \bool_if:NTF \l__zrefclever_typeset_name_bool
3277   {
3278     % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
3279     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
3280     {
3281       \bool_if:NTF \l__zrefclever_name_in_link_bool
3282       {
3283         \exp_not:N \group_begin:
3284         \exp_not:V \l__zrefclever_namefont_tl
3285         % It's two '@s', but escaped for DocStrip.
3286         \exp_not:N \hyper@@link
3287         {
3288           \__zrefclever_extract_url_unexp:V
3289           \l__zrefclever_type_first_label_tl
3290         }
3291         {
3292           \__zrefclever_extract_unexp:Vnn
3293           \l__zrefclever_type_first_label_tl
3294           { anchor } { }
3295         }
3296         { \exp_not:V \l__zrefclever_type_name_tl }
3297         \exp_not:N \group_end:
3298       }
3299       {
3300         \exp_not:N \group_begin:
3301         \exp_not:V \l__zrefclever_namefont_tl
3302         \exp_not:V \l__zrefclever_type_name_tl
3303         \exp_not:N \group_end:
3304       }
3305     }
3306   }
3307   {
3308     % Logically, this case would correspond to "typeset=none", but
3309     % it should not occur, given that the options are set up to
3310     % typeset either "ref" or "name". Still, leave here a
3311     % sensible fallback, equal to the behavior of "both".
3312     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
3313     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
3314     { \__zrefclever_get_ref_first: }
3315   }
3316 }
3317 }
3318
3319 % Typeset the previous type block, if there is one.
3320 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
3321 {

```

```

3322     \int_compare:nNt { \l__zrefclever_type_count_int } > { 1 }
3323     { \l__zrefclever_tlistsep_tl }
3324     \l__zrefclever_typeset_queue_prev_tl
3325   }
3326
3327   % Extra log for testing.
3328   \bool_if:NT \l__zrefclever_verbose_testing_bool
3329     { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
3330
3331   % Wrap up loop, or prepare for next iteration.
3332   \bool_if:NTF \l__zrefclever_typeset_last_bool
3333   {
3334     % We are finishing, typeset the current queue.
3335     \int_case:nnF { \l__zrefclever_type_count_int }
3336     {
3337       % Single type.
3338       % Test: 'zc-typeset01.lvt': "Last of type: single type"
3339       { 0 }
3340       { \l__zrefclever_typeset_queue_curr_tl }
3341       % Pair of types.
3342       % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
3343       { 1 }
3344       {
3345         \l__zrefclever_tpairsep_tl
3346         \l__zrefclever_typeset_queue_curr_tl
3347       }
3348     }
3349     {
3350       % Last in list of types.
3351       % Test: 'zc-typeset01.lvt': "Last of type: list of types"
3352       \l__zrefclever_tlastsep_tl
3353       \l__zrefclever_typeset_queue_curr_tl
3354     }
3355     % And nudge in case of multitype reference.
3356     \bool_lazy_all:nT
3357     {
3358       { \l__zrefclever_nudge_enabled_bool }
3359       { \l__zrefclever_nudge_multitype_bool }
3360       { \int_compare_p:nN { \l__zrefclever_type_count_int } > { 0 } }
3361     }
3362     { \msg_warning:nn { zref-clever } { nudge-multitype } }
3363   }
3364   {
3365     % There are further labels, set variables for next iteration.
3366     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
3367       \l__zrefclever_typeset_queue_curr_tl
3368     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3369     \tl_clear:N \l__zrefclever_type_first_label_tl
3370     \tl_clear:N \l__zrefclever_type_first_label_type_tl
3371     \tl_clear:N \l__zrefclever_range_beg_label_tl
3372     \int_zero:N \l__zrefclever_label_count_int
3373     \int_zero:N \l__zrefclever_ref_count_int
3374     \int_incr:N \l__zrefclever_type_count_int
3375     \int_zero:N \l__zrefclever_range_count_int

```

```

3376         \int_zero:N \l__zrefclever_range_same_count_int
3377         \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3378     }
3379 }

```

(End definition for __zrefclever_typeset_refs_last_of_type:.)

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

```

3380 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
3381 {
3382     % Signal if next label may form a range with the current one (only
3383     % considered if compression is enabled in the first place).
3384     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
3385     \bool_set_false:N \l__zrefclever_next_is_same_bool
3386     \bool_if:NT \l__zrefclever_typeset_compress_bool
3387     {
3388         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3389         { }
3390         {
3391             \__zrefclever_labels_in_sequence:nn
3392             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
3393         }
3394     }
3395
3396     % Process the current label to the current queue.
3397     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
3398     {
3399         % Current label is the first of its type (also not the last, but it
3400         % doesn't matter here): just store the label.
3401         \tl_set:NV \l__zrefclever_type_first_label_tl
3402             \l__zrefclever_label_a_tl
3403         \tl_set:NV \l__zrefclever_type_first_label_type_tl
3404             \l__zrefclever_label_type_a_tl
3405         \int_incr:N \l__zrefclever_ref_count_int
3406
3407         % If the next label may be part of a range, we set 'range_beg_label'
3408         % to "empty" (we deal with it as the "first", and must do it there, to
3409         % handle hyperlinking), but also step the range counters.
3410         % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
3411         \bool_if:NT \l__zrefclever_next_maybe_range_bool
3412         {
3413             \tl_clear:N \l__zrefclever_range_beg_label_tl
3414             \int_incr:N \l__zrefclever_range_count_int
3415             \bool_if:NT \l__zrefclever_next_is_same_bool
3416             { \int_incr:N \l__zrefclever_range_same_count_int }
3417         }
3418     }
3419     {
3420         % Current label is neither the first (nor the last) of its type.
3421         \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
3422         {
3423             % Starting, or continuing a range.
3424             \int_compare:nNnTF
3425             { \l__zrefclever_range_count_int } = { 0 }

```

```

3426 {
3427   % There was no range going, we are starting one.
3428   \tl_set:NV \l__zrefclever_range_beg_label_tl
3429   \l__zrefclever_label_a_tl
3430   \int_incr:N \l__zrefclever_range_count_int
3431   \bool_if:NT \l__zrefclever_next_is_same_bool
3432   { \int_incr:N \l__zrefclever_range_same_count_int }
3433 }
3434 {
3435   % Second or more in the range, but not the last.
3436   \int_incr:N \l__zrefclever_range_count_int
3437   \bool_if:NT \l__zrefclever_next_is_same_bool
3438   { \int_incr:N \l__zrefclever_range_same_count_int }
3439 }
3440 }
3441 {
3442   % Next element is not in sequence: there was no range, or we are
3443   % closing one.
3444   \int_case:nnF { \l__zrefclever_range_count_int }
3445   {
3446     % There was no range going on.
3447     % Test: 'zc-typeset01.lvt': "Not last of type: no range"
3448     { 0 }
3449     {
3450       \int_incr:N \l__zrefclever_ref_count_int
3451       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3452       {
3453         \exp_not:V \l__zrefclever_listsep_tl
3454         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3455         \l__zrefclever_refbounds_mid_seq
3456       }
3457     }
3458     % Last is second in the range: if 'range_same_count' is also
3459     % '1', it's a repetition (drop it), otherwise, it's a "pair
3460     % within a list", treat as list.
3461     % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
3462     % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
3463     { 1 }
3464     {
3465       \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3466       {
3467         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3468         \l__zrefclever_refbounds_first_seq
3469         \bool_set_true:N
3470         \l__zrefclever_type_first_refbounds_set_bool
3471       }
3472       {
3473         \int_incr:N \l__zrefclever_ref_count_int
3474         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3475         {
3476           \exp_not:V \l__zrefclever_listsep_tl
3477           \__zrefclever_get_ref:VN
3478           \l__zrefclever_range_beg_label_tl
3479           \l__zrefclever_refbounds_mid_seq

```

```

3480     }
3481   }
3482   \int_compare:nNnF
3483   { \l__zrefclever_range_same_count_int } = { 1 }
3484   {
3485     \int_incr:N \l__zrefclever_ref_count_int
3486     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3487     {
3488       \exp_not:V \l__zrefclever_listsep_tl
3489       \__zrefclever_get_ref:VN
3490       \l__zrefclever_label_a_tl
3491       \l__zrefclever_refbounds_mid_seq
3492     }
3493   }
3494 }
3495 }
3496 {
3497   % Last is third or more in the range: if 'range_count' and
3498   % 'range_same_count' are the same, its a repetition (drop it),
3499   % if they differ by '1', its a list, if they differ by more,
3500   % it is a real range.
3501   \int_case:nnF
3502   {
3503     \l__zrefclever_range_count_int -
3504     \l__zrefclever_range_same_count_int
3505   }
3506   {
3507     % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
3508     { 0 }
3509     {
3510       \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3511       {
3512         \seq_set_eq:NN
3513         \l__zrefclever_type_first_refbounds_seq
3514         \l__zrefclever_refbounds_first_seq
3515         \bool_set_true:N
3516         \l__zrefclever_type_first_refbounds_set_bool
3517       }
3518       {
3519         \int_incr:N \l__zrefclever_ref_count_int
3520         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3521         {
3522           \exp_not:V \l__zrefclever_listsep_tl
3523           \__zrefclever_get_ref:VN
3524           \l__zrefclever_range_beg_label_tl
3525           \l__zrefclever_refbounds_mid_seq
3526         }
3527       }
3528     }
3529     % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
3530     { 1 }
3531     {
3532       \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3533       {

```



```

3534         \seq_set_eq:NN
3535         \l__zrefclever_type_first_refbounds_seq
3536         \l__zrefclever_refbounds_first_seq
3537     \bool_set_true:N
3538         \l__zrefclever_type_first_refbounds_set_bool
3539     }
3540     {
3541         \int_incr:N \l__zrefclever_ref_count_int
3542         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3543         {
3544             \exp_not:V \l__zrefclever_listsep_tl
3545             \__zrefclever_get_ref:VN
3546             \l__zrefclever_range_beg_label_tl
3547             \l__zrefclever_refbounds_mid_seq
3548         }
3549     }
3550     \int_incr:N \l__zrefclever_ref_count_int
3551     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3552     {
3553         \exp_not:V \l__zrefclever_listsep_tl
3554         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3555         \l__zrefclever_refbounds_mid_seq
3556     }
3557 }
3558 }
3559 {
3560     % Test: 'zc-typeset01.lvt': "Not last of type: range"
3561     \tl_if_empty:VTF \l__zrefclever_range_beg_label_tl
3562     {
3563         \seq_set_eq:NN
3564         \l__zrefclever_type_first_refbounds_seq
3565         \l__zrefclever_refbounds_first_rb_seq
3566         \bool_set_true:N
3567         \l__zrefclever_type_first_refbounds_set_bool
3568     }
3569     {
3570         \int_incr:N \l__zrefclever_ref_count_int
3571         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3572         {
3573             \exp_not:V \l__zrefclever_listsep_tl
3574             \__zrefclever_get_ref:VN
3575             \l__zrefclever_range_beg_label_tl
3576             \l__zrefclever_refbounds_mid_rb_seq
3577         }
3578     }
3579     % For the purposes of the serial comma, and thus for the
3580     % distinction of 'lastsep' and 'pairsep', a "range" counts
3581     % as one. Since 'range_beg' has already been counted
3582     % (here or with the first of type), we refrain from
3583     % incrementing 'ref_count_int'.
3584     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3585     {
3586         \exp_not:V \l__zrefclever_rangeseq_tl
3587         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl

```

```

3588             \l__zrefclever_refbounds_mid_re_seq
3589         }
3590     }
3591 }
3592 % Reset counters.
3593 \int_zero:N \l__zrefclever_range_count_int
3594 \int_zero:N \l__zrefclever_range_same_count_int
3595 }
3596 }
3597 % Step label counter for next iteration.
3598 \int_incr:N \l__zrefclever_label_count_int
3599 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`)

Auxiliary functions

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

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

```

3600 \cs_new_protected:Npn \__zrefclever_ref_default:
3601 { \zref@default }
3602 \cs_new_protected:Npn \__zrefclever_name_default:
3603 { \zref@default }

```

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

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

```

    \__zrefclever_get_ref:nN {\label} {\refbounds}}
3604 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
3605 {
3606   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
3607   {
3608     \bool_if:nTF
3609     {
3610       \l__zrefclever_hyperlink_bool &&
3611       ! \l__zrefclever_link_star_bool
3612     }
3613     {
3614       \exp_not:N \group_begin:
3615       \exp_not:V \l__zrefclever_reffont_tl
3616       \seq_item:Nn #2 { 1 }
3617       % It's two '@s', but escaped for DocStrip.
3618       \exp_not:N \hyper@@link
3619       { \__zrefclever_extract_url_unexp:n {#1} }
3620       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
3621       {
3622         \seq_item:Nn #2 { 2 }
3623         \__zrefclever_extract_unexp:nvn {#1}
3624         { \l__zrefclever_ref_property_tl } { }
3625         \seq_item:Nn #2 { 3 }
3626       }
3627       \seq_item:Nn #2 { 4 }
3628       \exp_not:N \group_end:
3629     }
3630     {
3631       \exp_not:N \group_begin:
3632       \exp_not:V \l__zrefclever_reffont_tl
3633       \seq_item:Nn #2 { 1 }
3634       \seq_item:Nn #2 { 2 }
3635       \__zrefclever_extract_unexp:nvn {#1}
3636       { \l__zrefclever_ref_property_tl } { }
3637       \seq_item:Nn #2 { 3 }
3638       \seq_item:Nn #2 { 4 }
3639       \exp_not:N \group_end:
3640     }
3641   }
3642   { \__zrefclever_ref_default: }
3643 }
3644 \cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }

```

(End definition for __zrefclever_get_ref:nN.)

`__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.

```

3645 \cs_new:Npn \__zrefclever_get_ref_first:
3646 {
3647   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
3648   { \__zrefclever_ref_default: }
3649   {
3650     \bool_if:NTF \l__zrefclever_name_in_link_bool
3651     {
3652       \zref@ifrefcontainsprop
3653       { \l__zrefclever_type_first_label_tl }
3654       { \l__zrefclever_ref_property_tl }
3655       {
3656         \exp_not:N \group_begin:
3657         % It's two '@s', but escaped for DocStrip.
3658         \exp_not:N \hyper@@link
3659         {
3660           \__zrefclever_extract_url_unexp:V
3661           \l__zrefclever_type_first_label_tl
3662         }
3663         {
3664           \__zrefclever_extract_unexp:Vnn
3665           \l__zrefclever_type_first_label_tl { anchor } { }
3666         }
3667         {
3668           \exp_not:N \group_begin:
3669           \exp_not:V \l__zrefclever_namefont_tl
3670           \exp_not:V \l__zrefclever_type_name_tl
3671           \exp_not:N \group_end:
3672           \exp_not:V \l__zrefclever_namesep_tl
3673           \exp_not:N \group_begin:
3674           \exp_not:V \l__zrefclever_reffont_tl
3675           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
3676           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
3677           \__zrefclever_extract_unexp:Vvn
3678           \l__zrefclever_type_first_label_tl
3679           { \l__zrefclever_ref_property_tl } { }
3680           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
3681           \exp_not:N \group_end:
3682         }
3683         \exp_not:V \l__zrefclever_reffont_tl
3684         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
3685         \exp_not:N \group_end:
3686       }
3687       {
3688         \exp_not:N \group_begin:
3689         \exp_not:V \l__zrefclever_namefont_tl
3690         \exp_not:V \l__zrefclever_type_name_tl
3691         \exp_not:N \group_end:
3692         \exp_not:V \l__zrefclever_namesep_tl
3693         \__zrefclever_ref_default:
3694       }
3695     }
3696   }
3697   \bool_if:nTF \l__zrefclever_type_name_missing_bool
3698   {

```

```

3699     \_zrefclever_name_default:
3700     \exp_not:V \l__zrefclever_namesep_tl
3701   }
3702   {
3703     \exp_not:N \group_begin:
3704     \exp_not:V \l__zrefclever_namefont_tl
3705     \exp_not:V \l__zrefclever_type_name_tl
3706     \exp_not:N \group_end:
3707     \tl_if_empty:NF \l__zrefclever_type_name_tl
3708       { \exp_not:V \l__zrefclever_namesep_tl }
3709   }
3710 \zref@ifrefcontainsprop
3711 { \l__zrefclever_type_first_label_tl }
3712 { \l__zrefclever_ref_property_tl }
3713 {
3714   \bool_if:nTF
3715   {
3716     \l__zrefclever_hyperlink_bool &&
3717     ! \l__zrefclever_link_star_bool
3718   }
3719   {
3720     \exp_not:N \group_begin:
3721     \exp_not:V \l__zrefclever_reffont_tl
3722     \seq_item:Nn
3723       \l__zrefclever_type_first_refbounds_seq { 1 }
3724     % It's two '@s', but escaped for DocStrip.
3725     \exp_not:N \hyper@@link
3726       {
3727         \_zrefclever_extract_url_unexp:V
3728         \l__zrefclever_type_first_label_tl
3729       }
3730       {
3731         \_zrefclever_extract_unexp:Vnn
3732         \l__zrefclever_type_first_label_tl { anchor } { }
3733       }
3734       {
3735         \seq_item:Nn
3736           \l__zrefclever_type_first_refbounds_seq { 2 }
3737         \_zrefclever_extract_unexp:Vnn
3738         \l__zrefclever_type_first_label_tl
3739         { \l__zrefclever_ref_property_tl } { }
3740         \seq_item:Nn
3741           \l__zrefclever_type_first_refbounds_seq { 3 }
3742       }
3743     \seq_item:Nn
3744       \l__zrefclever_type_first_refbounds_seq { 4 }
3745     \exp_not:N \group_end:
3746   }
3747   {
3748     \exp_not:N \group_begin:
3749     \exp_not:V \l__zrefclever_reffont_tl
3750     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
3751     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
3752     \_zrefclever_extract_unexp:Vnn

```

```

3753         \l__zrefclever_type_first_label_tl
3754         { \l__zrefclever_ref_property_tl } { }
3755         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
3756         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
3757         \exp_not:N \group_end:
3758     }
3759 }
3760 { \__zrefclever_ref_default: }
3761 }
3762 }
3763 }

```

(End definition for __zrefclever_get_ref_first:.)

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

```

3764 \cs_new_protected:Npn \__zrefclever_type_name_setup:
3765 {
3766     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
3767     {
3768         \tl_clear:N \l__zrefclever_type_name_tl
3769         \bool_set_true:N \l__zrefclever_type_name_missing_bool
3770     }
3771     {
3772         \tl_if_eq:NnTF
3773         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
3774         {
3775             \tl_clear:N \l__zrefclever_type_name_tl
3776             \bool_set_true:N \l__zrefclever_type_name_missing_bool
3777         }
3778         {
3779             % Determine whether we should use capitalization, abbreviation,
3780             % and plural.
3781             \bool_lazy_or:nnTF
3782             { \l__zrefclever_cap_bool }
3783             {
3784                 \l__zrefclever_capfirst_bool &&
3785                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
3786             }
3787             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
3788             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
3789             % If the queue is empty, we have a singular, otherwise, plural.
3790             \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
3791             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }

```

```

3792     { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
3793 \bool_lazy_and:nnTF
3794 { \l__zrefclever_abbrev_bool }
3795 {
3796     ! \int_compare_p:nNn
3797       { \l__zrefclever_type_count_int } = { 0 } ||
3798     ! \l__zrefclever_noabbrev_first_bool
3799 }
3800 {
3801     \tl_set:NV \l__zrefclever_name_format_fallback_tl
3802       \l__zrefclever_name_format_tl
3803     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
3804 }
3805 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
3806
3807 % Handle number and gender nudges.
3808 \bool_if:NT \l__zrefclever_nudge_enabled_bool
3809 {
3810     \bool_if:NTF \l__zrefclever_nudge_singular_bool
3811     {
3812         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
3813         {
3814             \msg_warning:nnx { zref-clever }
3815               { nudge-plural-when-sg }
3816             { \l__zrefclever_type_first_label_type_tl }
3817         }
3818     }
3819     {
3820         \bool_lazy_all:nT
3821         {
3822             { \l__zrefclever_nudge_comptosing_bool }
3823             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
3824             {
3825                 \int_compare_p:nNn
3826                   { \l__zrefclever_label_count_int } > { 0 }
3827             }
3828         }
3829         {
3830             \msg_warning:nnx { zref-clever }
3831               { nudge-comptosing }
3832             { \l__zrefclever_type_first_label_type_tl }
3833         }
3834     }
3835 \bool_lazy_and:nnT
3836 { \l__zrefclever_nudge_gender_bool }
3837 { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
3838 {
3839     \__zrefclever_get_rf_opt_seq:nxxN { gender }
3840     { \l__zrefclever_type_first_label_type_tl }
3841     { \l__zrefclever_ref_language_tl }
3842     \l__zrefclever_type_name_gender_seq
3843 \seq_if_in:NVF
3844     \l__zrefclever_type_name_gender_seq
3845     \l__zrefclever_ref_gender_tl

```

```

3846 {
3847   \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
3848   {
3849     \msg_warning:nxxxx { zref-clever }
3850     { nudge-gender-not-declared-for-type }
3851     { \l__zrefclever_ref_gender_tl }
3852     { \l__zrefclever_type_first_label_type_tl }
3853     { \l__zrefclever_ref_language_tl }
3854   }
3855   {
3856     \msg_warning:nxxxxx { zref-clever }
3857     { nudge-gender-mismatch }
3858     { \l__zrefclever_type_first_label_type_tl }
3859     { \l__zrefclever_ref_gender_tl }
3860     {
3861       \seq_use:Nn
3862         \l__zrefclever_type_name_gender_seq { ,~ }
3863     }
3864     { \l__zrefclever_ref_language_tl }
3865   }
3866 }
3867 }
3868 }
3869
3870 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
3871 {
3872   \__zrefclever_opt_tl_get:cNF
3873   {
3874     \__zrefclever_opt_varname_type:een
3875     { \l__zrefclever_type_first_label_type_tl }
3876     { \l__zrefclever_name_format_tl }
3877     { tl }
3878   }
3879   \l__zrefclever_type_name_tl
3880   {
3881     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
3882     {
3883       \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
3884       \tl_put_left:NV \l__zrefclever_name_format_tl
3885         \l__zrefclever_ref_decl_case_tl
3886     }
3887     \__zrefclever_opt_tl_get:cNF
3888     {
3889       \__zrefclever_opt_varname_lang_type:eeen
3890       { \l__zrefclever_ref_language_tl }
3891       { \l__zrefclever_type_first_label_type_tl }
3892       { \l__zrefclever_name_format_tl }
3893       { tl }
3894     }
3895     \l__zrefclever_type_name_tl
3896     {
3897       \tl_clear:N \l__zrefclever_type_name_tl
3898       \bool_set_true:N \l__zrefclever_type_name_missing_bool
3899       \msg_warning:nxxx { zref-clever } { missing-name }

```



```

3900         { \l__zrefclever_name_format_tl }
3901         { \l__zrefclever_type_first_label_type_tl }
3902     }
3903 }
3904 }
3905 {
3906     \__zrefclever_opt_tl_get:cNF
3907     {
3908         \__zrefclever_opt_varname_type:een
3909         { \l__zrefclever_type_first_label_type_tl }
3910         { \l__zrefclever_name_format_tl }
3911         { tl }
3912     }
3913     \l__zrefclever_type_name_tl
3914     {
3915         \__zrefclever_opt_tl_get:cNF
3916         {
3917             \__zrefclever_opt_varname_type:een
3918             { \l__zrefclever_type_first_label_type_tl }
3919             { \l__zrefclever_name_format_fallback_tl }
3920             { tl }
3921         }
3922         \l__zrefclever_type_name_tl
3923         {
3924             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
3925             {
3926                 \tl_put_left:Nn
3927                 \l__zrefclever_name_format_tl { - }
3928                 \tl_put_left:NV \l__zrefclever_name_format_tl
3929                 \l__zrefclever_ref_decl_case_tl
3930                 \tl_put_left:Nn
3931                 \l__zrefclever_name_format_fallback_tl { - }
3932                 \tl_put_left:NV
3933                 \l__zrefclever_name_format_fallback_tl
3934                 \l__zrefclever_ref_decl_case_tl
3935             }
3936             \__zrefclever_opt_tl_get:cNF
3937             {
3938                 \__zrefclever_opt_varname_lang_type:eeen
3939                 { \l__zrefclever_ref_language_tl }
3940                 { \l__zrefclever_type_first_label_type_tl }
3941                 { \l__zrefclever_name_format_tl }
3942                 { tl }
3943             }
3944             \l__zrefclever_type_name_tl
3945             {
3946                 \__zrefclever_opt_tl_get:cNF
3947                 {
3948                     \__zrefclever_opt_varname_lang_type:eeen
3949                     { \l__zrefclever_ref_language_tl }
3950                     { \l__zrefclever_type_first_label_type_tl }
3951                     { \l__zrefclever_name_format_fallback_tl }
3952                     { tl }
3953                 }

```

```

3954         \l__zrefclever_type_name_tl
3955     {
3956         \tl_clear:N \l__zrefclever_type_name_tl
3957         \bool_set_true:N
3958             \l__zrefclever_type_name_missing_bool
3959         \msg_warning:nnxx { zref-clever }
3960             { missing-name }
3961             { \l__zrefclever_name_format_tl }
3962             { \l__zrefclever_type_first_label_type_tl }
3963     }
3964 }
3965 }
3966 }
3967 }
3968 }
3969 }
3970
3971 % Signal whether the type name is to be included in the hyperlink or not.
3972 \bool_lazy_any:nTF
3973 {
3974     { ! \l__zrefclever_hyperlink_bool }
3975     { \l__zrefclever_link_star_bool }
3976     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
3977     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
3978 }
3979 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
3980 {
3981     \bool_lazy_any:nTF
3982     {
3983         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
3984         {
3985             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
3986             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
3987         }
3988         {
3989             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
3990             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
3991             \l__zrefclever_typeset_last_bool &&
3992             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
3993         }
3994     }
3995     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
3996     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
3997 }
3998 }

```

(End definition for `__zrefclever_type_name_setup:.`)

`__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`.

```

3999 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
4000 {

```

```

4001 \zref@ifpropundefined { urluse }
4002 { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
4003 {
4004   \zref@ifrefcontainsprop {#1} { urluse }
4005   { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
4006   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
4007 }
4008 }
4009 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for __zrefclever_extract_url_unexp:n.)

__zrefclever_labels_in_sequence:nn Auxiliary function to __zrefclever_typeset_refs_not_last_of_type:. Sets \l__zrefclever_next_maybe_range_bool to true if $\langle label\ b \rangle$ 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>}
4010 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
4011 {
4012   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
4013   {#1} { externaldocument } { }
4014   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
4015   {#2} { externaldocument } { }
4016
4017   \tl_if_eq:NNT
4018   \l__zrefclever_label_extdoc_a_tl
4019   \l__zrefclever_label_extdoc_b_tl
4020   {
4021     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
4022     {
4023       \exp_args:Nxx \tl_if_eq:nnT
4024       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
4025       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
4026       {
4027         \int_compare:nNnTF
4028         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
4029         =
4030         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
4031         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
4032         {
4033           \int_compare:nNnT
4034           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
4035           =
4036           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
4037           {
4038             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
4039             \bool_set_true:N \l__zrefclever_next_is_same_bool
4040           }
4041         }
4042       }
4043     }
4044   }

```

```

4043     }
4044     {
4045         \exp_args:Nxx \tl_if_eq:nnT
4046         { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
4047         { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
4048         {
4049             \exp_args:Nxx \tl_if_eq:nnT
4050             { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
4051             { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
4052             {
4053                 \int_compare:nNnTF
4054                 { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
4055                 =
4056                 { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
4057                 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
4058                 {
4059                     \int_compare:nNnT
4060                     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
4061                     =
4062                     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
4063                     {
4064                         \bool_set_true:N
4065                         \l__zrefclever_next_maybe_range_bool
4066                         \exp_args:Nxx \tl_if_eq:nnT
4067                         {
4068                             \__zrefclever_extract_unexp:nvn {#1}
4069                             { l__zrefclever_ref_property_tl } { }
4070                         }
4071                         {
4072                             \__zrefclever_extract_unexp:nvn {#2}
4073                             { l__zrefclever_ref_property_tl } { }
4074                         }
4075                         {
4076                             \bool_set_true:N
4077                             \l__zrefclever_next_is_same_bool
4078                         }
4079                     }
4080                 }
4081             }
4082         }
4083     }
4084 }
4085 }

```

(End definition for __zrefclever_labels_in_sequence:nn.)

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

```

\__zrefclever_get_rf_opt_tl:nnnN \__zrefclever_get_rf_opt_tl:nnnN {<option>}
    {<ref type>} {<language>} {<tl variable>}
4086 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
4087 {

```

```

4088 % First attempt: general options.
4089 \__zrefclever_opt_tl_get:cNF
4090 { \__zrefclever_opt_varname_general:nn {#1} { tl } }
4091 #4
4092 {
4093 % If not found, try type specific options.
4094 \__zrefclever_opt_tl_get:cNF
4095 { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
4096 #4
4097 {
4098 % If not found, try type- and language-specific.
4099 \__zrefclever_opt_tl_get:cNF
4100 { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
4101 #4
4102 {
4103 % If not found, try language-specific default.
4104 \__zrefclever_opt_tl_get:cNF
4105 { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
4106 #4
4107 {
4108 % If not found, try fallback.
4109 \__zrefclever_opt_tl_get:cNF
4110 { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
4111 #4
4112 { \tl_clear:N #4 }
4113 }
4114 }
4115 }
4116 }
4117 }
4118 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { nxxN }

(End definition for \__zrefclever_get_rf_opt_tl:nnnN.)

```

```

\__zrefclever_get_rf_opt_seq:nnnN \__zrefclever_get_rf_opt_seq:nnnN {<option>}
{\<ref type>}{\<language>}{\<seq variable>}
4119 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
4120 {
4121 % First attempt: general options.
4122 \__zrefclever_opt_seq_get:cNF
4123 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
4124 #4
4125 {
4126 % If not found, try type specific options.
4127 \__zrefclever_opt_seq_get:cNF
4128 { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
4129 #4
4130 {
4131 % If not found, try type- and language-specific.
4132 \__zrefclever_opt_seq_get:cNF
4133 { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
4134 #4
4135 {
4136 % If not found, try language-specific default.

```

```

4137         \__zrefclever_opt_seq_get:cNF
4138         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
4139         #4
4140         {
4141             % If not found, try fallback.
4142             \__zrefclever_opt_seq_get:cNF
4143             { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
4144             #4
4145             { \seq_clear:N #4 }
4146         }
4147     }
4148 }
4149 }
4150 }
4151 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { nxxN }

```

(End definition for __zrefclever_get_rf_opt_seq:nnnN.)

```

\__zrefclever_get_rf_opt_bool:nnnnN
\__zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
{<ref type>} {<language>} {<bool variable>}
4152 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
4153 {
4154     % First attempt: general options.
4155     \__zrefclever_opt_bool_get:cNF
4156     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
4157     #5
4158     {
4159         % If not found, try type specific options.
4160         \__zrefclever_opt_bool_get:cNF
4161         { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
4162         #5
4163         {
4164             % If not found, try type- and language-specific.
4165             \__zrefclever_opt_bool_get:cNF
4166             { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
4167             #5
4168             {
4169                 % If not found, try language-specific default.
4170                 \__zrefclever_opt_bool_get:cNF
4171                 { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
4172                 #5
4173                 {
4174                     % If not found, try fallback.
4175                     \__zrefclever_opt_bool_get:cNF
4176                     { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
4177                     #5
4178                     { \use:c { bool_set_ #2 :N } #5 }
4179                 }
4180             }
4181         }
4182     }
4183 }
4184 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nnxxN }

```

(End 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.

```
4185 \__zrefclever_compat_module:nn { appendix }
4186 {
4187   \AddToHook { cmd / appendix / before }
4188   {
4189     \__zrefclever_zcsetup:n
4190     {
4191       countertype =
4192       {
4193         chapter      = appendix ,
4194         section      = appendix ,
4195         subsection   = appendix ,
4196         subsubsection = appendix ,
4197         paragraph    = appendix ,
4198         subparagraph = appendix ,
4199       }
4200     }
4201   }
4202 }
```

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.

```

4203 \__zrefclever_compat_module:n { appendices }
4204 {
4205   \__zrefclever_if_package_loaded:nT { appendix }
4206   {
4207     \newcounter { zc@appendix }
4208     \newcounter { zc@save@appendix }
4209     \setcounter { zc@appendix } { 0 }
4210     \setcounter { zc@save@appendix } { 0 }
4211     \cs_if_exist:cTF { chapter }
4212     {
4213       \__zrefclever_zcsetup:n
4214       { counterresetby = { chapter = zc@appendix } }
4215     }
4216     {
4217       \cs_if_exist:cT { section }
4218       {
4219         \__zrefclever_zcsetup:n
4220         { counterresetby = { section = zc@appendix } }
4221       }
4222     }
4223   \AddToHook { env / appendices / begin }
4224   {
4225     \stepcounter { zc@save@appendix }
4226     \setcounter { zc@appendix } { \value { zc@save@appendix } }
4227     \__zrefclever_zcsetup:n
4228     {
4229       countertype =
4230       {
4231         chapter      = appendix ,
4232         section      = appendix ,
4233         subsection   = appendix ,
4234         subsubsection = appendix ,
4235         paragraph    = appendix ,
4236         subparagraph = appendix ,
4237       }
4238     }
4239   }
4240   \AddToHook { env / appendices / end }
4241   { \setcounter { zc@appendix } { 0 } }
4242   \AddToHook { cmd / appendix / before }
4243   {
4244     \stepcounter { zc@save@appendix }
4245     \setcounter { zc@appendix } { \value { zc@save@appendix } }
4246   }
4247   \AddToHook { env / subappendices / begin }
4248   {
4249     \__zrefclever_zcsetup:n
4250     {
4251       countertype =
4252       {

```



```

4253             section      = appendix ,
4254             subsection   = appendix ,
4255             subsubsection = appendix ,
4256             paragraph    = appendix ,
4257             subparagraph  = appendix ,
4258         } ,
4259     }
4260 }
4261 \msg_info:nnn { zref-clever } { compat-package } { appendix }
4262 }
4263 }

```

9.3 memoir

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

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

- (a) The `sidecaption` and `sidecontcaption` environments. These environments *store* the label in an internal macro, `\m@mscaplabel`, at the begin environment code (more precisely in `\@@sidecaption`), but both the call to `\refstepcounter` and the expansion of `\m@mscaplabel` take place at `\endsidecaption`. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the $\langle label \rangle$ argument to start with. I can see two ways to deal with these environments, none of which I really like. First, map through `\m@mscaplabel` until `\label` is found, then grab the next token which is the $\langle label \rangle$. This can be used to set a `\zlabel` either with a kernel environment hook, or with `\@mem@scap@afterhook` (the former requires running `\refstepcounter` on our own, since the `env/.../end` hook comes before this is done by `\endsidecaption`). Second, locally redefine `\label` to set both labels inside the environments.
- (b) The bilingual caption commands: `\bitwonumcaption`, `\bionenumcaption`, and `\bicaption`. These commands do not support setting the label in their arguments (the labels do get set, but they end up included in the `title` property of the label too). So we do the same for them as for `sidecaption`, just taking care of grouping, since we can't count on the convenience of the environment hook (luckily for us, they are scoped themselves, so we can add an extra group there).

2. The `\subcaptionref` command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside the $\langle subtitle \rangle$ argument of the subcaptioning commands, namely: `\subcaption`, `\contsubcaption`, `\subbottom`, `\contsubbottom`, `\subtop`. This functionality is implemented by `memoir` by setting a *second label* with prefix `sub@ $\langle label \rangle$` , and storing there just that part of interest. With `zref` this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find

a place to hook into to add the property to the `main` list, since `memoir` does not really consider the possibility of some other command setting labels. `\@memsubcaption` is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.

3. `memoir`'s `\footnote`, `\verbfootnote`, `\sidefootnote` and `\pagenote`, just as the regular `\footnote` until recently in the kernel, do not set `\@currentcounter` alongside `\@currentlabel`, proper referencing to them requires setting the type for it.
4. Note that `memoir`'s appendix features “emulates” the `appendix` package, hence the corresponding compatibility module is loaded for `memoir` even if that package is not itself loaded. The same is true for the `\appendix` command module, since it is also defined.

```

4264 \__zrefclever_compat_module:nn { memoir }
4265 {
4266   \__zrefclever_if_class_loaded:nT { memoir }
4267   {

```

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`.

```

4268   \__zrefclever_zcsetup:n
4269   {
4270     countertype =
4271     {
4272       subfigure = figure ,
4273       subtable  = table  ,
4274       poemline  = line   ,
4275     } ,
4276     counterresetby =
4277     {
4278       subfigure = figure ,
4279       subtable  = table  ,
4280     } ,
4281   }

```

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

```

4282   \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
4283   {
4284     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
4285     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
4286     {
4287       \__zrefclever_memoir_orig_label:n {##1}
4288       \zlabel{##1}
4289     }
4290     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
4291   }
4292   \AddToHook { env / sidecaption / begin }
4293   { \__zrefclever_memoir_both_labels: }
4294   \AddToHook { env / sidecontcaption / begin }
4295   { \__zrefclever_memoir_both_labels: }

```

```

4296 \AddToHook{ cmd / bitwonumcaption / before }
4297 { \group_begin: \__zrefclever_memoir_both_labels: }
4298 \AddToHook{ cmd / bitwonumcaption / after }
4299 { \group_end: }
4300 \AddToHook{ cmd / bionenumcaption / before }
4301 { \group_begin: \__zrefclever_memoir_both_labels: }
4302 \AddToHook{ cmd / bionenumcaption / after }
4303 { \group_end: }
4304 \AddToHook{ cmd / bicaption / before }
4305 { \group_begin: \__zrefclever_memoir_both_labels: }
4306 \AddToHook{ cmd / bicaption / after }
4307 { \group_end: }

```

Support for subcaption reference.

```

4308 \zref@newprop { subcaption }
4309 { \cs_if_exist_use:c { @@thesub \@captype } }
4310 \AddToHook{ cmd / @memsubcaption / before }
4311 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

4312 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
4313 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
4314 \AddToHook { env / minipage / begin }
4315 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
4316 \AddToHook { cmd / @makefntext / before }
4317 {
4318   \__zrefclever_zcsetup:x
4319   { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
4320 }
4321 \AddToHook { cmd / @makesidefntext / before }
4322 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
4323 \__zrefclever_zcsetup:n
4324 {
4325   countertype =
4326   {
4327     sidefootnote = footnote ,
4328     pagenote = endnote ,
4329   } ,
4330 }
4331 \AddToHook { file / \jobname.ent / before }
4332 { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
4333 \msg_info:nnn { zref-clever } { compat-class } { memoir }
4334 }
4335 }

```

9.4 KOMA

Support for KOMA-Script document classes.

```

4336 \__zrefclever_compat_module:nn { KOMA }
4337 {
4338   \cs_if_exist:NT \KOMAClassName
4339   {

```

Add support for `captionbeside` and `captionofbeside` environments. These environments *do* run some variation of `\caption` and hence `\refstepcounter`. However, this happens inside a parbox inside the environment, thus grouped, such that we cannot see the variables set by `\refstepcounter` when we are setting the label. `\@currentlabel` is smuggled out of the group by KOMA, but the same care is not granted for `\@currentcounter`. So we have to rely on `\@capttype`, which the underlying caption infrastructure feeds to `\refstepcounter`. Since we must use `env/.../after` hooks, care should be taken not to set the `currentcounter` option unscoped, which would be quite disastrous. For this reason, though more “invasive”, we set `\@currentcounter` instead, which at least will be set straight the next time `\refstepcounter` runs. It sounds reasonable, it is the same treatment `\@currentlabel` is receiving in this case.

```

4340     \AddToHook { env / captionbeside / after }
4341     {
4342         \tl_if_exist:NT \@capttype
4343         { \tl_set_eq:NN \@currentcounter \@capttype }
4344     }
4345     \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
4346     \AddToHook { env / captionofbeside / end }
4347     { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
4348     \AddToHook { env / captionofbeside / after }
4349     {
4350         \tl_if_eq:NnF \@currenvir { document }
4351         {
4352             \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
4353             {
4354                 \tl_set_eq:NN
4355                 \@currentcounter \g__zrefclever_koma_capttype_tl
4356             }
4357         }
4358         \tl_gclear:N \g__zrefclever_koma_capttype_tl
4359     }
4360     \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
4361 }
4362 }
```

9.5 amsmath

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

```

4363 \__zrefclever_compat_module:nn { amsmath }
4364 {
4365     \__zrefclever_if_package_loaded:nT { amsmath }
4366     {
```

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

```

4367         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
4368         {
```

```

4369         \_zrefclever_orig_ltxlabel:n {#1}
4370         \zref@wrapper@babel \zref@label {#1}
4371     }

```

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside `amsmath`'s math environments. And, after that, redefine it to be `_zrefclever_ltxlabel:n` instead. We must handle `hyperref` here, which comes very late in the preamble, and which loads `nameref` at `begindocument`, which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. Other classes packages also redefine `\ltx@label`, which may cause some trouble. A `grep` on `texmf-dist` returns hits for: `thm-restate.sty`, `smartref.sty`, `jmlrbook.cls`, `cleveref.sty`, `cryptocode.sty`, `nameref.sty`, `easyeqn.sty`, `empheq.sty`, `ntheorem.sty`, `nccmath.sty`, `nwejm.cls`, `nwejmart.cls`, `aguplus.sty`, `aguplus.cls`, `agupp.sty`, `amsmath.hyp`, `amsmath.sty` (surprise!), `amsmath.4ht`, `nameref.4ht`, `frenchle.sty`, `french.sty`, plus corresponding documentations and different versions of the same packages. That's not too many, but not "just a few" either. The critical ones are explicitly handled here (`amsmath` itself, and `nameref`). A number of those I'm really not acquainted with. For `cleveref`, in particular, this procedure is not compatible with it. If we happen to come later than it and override its definition, this may be a substantial problem for `cleveref`, since it will find the label, but it won't contain the data it is expecting. However, this should normally not occur, if the user has followed the documented recommendation for `cleveref` to load it last, or at least very late, and besides I don't see much of an use case for using both `cleveref` and `zref-clever` together. I have documented in the user manual that this module may cause potential issues, and how to work around them. And I have made an upstream feature request for a hook, so that this could be made more cleanly at <https://github.com/latex3/hyperref/issues/212>.

```

4372     \_zrefclever_if_package_loaded:nTF { hyperref }
4373     {
4374         \AddToHook { package / nameref / after }
4375         {
4376             \cs_new_eq:NN \_zrefclever_orig_ltxlabel:n \ltx@label
4377             \cs_set_eq:NN \ltx@label \_zrefclever_ltxlabel:n
4378         }
4379     }
4380     {
4381         \cs_new_eq:NN \_zrefclever_orig_ltxlabel:n \ltx@label
4382         \cs_set_eq:NN \ltx@label \_zrefclever_ltxlabel:n
4383     }

```

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).

```

4384     \AddToHook { env / subequations / begin }

```

```

4385     {
4386       \__zrefclever_zcsetup:x
4387       {
4388         counterresetby =
4389         {
4390           parentequation =
4391             \__zrefclever_counter_reset_by:n { equation } ,
4392           equation = parentequation ,
4393         } ,
4394         currentcounter = parentequation ,
4395         countertype = { parentequation = equation } ,
4396       }
4397     }

```

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.

```

4398     \clist_map_inline:nn
4399     {
4400       equation ,
4401       equation* ,
4402       align ,
4403       align* ,
4404       alignat ,
4405       alignat* ,
4406       flalign ,
4407       flalign* ,
4408       xalignat ,
4409       xalignat* ,
4410       gather ,
4411       gather* ,
4412       multiline ,
4413       multiline* ,
4414     }
4415     {
4416       \AddToHook { env / #1 / begin }
4417       { \__zrefclever_zcsetup:n { currentcounter = equation } }
4418     }

```

And a last touch of care for amsmath’s refinements: make the equation references `\textup`.

```

4419     \zcRefTypeSetup { equation }
4420     { reffont = \upshape }
4421     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
4422   }
4423 }

```

9.6 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.

```

4424 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
4425 \__zrefclever_compat_module:nn { mathtools }
4426 {
4427   \__zrefclever_if_package_loaded:nT { mathtools }
4428   {
4429     \MH_if_boolean:nT { show_only_refs }
4430     {
4431       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
4432       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
4433       {
4434         \@bsphack
4435         \seq_map_inline:Nn #1
4436         {
4437           \exp_args:Nx \tl_if_eq:nnTF
4438           { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
4439           { equation }
4440           {
4441             \protected@write \@auxout { }
4442             { \string \MT@newlabel {##1} }
4443           }
4444           {
4445             \exp_args:Nx \tl_if_eq:nnT
4446             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
4447             { parentequation }
4448             {
4449               \protected@write \@auxout { }
4450               { \string \MT@newlabel {##1} }
4451             }
4452           }
4453         }
4454       }
4455     }
4456     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
4457   }
4458 }
4459 }
```

9.7 breqn

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for `\zlabel` just as well. However, if it happens not to work, there was no

easy alternative handle I could find. In particular, it does not seem viable to leverage the `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

```

4460 \__zrefclever_compat_module:nn { breqn }
4461 {
4462   \__zrefclever_if_package_loaded:nT { breqn }
4463   {

```

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>).

```

4464   \AddToHook { env / dgroup / begin }
4465   {
4466     \__zrefclever_zcsetup:x
4467     {
4468       counterresetby =
4469       {
4470         parentequation =
4471         \__zrefclever_counter_reset_by:n { equation } ,
4472         equation = parentequation ,
4473       } ,
4474       currentcounter = parentequation ,
4475       countertype = { parentequation = equation } ,
4476     }
4477   }
4478   \clist_map_inline:nn
4479   {
4480     dmath ,
4481     dseries ,
4482     darray ,
4483   }
4484   {
4485     \AddToHook { env / #1 / begin }
4486     { \__zrefclever_zcsetup:n { currentcounter = equation } }
4487   }
4488   \msg_info:nnn { zref-clever } { compat-package } { breqn }
4489 }
4490 }

```

9.8 listings

```

4491 \__zrefclever_compat_module:nn { listings }
4492 {
4493   \__zrefclever_if_package_loaded:nT { listings }
4494   {
4495     \__zrefclever_zcsetup:n
4496     {
4497       countertype =
4498       {
4499         lstlisting = listing ,

```



```

4500         lstnumber = line ,
4501     } ,
4502     counterresetby = { lstnumber = lstlisting } ,
4503 }

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment. The *only* place to set this label is the `PreInit` hook. This hook, comes right after `\lst@MakeCaption` in `\lst@Init`, which runs `\refstepcounter` on `lstlisting`, so we must come after it. Also `listings` itself sets `\@currentlabel` to `\thelstnumber` in the `Init` hook, which comes right after the `PreInit` one in `\lst@Init`. Since, if we add to `Init` here, we go to the end of it, we'd be seeing the wrong `\@currentlabel` at that point.

```

4504     \lst@AddToHook { PreInit }
4505     { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }

```

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

```

4506     \lst@AddToHook { Init }
4507     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
4508     \msg_info:nnn { zref-clever } { compat-package } { listings }
4509 }
4510 }

```

9.9 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.

```

4511 \__zrefclever_compat_module:nn { enumitem }
4512 {
4513     \__zrefclever_if_package_loaded:nT { enumitem }
4514     {
4515         \int_set:Nn \l_tmpa_int { 5 }
4516         \bool_while_do:nn
4517         {
4518             \cs_if_exist_p:c
4519             { c@ enum \int_to_roman:n { \l_tmpa_int } }

```

```

4520     }
4521     {
4522         \_zrefclever_zcsetup:x
4523         {
4524             counterresetby =
4525             {
4526                 enum \int_to_roman:n { \l_tmpa_int } =
4527                 enum \int_to_roman:n { \l_tmpa_int - 1 }
4528             } ,
4529             countertype =
4530             { enum \int_to_roman:n { \l_tmpa_int } = item } ,
4531         }
4532         \int_incr:N \l_tmpa_int
4533     }
4534     \int_compare:nNnT { \l_tmpa_int } > { 5 }
4535     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
4536 }
4537 }

```

9.10 subcaption

```

4538 \_zrefclever_compat_module:nn { subcaption }
4539 {
4540     \_zrefclever_if_package_loaded:nT { subcaption }
4541     {
4542         \_zrefclever_zcsetup:n
4543         {
4544             countertype =
4545             {
4546                 subfigure = figure ,
4547                 subtable = table ,
4548             } ,
4549             counterresetby =
4550             {
4551                 subfigure = figure ,
4552                 subtable = table ,
4553             } ,
4554         }
4555     }
4556 }

```

Support for `subref` reference.

```

4555 \zref@newprop { subref }
4556 { \cs_if_exist_use:c { thesub \@captype } }
4557 \tl_put_right:Nn \caption@subtypehook
4558 { \zref@localaddprop \ZREF@mainlist { subref } }
4559 }
4560 }

```

9.11 subfig

Though `subfig` offers `\subref` (as `subcaption`), I could not find any reasonable place to add the `subref` property to `zref`'s main list.

```

4561 \_zrefclever_compat_module:nn { subfig }
4562 {
4563     \_zrefclever_if_package_loaded:nT { subfig }
4564     {

```

```

4565 \__zrefclever_zcsetup:n
4566 {
4567     countertype =
4568     {
4569         subfigure = figure ,
4570         subtable = table ,
4571     } ,
4572     counterresetby =
4573     {
4574         subfigure = figure ,
4575         subtable = table ,
4576     } ,
4577 }
4578 }
4579 }
4580 </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 English

English language file has been initially provided by the author.

```

4581 <*package>
4582 \zcDeclareLanguage { english }
4583 \zcDeclareLanguageAlias { american } { english }
4584 \zcDeclareLanguageAlias { australian } { english }
4585 \zcDeclareLanguageAlias { british } { english }
4586 \zcDeclareLanguageAlias { canadian } { english }
4587 \zcDeclareLanguageAlias { newzealand } { english }
4588 \zcDeclareLanguageAlias { UKenglish } { english }
4589 \zcDeclareLanguageAlias { USenglish } { english }
4590 </package>
4591 <*lang-english>
4592 namesep = {\nobreakspace} ,
4593 pairsep = {\~and\nobreakspace} ,
4594 listsep = {\~,~} ,
4595 lastsep = {\~and\nobreakspace} ,
4596 tpairsep = {\~and\nobreakspace} ,
4597 tlistsep = {\~,~} ,
4598 tlastsep = {\~,~and\nobreakspace} ,
4599 notesep = {\~} ,
4600 rangesep = {\~to\nobreakspace} ,
4601
4602 type = book ,
4603 Name-sg = Book ,
4604 name-sg = book ,

```

```

4605     Name-pl = Books ,
4606     name-pl = books ,
4607
4608     type = part ,
4609     Name-sg = Part ,
4610     name-sg = part ,
4611     Name-pl = Parts ,
4612     name-pl = parts ,
4613
4614     type = chapter ,
4615     Name-sg = Chapter ,
4616     name-sg = chapter ,
4617     Name-pl = Chapters ,
4618     name-pl = chapters ,
4619
4620     type = section ,
4621     Name-sg = Section ,
4622     name-sg = section ,
4623     Name-pl = Sections ,
4624     name-pl = sections ,
4625
4626     type = paragraph ,
4627     Name-sg = Paragraph ,
4628     name-sg = paragraph ,
4629     Name-pl = Paragraphs ,
4630     name-pl = paragraphs ,
4631     Name-sg-ab = Par. ,
4632     name-sg-ab = par. ,
4633     Name-pl-ab = Par. ,
4634     name-pl-ab = par. ,
4635
4636     type = appendix ,
4637     Name-sg = Appendix ,
4638     name-sg = appendix ,
4639     Name-pl = Appendices ,
4640     name-pl = appendices ,
4641
4642     type = page ,
4643     Name-sg = Page ,
4644     name-sg = page ,
4645     Name-pl = Pages ,
4646     name-pl = pages ,
4647     rangesep = {\textendash} ,
4648
4649     type = line ,
4650     Name-sg = Line ,
4651     name-sg = line ,
4652     Name-pl = Lines ,
4653     name-pl = lines ,
4654
4655     type = figure ,
4656     Name-sg = Figure ,
4657     name-sg = figure ,
4658     Name-pl = Figures ,

```

```

4659 name-pl = figures ,
4660 Name-sg-ab = Fig. ,
4661 name-sg-ab = fig. ,
4662 Name-pl-ab = Figs. ,
4663 name-pl-ab = figs. ,
4664
4665 type = table ,
4666 Name-sg = Table ,
4667 name-sg = table ,
4668 Name-pl = Tables ,
4669 name-pl = tables ,
4670
4671 type = item ,
4672 Name-sg = Item ,
4673 name-sg = item ,
4674 Name-pl = Items ,
4675 name-pl = items ,
4676
4677 type = footnote ,
4678 Name-sg = Footnote ,
4679 name-sg = footnote ,
4680 Name-pl = Footnotes ,
4681 name-pl = footnotes ,
4682
4683 type = endnote ,
4684 Name-sg = Note ,
4685 name-sg = note ,
4686 Name-pl = Notes ,
4687 name-pl = notes ,
4688
4689 type = note ,
4690 Name-sg = Note ,
4691 name-sg = note ,
4692 Name-pl = Notes ,
4693 name-pl = notes ,
4694
4695 type = equation ,
4696 Name-sg = Equation ,
4697 name-sg = equation ,
4698 Name-pl = Equations ,
4699 name-pl = equations ,
4700 Name-sg-ab = Eq. ,
4701 name-sg-ab = eq. ,
4702 Name-pl-ab = Eqs. ,
4703 name-pl-ab = eqs. ,
4704 rebounds-first-sg = {,(,)}, ,
4705 rebounds = {(,,)} ,
4706
4707 type = theorem ,
4708 Name-sg = Theorem ,
4709 name-sg = theorem ,
4710 Name-pl = Theorems ,
4711 name-pl = theorems ,
4712

```

```

4713 type = lemma ,
4714     Name-sg = Lemma ,
4715     name-sg = lemma ,
4716     Name-pl = Lemmas ,
4717     name-pl = lemmas ,
4718
4719 type = corollary ,
4720     Name-sg = Corollary ,
4721     name-sg = corollary ,
4722     Name-pl = Corollaries ,
4723     name-pl = corollaries ,
4724
4725 type = proposition ,
4726     Name-sg = Proposition ,
4727     name-sg = proposition ,
4728     Name-pl = Propositions ,
4729     name-pl = propositions ,
4730
4731 type = definition ,
4732     Name-sg = Definition ,
4733     name-sg = definition ,
4734     Name-pl = Definitions ,
4735     name-pl = definitions ,
4736
4737 type = proof ,
4738     Name-sg = Proof ,
4739     name-sg = proof ,
4740     Name-pl = Proofs ,
4741     name-pl = proofs ,
4742
4743 type = result ,
4744     Name-sg = Result ,
4745     name-sg = result ,
4746     Name-pl = Results ,
4747     name-pl = results ,
4748
4749 type = remark ,
4750     Name-sg = Remark ,
4751     name-sg = remark ,
4752     Name-pl = Remarks ,
4753     name-pl = remarks ,
4754
4755 type = example ,
4756     Name-sg = Example ,
4757     name-sg = example ,
4758     Name-pl = Examples ,
4759     name-pl = examples ,
4760
4761 type = algorithm ,
4762     Name-sg = Algorithm ,
4763     name-sg = algorithm ,
4764     Name-pl = Algorithms ,
4765     name-pl = algorithms ,
4766

```

```

4767 type = listing ,
4768   Name-sg = Listing ,
4769   name-sg = listing ,
4770   Name-pl = Listings ,
4771   name-pl = listings ,
4772
4773 type = exercise ,
4774   Name-sg = Exercise ,
4775   name-sg = exercise ,
4776   Name-pl = Exercises ,
4777   name-pl = exercises ,
4778
4779 type = solution ,
4780   Name-sg = Solution ,
4781   name-sg = solution ,
4782   Name-pl = Solutions ,
4783   name-pl = solutions ,
4784 </lang-english>

```

10.2 German

German language file has been initially provided by the author.

```

4785 <*package>
4786 \zcDeclareLanguage
4787   [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
4788   { german }
4789 \zcDeclareLanguageAlias { austrian    } { german }
4790 \zcDeclareLanguageAlias { germanb     } { german }
4791 \zcDeclareLanguageAlias { ngerman     } { german }
4792 \zcDeclareLanguageAlias { naustrian   } { german }
4793 \zcDeclareLanguageAlias { nswissgerman } { german }
4794 \zcDeclareLanguageAlias { swissgerman  } { german }
4795 </package>
4796 <*lang-german>
4797 namesep = {\nobreakspace} ,
4798 pairsep  = {\~und\nobreakspace} ,
4799 listsep  = { , ~ } ,
4800 lastsep  = {\~und\nobreakspace} ,
4801 tpairsep = {\~und\nobreakspace} ,
4802 tlistsep = { , ~ } ,
4803 tlastsep = {\~und\nobreakspace} ,
4804 notesep  = { ~ } ,
4805 rangesep = {\~bis\nobreakspace} ,
4806
4807 type = book ,
4808   gender = n ,
4809   case = N ,
4810     Name-sg = Buch ,
4811     Name-pl = Bücher ,
4812   case = A ,
4813     Name-sg = Buch ,
4814     Name-pl = Bücher ,

```

```

4815     case = D ,
4816         Name-sg = Buch ,
4817         Name-pl = Büchern ,
4818     case = G ,
4819         Name-sg = Buches ,
4820         Name-pl = Bücher ,
4821
4822 type = part ,
4823     gender = m ,
4824     case = N ,
4825         Name-sg = Teil ,
4826         Name-pl = Teile ,
4827     case = A ,
4828         Name-sg = Teil ,
4829         Name-pl = Teile ,
4830     case = D ,
4831         Name-sg = Teil ,
4832         Name-pl = Teilen ,
4833     case = G ,
4834         Name-sg = Teiles ,
4835         Name-pl = Teile ,
4836
4837 type = chapter ,
4838     gender = n ,
4839     case = N ,
4840         Name-sg = Kapitel ,
4841         Name-pl = Kapitel ,
4842     case = A ,
4843         Name-sg = Kapitel ,
4844         Name-pl = Kapitel ,
4845     case = D ,
4846         Name-sg = Kapitel ,
4847         Name-pl = Kapiteln ,
4848     case = G ,
4849         Name-sg = Kapitels ,
4850         Name-pl = Kapitel ,
4851
4852 type = section ,
4853     gender = m ,
4854     case = N ,
4855         Name-sg = Abschnitt ,
4856         Name-pl = Abschnitte ,
4857     case = A ,
4858         Name-sg = Abschnitt ,
4859         Name-pl = Abschnitte ,
4860     case = D ,
4861         Name-sg = Abschnitt ,
4862         Name-pl = Abschnitten ,
4863     case = G ,
4864         Name-sg = Abschnitts ,
4865         Name-pl = Abschnitte ,
4866
4867 type = paragraph ,
4868     gender = m ,

```



```

4869 case = N ,
4870     Name-sg = Absatz ,
4871     Name-pl = Absätze ,
4872 case = A ,
4873     Name-sg = Absatz ,
4874     Name-pl = Absätze ,
4875 case = D ,
4876     Name-sg = Absatz ,
4877     Name-pl = Absätzen ,
4878 case = G ,
4879     Name-sg = Absatzes ,
4880     Name-pl = Absätze ,
4881
4882 type = appendix ,
4883     gender = m ,
4884     case = N ,
4885         Name-sg = Anhang ,
4886         Name-pl = Anhänge ,
4887     case = A ,
4888         Name-sg = Anhang ,
4889         Name-pl = Anhänge ,
4890     case = D ,
4891         Name-sg = Anhang ,
4892         Name-pl = Anhängen ,
4893     case = G ,
4894         Name-sg = Anhangs ,
4895         Name-pl = Anhänge ,
4896
4897 type = page ,
4898     gender = f ,
4899     case = N ,
4900         Name-sg = Seite ,
4901         Name-pl = Seiten ,
4902     case = A ,
4903         Name-sg = Seite ,
4904         Name-pl = Seiten ,
4905     case = D ,
4906         Name-sg = Seite ,
4907         Name-pl = Seiten ,
4908     case = G ,
4909         Name-sg = Seite ,
4910         Name-pl = Seiten ,
4911     rangesep = {\textendash} ,
4912
4913 type = line ,
4914     gender = f ,
4915     case = N ,
4916         Name-sg = Zeile ,
4917         Name-pl = Zeilen ,
4918     case = A ,
4919         Name-sg = Zeile ,
4920         Name-pl = Zeilen ,
4921     case = D ,
4922         Name-sg = Zeile ,

```

```

4923     Name-pl = Zeilen ,
4924     case = G ,
4925     Name-sg = Zeile ,
4926     Name-pl = Zeilen ,
4927
4928 type = figure ,
4929     gender = f ,
4930     case = N ,
4931     Name-sg = Abbildung ,
4932     Name-pl = Abbildungen ,
4933     Name-sg-ab = Abb. ,
4934     Name-pl-ab = Abb. ,
4935     case = A ,
4936     Name-sg = Abbildung ,
4937     Name-pl = Abbildungen ,
4938     Name-sg-ab = Abb. ,
4939     Name-pl-ab = Abb. ,
4940     case = D ,
4941     Name-sg = Abbildung ,
4942     Name-pl = Abbildungen ,
4943     Name-sg-ab = Abb. ,
4944     Name-pl-ab = Abb. ,
4945     case = G ,
4946     Name-sg = Abbildung ,
4947     Name-pl = Abbildungen ,
4948     Name-sg-ab = Abb. ,
4949     Name-pl-ab = Abb. ,
4950
4951 type = table ,
4952     gender = f ,
4953     case = N ,
4954     Name-sg = Tabelle ,
4955     Name-pl = Tabellen ,
4956     case = A ,
4957     Name-sg = Tabelle ,
4958     Name-pl = Tabellen ,
4959     case = D ,
4960     Name-sg = Tabelle ,
4961     Name-pl = Tabellen ,
4962     case = G ,
4963     Name-sg = Tabelle ,
4964     Name-pl = Tabellen ,
4965
4966 type = item ,
4967     gender = m ,
4968     case = N ,
4969     Name-sg = Punkt ,
4970     Name-pl = Punkte ,
4971     case = A ,
4972     Name-sg = Punkt ,
4973     Name-pl = Punkte ,
4974     case = D ,
4975     Name-sg = Punkt ,
4976     Name-pl = Punkten ,

```

```

4977     case = G ,
4978         Name-sg = Punktes ,
4979         Name-pl = Punkte ,
4980
4981 type = footnote ,
4982     gender = f ,
4983     case = N ,
4984         Name-sg = Fußnote ,
4985         Name-pl = Fußnoten ,
4986     case = A ,
4987         Name-sg = Fußnote ,
4988         Name-pl = Fußnoten ,
4989     case = D ,
4990         Name-sg = Fußnote ,
4991         Name-pl = Fußnoten ,
4992     case = G ,
4993         Name-sg = Fußnote ,
4994         Name-pl = Fußnoten ,
4995
4996 type = endnote ,
4997     gender = f ,
4998     case = N ,
4999         Name-sg = Endnote ,
5000         Name-pl = Endnoten ,
5001     case = A ,
5002         Name-sg = Endnote ,
5003         Name-pl = Endnoten ,
5004     case = D ,
5005         Name-sg = Endnote ,
5006         Name-pl = Endnoten ,
5007     case = G ,
5008         Name-sg = Endnote ,
5009         Name-pl = Endnoten ,
5010
5011 type = note ,
5012     gender = f ,
5013     case = N ,
5014         Name-sg = Anmerkung ,
5015         Name-pl = Anmerkungen ,
5016     case = A ,
5017         Name-sg = Anmerkung ,
5018         Name-pl = Anmerkungen ,
5019     case = D ,
5020         Name-sg = Anmerkung ,
5021         Name-pl = Anmerkungen ,
5022     case = G ,
5023         Name-sg = Anmerkung ,
5024         Name-pl = Anmerkungen ,
5025
5026 type = equation ,
5027     gender = f ,
5028     case = N ,
5029         Name-sg = Gleichung ,
5030         Name-pl = Gleichungen ,

```

```

5031 case = A ,
5032     Name-sg = Gleichung ,
5033     Name-pl = Gleichungen ,
5034 case = D ,
5035     Name-sg = Gleichung ,
5036     Name-pl = Gleichungen ,
5037 case = G ,
5038     Name-sg = Gleichung ,
5039     Name-pl = Gleichungen ,
5040 refbounds-first-sg = {(,),(,)},
5041 refbounds = {(,),(,)} ,
5042
5043 type = theorem ,
5044     gender = n ,
5045     case = N ,
5046         Name-sg = Theorem ,
5047         Name-pl = Theoreme ,
5048     case = A ,
5049         Name-sg = Theorem ,
5050         Name-pl = Theoreme ,
5051     case = D ,
5052         Name-sg = Theorem ,
5053         Name-pl = Theoremen ,
5054     case = G ,
5055         Name-sg = Theorems ,
5056         Name-pl = Theoreme ,
5057
5058 type = lemma ,
5059     gender = n ,
5060     case = N ,
5061         Name-sg = Lemma ,
5062         Name-pl = Lemmata ,
5063     case = A ,
5064         Name-sg = Lemma ,
5065         Name-pl = Lemmata ,
5066     case = D ,
5067         Name-sg = Lemma ,
5068         Name-pl = Lemmata ,
5069     case = G ,
5070         Name-sg = Lemmas ,
5071         Name-pl = Lemmata ,
5072
5073 type = corollary ,
5074     gender = n ,
5075     case = N ,
5076         Name-sg = Korollar ,
5077         Name-pl = Korollare ,
5078     case = A ,
5079         Name-sg = Korollar ,
5080         Name-pl = Korollare ,
5081     case = D ,
5082         Name-sg = Korollar ,
5083         Name-pl = Korollaren ,
5084     case = G ,

```

```

5085     Name-sg = Korollars ,
5086     Name-pl = Korollare ,
5087
5088 type = proposition ,
5089     gender = m ,
5090     case = N ,
5091     Name-sg = Satz ,
5092     Name-pl = Sätze ,
5093     case = A ,
5094     Name-sg = Satz ,
5095     Name-pl = Sätze ,
5096     case = D ,
5097     Name-sg = Satz ,
5098     Name-pl = Sätzen ,
5099     case = G ,
5100     Name-sg = Satzes ,
5101     Name-pl = Sätze ,
5102
5103 type = definition ,
5104     gender = f ,
5105     case = N ,
5106     Name-sg = Definition ,
5107     Name-pl = Definitionen ,
5108     case = A ,
5109     Name-sg = Definition ,
5110     Name-pl = Definitionen ,
5111     case = D ,
5112     Name-sg = Definition ,
5113     Name-pl = Definitionen ,
5114     case = G ,
5115     Name-sg = Definition ,
5116     Name-pl = Definitionen ,
5117
5118 type = proof ,
5119     gender = m ,
5120     case = N ,
5121     Name-sg = Beweis ,
5122     Name-pl = Beweise ,
5123     case = A ,
5124     Name-sg = Beweis ,
5125     Name-pl = Beweise ,
5126     case = D ,
5127     Name-sg = Beweis ,
5128     Name-pl = Beweisen ,
5129     case = G ,
5130     Name-sg = Beweises ,
5131     Name-pl = Beweise ,
5132
5133 type = result ,
5134     gender = n ,
5135     case = N ,
5136     Name-sg = Ergebnis ,
5137     Name-pl = Ergebnisse ,
5138     case = A ,

```

```

5139     Name-sg = Ergebnis ,
5140     Name-pl = Ergebnisse ,
5141     case = D ,
5142     Name-sg = Ergebnis ,
5143     Name-pl = Ergebnissen ,
5144     case = G ,
5145     Name-sg = Ergebnisses ,
5146     Name-pl = Ergebnisse ,
5147
5148     type = remark ,
5149     gender = f ,
5150     case = N ,
5151     Name-sg = Bemerkung ,
5152     Name-pl = Bemerkungen ,
5153     case = A ,
5154     Name-sg = Bemerkung ,
5155     Name-pl = Bemerkungen ,
5156     case = D ,
5157     Name-sg = Bemerkung ,
5158     Name-pl = Bemerkungen ,
5159     case = G ,
5160     Name-sg = Bemerkung ,
5161     Name-pl = Bemerkungen ,
5162
5163     type = example ,
5164     gender = n ,
5165     case = N ,
5166     Name-sg = Beispiel ,
5167     Name-pl = Beispiele ,
5168     case = A ,
5169     Name-sg = Beispiel ,
5170     Name-pl = Beispiele ,
5171     case = D ,
5172     Name-sg = Beispiel ,
5173     Name-pl = Beispielen ,
5174     case = G ,
5175     Name-sg = Beispiels ,
5176     Name-pl = Beispiele ,
5177
5178     type = algorithm ,
5179     gender = m ,
5180     case = N ,
5181     Name-sg = Algorithmus ,
5182     Name-pl = Algorithmen ,
5183     case = A ,
5184     Name-sg = Algorithmus ,
5185     Name-pl = Algorithmen ,
5186     case = D ,
5187     Name-sg = Algorithmus ,
5188     Name-pl = Algorithmen ,
5189     case = G ,
5190     Name-sg = Algorithmus ,
5191     Name-pl = Algorithmen ,
5192

```

```

5193 type = listing ,
5194 gender = n ,
5195 case = N ,
5196     Name-sg = Listing ,
5197     Name-pl = Listings ,
5198 case = A ,
5199     Name-sg = Listing ,
5200     Name-pl = Listings ,
5201 case = D ,
5202     Name-sg = Listing ,
5203     Name-pl = Listings ,
5204 case = G ,
5205     Name-sg = Listings ,
5206     Name-pl = Listings ,
5207
5208 type = exercise ,
5209 gender = f ,
5210 case = N ,
5211     Name-sg = Übungsaufgabe ,
5212     Name-pl = Übungsaufgaben ,
5213 case = A ,
5214     Name-sg = Übungsaufgabe ,
5215     Name-pl = Übungsaufgaben ,
5216 case = D ,
5217     Name-sg = Übungsaufgabe ,
5218     Name-pl = Übungsaufgaben ,
5219 case = G ,
5220     Name-sg = Übungsaufgabe ,
5221     Name-pl = Übungsaufgaben ,
5222
5223 type = solution ,
5224 gender = f ,
5225 case = N ,
5226     Name-sg = Lösung ,
5227     Name-pl = Lösungen ,
5228 case = A ,
5229     Name-sg = Lösung ,
5230     Name-pl = Lösungen ,
5231 case = D ,
5232     Name-sg = Lösung ,
5233     Name-pl = Lösungen ,
5234 case = G ,
5235     Name-sg = Lösung ,
5236     Name-pl = Lösungen ,
5237 </lang-german>

```

10.3 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.

```

5238 <*package>
5239 \zcDeclareLanguage [ gender = { f , m } ] { french }
5240 \zcDeclareLanguageAlias { acadian } { french }
5241 \zcDeclareLanguageAlias { canadien } { french }
5242 \zcDeclareLanguageAlias { francais } { french }
5243 \zcDeclareLanguageAlias { frenchb } { french }
5244 </package>
5245 <*lang-french>
5246 namesep = {\nobreakspace} ,
5247 pairsep = {\et\nobreakspace} ,
5248 listsep = {,~} ,
5249 lastsep = {\et\nobreakspace} ,
5250 tpairsep = {\et\nobreakspace} ,
5251 tlistsep = {,~} ,
5252 tlastsep = {\et\nobreakspace} ,
5253 notesep = {~} ,
5254 rangesep = {\~\nobreakspace} ,
5255
5256 type = book ,
5257   gender = m ,
5258   Name-sg = Livre ,
5259   name-sg = livre ,
5260   Name-pl = Livres ,
5261   name-pl = livres ,
5262
5263 type = part ,
5264   gender = f ,
5265   Name-sg = Partie ,
5266   name-sg = partie ,
5267   Name-pl = Parties ,
5268   name-pl = parties ,
5269
5270 type = chapter ,
5271   gender = m ,
5272   Name-sg = Chapitre ,
5273   name-sg = chapitre ,
5274   Name-pl = Chapitres ,
5275   name-pl = chapitres ,
5276
5277 type = section ,
5278   gender = f ,
5279   Name-sg = Section ,
5280   name-sg = section ,
5281   Name-pl = Sections ,
5282   name-pl = sections ,
5283
5284 type = paragraph ,
5285   gender = m ,
5286   Name-sg = Paragraphe ,
5287   name-sg = paragraphe ,
5288   Name-pl = Paragraphes ,
5289   name-pl = paragraphes ,
5290

```



```

5291 type = appendix ,
5292     gender = f ,
5293     Name-sg = Annexe ,
5294     name-sg = annexe ,
5295     Name-pl = Annexes ,
5296     name-pl = annexes ,
5297
5298 type = page ,
5299     gender = f ,
5300     Name-sg = Page ,
5301     name-sg = page ,
5302     Name-pl = Pages ,
5303     name-pl = pages ,
5304     rangesep = {-} ,
5305
5306 type = line ,
5307     gender = f ,
5308     Name-sg = Ligne ,
5309     name-sg = ligne ,
5310     Name-pl = Lignes ,
5311     name-pl = lignes ,
5312
5313 type = figure ,
5314     gender = f ,
5315     Name-sg = Figure ,
5316     name-sg = figure ,
5317     Name-pl = Figures ,
5318     name-pl = figures ,
5319
5320 type = table ,
5321     gender = f ,
5322     Name-sg = Table ,
5323     name-sg = table ,
5324     Name-pl = Tables ,
5325     name-pl = tables ,
5326
5327 type = item ,
5328     gender = m ,
5329     Name-sg = Point ,
5330     name-sg = point ,
5331     Name-pl = Points ,
5332     name-pl = points ,
5333
5334 type = footnote ,
5335     gender = f ,
5336     Name-sg = Note ,
5337     name-sg = note ,
5338     Name-pl = Notes ,
5339     name-pl = notes ,
5340
5341 type = endnote ,
5342     gender = f ,
5343     Name-sg = Note ,
5344     name-sg = note ,

```

```

5345   Name-pl = Notes ,
5346   name-pl = notes ,
5347
5348   type = note ,
5349   gender = f ,
5350   Name-sg = Note ,
5351   name-sg = note ,
5352   Name-pl = Notes ,
5353   name-pl = notes ,
5354
5355   type = equation ,
5356   gender = f ,
5357   Name-sg = Équation ,
5358   name-sg = équation ,
5359   Name-pl = Équations ,
5360   name-pl = équations ,
5361   refbounds-first-sg = {,(,),} ,
5362   refbounds = {(,,)} ,
5363
5364   type = theorem ,
5365   gender = m ,
5366   Name-sg = Théorème ,
5367   name-sg = théorème ,
5368   Name-pl = Théorèmes ,
5369   name-pl = théorèmes ,
5370
5371   type = lemma ,
5372   gender = m ,
5373   Name-sg = Lemme ,
5374   name-sg = lemme ,
5375   Name-pl = Lemmes ,
5376   name-pl = lemmes ,
5377
5378   type = corollary ,
5379   gender = m ,
5380   Name-sg = Corollaire ,
5381   name-sg = corollaire ,
5382   Name-pl = Corollaires ,
5383   name-pl = corollaires ,
5384
5385   type = proposition ,
5386   gender = f ,
5387   Name-sg = Proposition ,
5388   name-sg = proposition ,
5389   Name-pl = Propositions ,
5390   name-pl = propositions ,
5391
5392   type = definition ,
5393   gender = f ,
5394   Name-sg = Définition ,
5395   name-sg = définition ,
5396   Name-pl = Définitions ,
5397   name-pl = définitions ,
5398

```

```

5399 type = proof ,
5400     gender = f ,
5401     Name-sg = Démonstration ,
5402     name-sg = démonstration ,
5403     Name-pl = Démonstrations ,
5404     name-pl = démonstrations ,
5405
5406 type = result ,
5407     gender = m ,
5408     Name-sg = Résultat ,
5409     name-sg = résultat ,
5410     Name-pl = Résultats ,
5411     name-pl = résultats ,
5412
5413 type = remark ,
5414     gender = f ,
5415     Name-sg = Remarque ,
5416     name-sg = remarque ,
5417     Name-pl = Remarques ,
5418     name-pl = remarques ,
5419
5420 type = example ,
5421     gender = m ,
5422     Name-sg = Exemple ,
5423     name-sg = exemple ,
5424     Name-pl = Exemples ,
5425     name-pl = exemples ,
5426
5427 type = algorithm ,
5428     gender = m ,
5429     Name-sg = Algorithme ,
5430     name-sg = algorithme ,
5431     Name-pl = Algorithmes ,
5432     name-pl = algorithmes ,
5433
5434 type = listing ,
5435     gender = m ,
5436     Name-sg = Listing ,
5437     name-sg = listing ,
5438     Name-pl = Listings ,
5439     name-pl = listings ,
5440
5441 type = exercise ,
5442     gender = m ,
5443     Name-sg = Exercice ,
5444     name-sg = exercice ,
5445     Name-pl = Exercices ,
5446     name-pl = exercices ,
5447
5448 type = solution ,
5449     gender = f ,
5450     Name-sg = Solution ,
5451     name-sg = solution ,
5452     Name-pl = Solutions ,

```

```

5453   name-pl = solutions ,
5454 </lang-french>

```

10.4 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.

```

5455 <*package>
5456 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
5457 \zcDeclareLanguageAlias { brazilian } { portuguese }
5458 \zcDeclareLanguageAlias { brazil   } { portuguese }
5459 \zcDeclareLanguageAlias { portuges  } { portuguese }
5460 </package>
5461 <*lang-portuguese>
5462 namesep = {\nobreakspace} ,
5463 pairsep  = {\~e\nobreakspace} ,
5464 listsep  = { ,~ } ,
5465 lastsep  = {\~e\nobreakspace} ,
5466 tpairsep = {\~e\nobreakspace} ,
5467 tlistsep = { ,~ } ,
5468 tlastsep = {\~e\nobreakspace} ,
5469 notesep  = { ~ } ,
5470 rangesep = {\~a\nobreakspace} ,
5471
5472 type = book ,
5473   gender = m ,
5474   Name-sg = Livro ,
5475   name-sg  = livro ,
5476   Name-pl  = Livros ,
5477   name-pl  = livros ,
5478
5479 type = part ,
5480   gender = f ,
5481   Name-sg = Parte ,
5482   name-sg = parte ,
5483   Name-pl = Partes ,
5484   name-pl = partes ,
5485
5486 type = chapter ,
5487   gender = m ,
5488   Name-sg = Capítulo ,
5489   name-sg = capítulo ,
5490   Name-pl = Capítulos ,
5491   name-pl = capítulos ,
5492
5493 type = section ,
5494   gender = f ,
5495   Name-sg = Seção ,
5496   name-sg = seção ,
5497   Name-pl = Seções ,
5498   name-pl = seções ,

```

```

5499
5500 type = paragraph ,
5501     gender = m ,
5502     Name-sg = Parágrafo ,
5503     name-sg = parágrafo ,
5504     Name-pl = Parágrafos ,
5505     name-pl = parágrafos ,
5506     Name-sg-ab = Par. ,
5507     name-sg-ab = par. ,
5508     Name-pl-ab = Par. ,
5509     name-pl-ab = par. ,
5510
5511 type = appendix ,
5512     gender = m ,
5513     Name-sg = Apêndice ,
5514     name-sg = apêndice ,
5515     Name-pl = Apêndices ,
5516     name-pl = apêndices ,
5517
5518 type = page ,
5519     gender = f ,
5520     Name-sg = Página ,
5521     name-sg = página ,
5522     Name-pl = Páginas ,
5523     name-pl = páginas ,
5524     rangesep = {\textendash} ,
5525
5526 type = line ,
5527     gender = f ,
5528     Name-sg = Linha ,
5529     name-sg = linha ,
5530     Name-pl = Linhas ,
5531     name-pl = linhas ,
5532
5533 type = figure ,
5534     gender = f ,
5535     Name-sg = Figura ,
5536     name-sg = figura ,
5537     Name-pl = Figuras ,
5538     name-pl = figuras ,
5539     Name-sg-ab = Fig. ,
5540     name-sg-ab = fig. ,
5541     Name-pl-ab = Figs. ,
5542     name-pl-ab = figs. ,
5543
5544 type = table ,
5545     gender = f ,
5546     Name-sg = Tabela ,
5547     name-sg = tabela ,
5548     Name-pl = Tabelas ,
5549     name-pl = tabelas ,
5550
5551 type = item ,
5552     gender = m ,

```

```

5553 Name-sg = Item ,
5554 name-sg = item ,
5555 Name-pl = Itens ,
5556 name-pl = itens ,
5557
5558 type = footnote ,
5559 gender = f ,
5560 Name-sg = Nota ,
5561 name-sg = nota ,
5562 Name-pl = Notas ,
5563 name-pl = notas ,
5564
5565 type = endnote ,
5566 gender = f ,
5567 Name-sg = Nota ,
5568 name-sg = nota ,
5569 Name-pl = Notas ,
5570 name-pl = notas ,
5571
5572 type = note ,
5573 gender = f ,
5574 Name-sg = Nota ,
5575 name-sg = nota ,
5576 Name-pl = Notas ,
5577 name-pl = notas ,
5578
5579 type = equation ,
5580 gender = f ,
5581 Name-sg = Equação ,
5582 name-sg = equação ,
5583 Name-pl = Equações ,
5584 name-pl = equações ,
5585 Name-sg-ab = Eq. ,
5586 name-sg-ab = eq. ,
5587 Name-pl-ab = Eqs. ,
5588 name-pl-ab = eqs. ,
5589 refbounds-first-sg = {(,),(,)} ,
5590 refbounds = {(,,)} ,
5591
5592 type = theorem ,
5593 gender = m ,
5594 Name-sg = Teorema ,
5595 name-sg = teorema ,
5596 Name-pl = Teoremas ,
5597 name-pl = teoremas ,
5598
5599 type = lemma ,
5600 gender = m ,
5601 Name-sg = Lema ,
5602 name-sg = lema ,
5603 Name-pl = Lemas ,
5604 name-pl = lemas ,
5605
5606 type = corollary ,

```

```

5607     gender = m ,
5608     Name-sg = Corolário ,
5609     name-sg = corolário ,
5610     Name-pl = Corolários ,
5611     name-pl = corolários ,
5612
5613 type = proposition ,
5614     gender = f ,
5615     Name-sg = Proposição ,
5616     name-sg = proposição ,
5617     Name-pl = Proposições ,
5618     name-pl = proposições ,
5619
5620 type = definition ,
5621     gender = f ,
5622     Name-sg = Definição ,
5623     name-sg = definição ,
5624     Name-pl = Definições ,
5625     name-pl = definições ,
5626
5627 type = proof ,
5628     gender = f ,
5629     Name-sg = Demonstração ,
5630     name-sg = demonstração ,
5631     Name-pl = Demonstrações ,
5632     name-pl = demonstrações ,
5633
5634 type = result ,
5635     gender = m ,
5636     Name-sg = Resultado ,
5637     name-sg = resultado ,
5638     Name-pl = Resultados ,
5639     name-pl = resultados ,
5640
5641 type = remark ,
5642     gender = f ,
5643     Name-sg = Observação ,
5644     name-sg = observação ,
5645     Name-pl = Observações ,
5646     name-pl = observações ,
5647
5648 type = example ,
5649     gender = m ,
5650     Name-sg = Exemplo ,
5651     name-sg = exemplo ,
5652     Name-pl = Exemplos ,
5653     name-pl = exemplos ,
5654
5655 type = algorithm ,
5656     gender = m ,
5657     Name-sg = Algoritmo ,
5658     name-sg = algoritmo ,
5659     Name-pl = Algoritmos ,
5660     name-pl = algoritmos ,

```

```

5661
5662 type = listing ,
5663     gender = f ,
5664     Name-sg = Listagem ,
5665     name-sg = listagem ,
5666     Name-pl = Listagens ,
5667     name-pl = listagens ,
5668
5669 type = exercise ,
5670     gender = m ,
5671     Name-sg = Exercício ,
5672     name-sg = exercício ,
5673     Name-pl = Exercícios ,
5674     name-pl = exercícios ,
5675
5676 type = solution ,
5677     gender = f ,
5678     Name-sg = Solução ,
5679     name-sg = solução ,
5680     Name-pl = Soluções ,
5681     name-pl = soluções ,
5682 </lang-portuguese>

```

10.5 Spanish

Spanish language file has been initially provided by the author.

```

5683 <*package>
5684 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
5685 </package>
5686 <*lang-spanish>
5687 namesep = {\nobreakspace} ,
5688 pairsep = {\sim\nobreakspace} ,
5689 listsep = {,~} ,
5690 lastsep = {\sim\nobreakspace} ,
5691 tpairsep = {\sim\nobreakspace} ,
5692 tlistsep = {,~} ,
5693 tlastsep = {\sim\nobreakspace} ,
5694 notesep = {\sim} ,
5695 rangesep = {\sim\nobreakspace} ,
5696
5697 type = book ,
5698     gender = m ,
5699     Name-sg = Libro ,
5700     name-sg = libro ,
5701     Name-pl = Libros ,
5702     name-pl = libros ,
5703
5704 type = part ,
5705     gender = f ,
5706     Name-sg = Parte ,
5707     name-sg = parte ,
5708     Name-pl = Partes ,

```



```

5709     name-pl = partes ,
5710
5711 type = chapter ,
5712     gender = m ,
5713     Name-sg = Capítulo ,
5714     name-sg = capítulo ,
5715     Name-pl = Capítulos ,
5716     name-pl = capítulos ,
5717
5718 type = section ,
5719     gender = f ,
5720     Name-sg = Sección ,
5721     name-sg = sección ,
5722     Name-pl = Secciones ,
5723     name-pl = secciones ,
5724
5725 type = paragraph ,
5726     gender = m ,
5727     Name-sg = Párrafo ,
5728     name-sg = párrafo ,
5729     Name-pl = Párrafos ,
5730     name-pl = párrafos ,
5731
5732 type = appendix ,
5733     gender = m ,
5734     Name-sg = Apéndice ,
5735     name-sg = apéndice ,
5736     Name-pl = Apéndices ,
5737     name-pl = apéndices ,
5738
5739 type = page ,
5740     gender = f ,
5741     Name-sg = Página ,
5742     name-sg = página ,
5743     Name-pl = Páginas ,
5744     name-pl = páginas ,
5745     rangesep = {\textendash} ,
5746
5747 type = line ,
5748     gender = f ,
5749     Name-sg = Línea ,
5750     name-sg = línea ,
5751     Name-pl = Líneas ,
5752     name-pl = líneas ,
5753
5754 type = figure ,
5755     gender = f ,
5756     Name-sg = Figura ,
5757     name-sg = figura ,
5758     Name-pl = Figuras ,
5759     name-pl = figuras ,
5760
5761 type = table ,
5762     gender = m ,

```

```

5763 Name-sg = Cuadro ,
5764 name-sg = cuadro ,
5765 Name-pl = Cuadros ,
5766 name-pl = cuadros ,
5767
5768 type = item ,
5769 gender = m ,
5770 Name-sg = Punto ,
5771 name-sg = punto ,
5772 Name-pl = Puntos ,
5773 name-pl = puntos ,
5774
5775 type = footnote ,
5776 gender = f ,
5777 Name-sg = Nota ,
5778 name-sg = nota ,
5779 Name-pl = Notas ,
5780 name-pl = notas ,
5781
5782 type = endnote ,
5783 gender = f ,
5784 Name-sg = Nota ,
5785 name-sg = nota ,
5786 Name-pl = Notas ,
5787 name-pl = notas ,
5788
5789 type = note ,
5790 gender = f ,
5791 Name-sg = Nota ,
5792 name-sg = nota ,
5793 Name-pl = Notas ,
5794 name-pl = notas ,
5795
5796 type = equation ,
5797 gender = f ,
5798 Name-sg = Ecuación ,
5799 name-sg = ecuación ,
5800 Name-pl = Ecuaciones ,
5801 name-pl = ecuaciones ,
5802 refbounds-first-sg = {(,),(,)},
5803 refbounds = {(,,)},
5804
5805 type = theorem ,
5806 gender = m ,
5807 Name-sg = Teorema ,
5808 name-sg = teorema ,
5809 Name-pl = Teoremas ,
5810 name-pl = teoremas ,
5811
5812 type = lemma ,
5813 gender = m ,
5814 Name-sg = Lema ,
5815 name-sg = lema ,
5816 Name-pl = Lemas ,

```

```

5817     name-pl = lemas ,
5818
5819 type = corollary ,
5820     gender = m ,
5821     Name-sg = Corolario ,
5822     name-sg = corolario ,
5823     Name-pl = Corolarios ,
5824     name-pl = corolarios ,
5825
5826 type = proposition ,
5827     gender = f ,
5828     Name-sg = Proposición ,
5829     name-sg = proposición ,
5830     Name-pl = Proposiciones ,
5831     name-pl = proposiciones ,
5832
5833 type = definition ,
5834     gender = f ,
5835     Name-sg = Definición ,
5836     name-sg = definición ,
5837     Name-pl = Definiciones ,
5838     name-pl = definiciones ,
5839
5840 type = proof ,
5841     gender = f ,
5842     Name-sg = Demostración ,
5843     name-sg = demostración ,
5844     Name-pl = Demostraciones ,
5845     name-pl = demostraciones ,
5846
5847 type = result ,
5848     gender = m ,
5849     Name-sg = Resultado ,
5850     name-sg = resultado ,
5851     Name-pl = Resultados ,
5852     name-pl = resultados ,
5853
5854 type = remark ,
5855     gender = f ,
5856     Name-sg = Observación ,
5857     name-sg = observación ,
5858     Name-pl = Observaciones ,
5859     name-pl = observaciones ,
5860
5861 type = example ,
5862     gender = m ,
5863     Name-sg = Ejemplo ,
5864     name-sg = ejemplo ,
5865     Name-pl = Ejemplos ,
5866     name-pl = ejemplos ,
5867
5868 type = algorithm ,
5869     gender = m ,
5870     Name-sg = Algoritmo ,

```

```

5871   name-sg = algoritmo ,
5872   Name-pl = Algoritmos ,
5873   name-pl = algoritmos ,
5874
5875   type = listing ,
5876   gender = m ,
5877   Name-sg = Listado ,
5878   name-sg = listado ,
5879   Name-pl = Listados ,
5880   name-pl = listados ,
5881
5882   type = exercise ,
5883   gender = m ,
5884   Name-sg = Ejercicio ,
5885   name-sg = ejercicio ,
5886   Name-pl = Ejercicios ,
5887   name-pl = ejercicios ,
5888
5889   type = solution ,
5890   gender = f ,
5891   Name-sg = Solución ,
5892   name-sg = solución ,
5893   Name-pl = Soluciones ,
5894   name-pl = soluciones ,
5895 </lang-spanish>

```

10.6 Dutch

Dutch language file initially contributed by niluxv (PR [#5](#)). All genders were checked against the “Dikke Van Dale”. Many words have multiple genders.

```

5896 <*package>
5897 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
5898 </package>
5899 <*lang-dutch>
5900 namesep   = {\nobreakspace} ,
5901 pairsep   = {\~en\nobreakspace} ,
5902 listsep   = { , ~ } ,
5903 lastsep   = {\~en\nobreakspace} ,
5904 tpairsep  = {\~en\nobreakspace} ,
5905 tlistsep  = { , ~ } ,
5906 tlastsep  = { , ~en\nobreakspace} ,
5907 notesep   = { ~ } ,
5908 rangesep  = {\~t/m\nobreakspace} ,
5909
5910 type = book ,
5911   gender = n ,
5912   Name-sg = Boek ,
5913   name-sg = boek ,
5914   Name-pl = Boeken ,
5915   name-pl = boeken ,
5916
5917 type = part ,

```

```

5918 gender = n ,
5919 Name-sg = Deel ,
5920 name-sg = deel ,
5921 Name-pl = Delen ,
5922 name-pl = delen ,
5923
5924 type = chapter ,
5925 gender = n ,
5926 Name-sg = Hoofdstuk ,
5927 name-sg = hoofdstuk ,
5928 Name-pl = Hoofdstukken ,
5929 name-pl = hoofdstukken ,
5930
5931 type = section ,
5932 gender = m ,
5933 Name-sg = Paragraaf ,
5934 name-sg = paragraaf ,
5935 Name-pl = Paragrafen ,
5936 name-pl = paragrafen ,
5937
5938 type = paragraph ,
5939 gender = f ,
5940 Name-sg = Alinea ,
5941 name-sg = alinea ,
5942 Name-pl = Alinea's ,
5943 name-pl = alinea's ,
5944
5945 type = appendix ,
5946 gender = { m , n } ,
5947 Name-sg = Appendix ,
5948 name-sg = appendix ,
5949 Name-pl = Appendices ,
5950 name-pl = appendices ,
5951
5952 type = page ,
5953 gender = { f , m } ,
5954 Name-sg = Pagina ,
5955 name-sg = pagina ,
5956 Name-pl = Pagina's ,
5957 name-pl = pagina's ,
5958 rangesep = {\textendash} ,
5959
5960 type = line ,
5961 gender = m ,
5962 Name-sg = Regel ,
5963 name-sg = regel ,
5964 Name-pl = Regels ,
5965 name-pl = regels ,
5966
5967 type = figure ,
5968 gender = { n , f , m } ,
5969 Name-sg = Figuur ,
5970 name-sg = figuur ,
5971 Name-pl = Figuren ,

```

```

5972     name-pl = figuren ,
5973
5974 type = table ,
5975     gender = { f , m } ,
5976     Name-sg = Tabel ,
5977     name-sg = tabel ,
5978     Name-pl = Tabellen ,
5979     name-pl = tabellen ,
5980
5981 type = item ,
5982     gender = n ,
5983     Name-sg = Punt ,
5984     name-sg = punt ,
5985     Name-pl = Punten ,
5986     name-pl = punten ,
5987
5988 type = footnote ,
5989     gender = { f , m } ,
5990     Name-sg = Voetnoot ,
5991     name-sg = voetnoot ,
5992     Name-pl = Voetnoten ,
5993     name-pl = voetnoten ,
5994
5995 type = endnote ,
5996     gender = { f , m } ,
5997     Name-sg = Eindnoot ,
5998     name-sg = eindnoot ,
5999     Name-pl = Eindnoten ,
6000     name-pl = eindnoten ,
6001
6002 type = note ,
6003     gender = f ,
6004     Name-sg = Opmerking ,
6005     name-sg = opmerking ,
6006     Name-pl = Opmerkingen ,
6007     name-pl = opmerkingen ,
6008
6009 type = equation ,
6010     gender = f ,
6011     Name-sg = Vergelijking ,
6012     name-sg = vergelijking ,
6013     Name-pl = Vergelijkingen ,
6014     name-pl = vergelijkingen ,
6015     Name-sg-ab = Vgl. ,
6016     name-sg-ab = vgl. ,
6017     Name-pl-ab = Vgl.'s ,
6018     name-pl-ab = vgl.'s ,
6019     refbounds-first-sg = {,(,)}, ,
6020     refbounds = {(,,)} ,
6021
6022 type = theorem ,
6023     gender = f ,
6024     Name-sg = Stelling ,
6025     name-sg = stelling ,

```

```

6026 Name-pl = Stellingen ,
6027 name-pl = stellingen ,
6028

```

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”.

```

6029 type = lemma ,
6030   gender = n ,
6031   Name-sg = Lemma ,
6032   name-sg = lemma ,
6033   Name-pl = Lemma’s ,
6034   name-pl = lemma’s ,
6035
6036 type = corollary ,
6037   gender = n ,
6038   Name-sg = Gevolg ,
6039   name-sg = gevolg ,
6040   Name-pl = Gevolgen ,
6041   name-pl = gevolgen ,
6042
6043 type = proposition ,
6044   gender = f ,
6045   Name-sg = Propositie ,
6046   name-sg = propositie ,
6047   Name-pl = Propositions ,
6048   name-pl = propositions ,
6049
6050 type = definition ,
6051   gender = f ,
6052   Name-sg = Definitie ,
6053   name-sg = definitie ,
6054   Name-pl = Definities ,
6055   name-pl = definities ,
6056
6057 type = proof ,
6058   gender = n ,
6059   Name-sg = Bewijs ,
6060   name-sg = bewijs ,
6061   Name-pl = Bewijzen ,
6062   name-pl = bewijzen ,
6063
6064 type = result ,
6065   gender = n ,
6066   Name-sg = Resultaat ,
6067   name-sg = resultaat ,
6068   Name-pl = Resultaten ,
6069   name-pl = resultaten ,
6070
6071 type = remark ,
6072   gender = f ,
6073   Name-sg = Opmerking ,
6074   name-sg = opmerking ,
6075   Name-pl = Opmerkingen ,

```

```

6076   name-pl = opmerkingen ,
6077
6078   type = example ,
6079   gender = n ,
6080   Name-sg = Voorbeeld ,
6081   name-sg = voorbeeld ,
6082   Name-pl = Voorbeelden ,
6083   name-pl = voorbeelden ,
6084
6085   type = algorithm ,
6086   gender = { n , f , m } ,
6087   Name-sg = Algoritme ,
6088   name-sg = algoritme ,
6089   Name-pl = Algoritmes ,
6090   name-pl = algoritmes ,
6091

```

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

```

6092   type = listing ,
6093   gender = m ,
6094   Name-sg = Listing ,
6095   name-sg = listing ,
6096   Name-pl = Listings ,
6097   name-pl = listings ,
6098
6099   type = exercise ,
6100   gender = { f , m } ,
6101   Name-sg = Opgave ,
6102   name-sg = opgave ,
6103   Name-pl = Opgaven ,
6104   name-pl = opgaven ,
6105
6106   type = solution ,
6107   gender = f ,
6108   Name-sg = Oplossing ,
6109   name-sg = oplossing ,
6110   Name-pl = Oplossingen ,
6111   name-pl = oplossingen ,
6112 </lang-dutch>

```

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		4314, 4316, 4321, 4331, 4340, 4346, 4348, 4374, 4384, 4416, 4464, 4485
<code>\AddToHook</code>	100, 1271, 1318, 1342, 1373, 1375, 1413, 1486, 1502, 1523, 1654, 1667, 1675, 4187, 4223, 4240, 4242, 4247, 4292, 4294, 4296, 4298, 4300, 4302, 4304, 4306, 4310,	<code>\appendix</code> 2, 103, 106
		<code>\appendixname</code> 103
		<code>\AtEndOfPackage</code> 1665

B

<code>\babelname</code>	1328
<code>\babelprovide</code>	24, 39
<code>\bicaption</code>	105
<code>\bionenumcaption</code>	105
<code>\bitwonumcaption</code>	105
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_gset_false:N</code>	1083, 1100, 2347, 2355
<code>\bool_gset_true:N</code>	1045, 1062, 1641, 2326, 2334
<code>\bool_if:NTF</code>	393, 1275, 1279, 1677, 2429, 2825, 2954, 3199, 3221, 3249, 3266, 3276, 3281, 3328, 3332, 3386, 3411, 3415, 3421, 3431, 3437, 3650, 3808, 3810
<code>\bool_if:nTF</code>	68, 2543, 2553, 2577, 2594, 2609, 2674, 2682, 3259, 3608, 3697, 3714
<code>\bool_if_exist:NTF</code>	376
<code>\bool_lazy_all:nTF</code>	3356, 3820
<code>\bool_lazy_and:nnTF</code>	321, 2400, 2421, 3793, 3835
<code>\bool_lazy_any:nTF</code>	3972, 3981
<code>\bool_lazy_or:nnTF</code>	965, 1001, 1747, 1895, 2255, 2291, 2404, 3781
<code>\bool_new:N</code>	1158, 1159, 1186, 1210, 1219, 1226, 1233, 1246, 1247, 1421, 1422, 1423, 1424, 1425, 1516, 1517, 1634, 2437, 2452, 2714, 2715, 2726, 2727, 2734, 2735, 2746, 2747, 2759, 2760, 4424
<code>\bool_set:Nn</code>	2397
<code>\bool_set_eq:NN</code>	383
<code>\bool_set_false:N</code>	1171, 1175, 1254, 1263, 1264, 1281, 1288, 1433, 1437, 1444, 1452, 1453, 1454, 1539, 1777, 1932, 2442, 2535, 2775, 2792, 2831, 2842, 3211, 3377, 3384, 3385, 3979, 3996
<code>\bool_set_true:N</code>	1165, 1166, 1170, 1176, 1253, 1258, 1259, 1431, 1438, 1443, 1460, 1462, 1464, 1467, 1468, 1469, 1527, 1532, 1772, 1923, 2549, 2559, 2563, 2585, 2600, 2615, 2639, 2800, 2826, 2832, 2836, 2843, 2977, 2988, 2999, 3049, 3085, 3121, 3155, 3172, 3231, 3242, 3469, 3515, 3537, 3566, 3769, 3776, 3898, 3957, 3995, 4031, 4038, 4039, 4057, 4064, 4076, 4431
<code>\bool_until_do:Nn</code>	2575, 2793
<code>\bool_while_do:nn</code>	4516

C

<code>\caption</code>	108
clist commands:	
<code>\clist_map_inline:nn</code>	462, 800, 1455, 1582, 1643, 2043, 4398, 4478
<code>\contsubbottom</code>	105
<code>\contsubcaption</code>	105
<code>\counterwithin</code>	5
cs commands:	
<code>\cs_generate_variant:Nn</code>	65, 263, 269, 276, 287, 298, 309, 317, 318, 332, 352, 353, 373, 374, 752, 1803, 3644, 4009, 4118, 4151, 4184
<code>\cs_gset_eq:NN</code>	351, 372
<code>\cs_if_exist:NTF</code>	25, 28, 46, 49, 58, 78, 4211, 4217, 4338
<code>\cs_if_exist_p:N</code>	4518
<code>\cs_if_exist_use:N</code>	4309, 4556
<code>\cs_new:Npn</code>	56, 66, 76, 87, 264, 270, 272, 274, 277, 288, 299, 311, 517, 3604, 3645, 3999
<code>\cs_new_eq:NN</code>	4376, 4381
<code>\cs_new_protected:Npn</code>	258, 313, 315, 327, 344, 346, 348, 350, 369, 371, 594, 696, 1116, 1135, 1673, 1801, 2392, 2456, 2498, 2509, 2522, 2652, 2704, 2761, 2961, 3380, 3600, 3602, 3764, 4010, 4086, 4119, 4152, 4282, 4432
<code>\cs_new_protected:Npx</code>	99
<code>\cs_set:Npn</code>	4285
<code>\cs_set_eq:NN</code>	103, 349, 370, 519, 4284, 4290, 4377, 4382
<code>\cs_set_nopar:Npn</code>	4367

E

<code>\endinput</code>	12
<code>\endsidecaption</code>	105
exp commands:	
<code>\exp_args:NNe</code>	35, 38
<code>\exp_args:NNNo</code>	260
<code>\exp_args:NNNo</code>	260, 266
<code>\exp_args:No</code>	266
<code>\exp_args:Nx</code>	706, 4437, 4445
<code>\exp_args:Nxx</code> ..	4023, 4045, 4049, 4066
<code>\exp_not:N</code>	90, 3283, 3286, 3297, 3300, 3303, 3614, 3618, 3628, 3631, 3639, 3656, 3658, 3668, 3671, 3673, 3681, 3685, 3688, 3691, 3703, 3706, 3720, 3725, 3745, 3748, 3757
<code>\exp_not:n</code>	267, 2993, 3016, 3024, 3042, 3055, 3059, 3094, 3103, 3125, 3133, 3140, 3164, 3178, 3187, 3225, 3236, 3284, 3296, 3301, 3302,

3453, 3476, 3488, 3522, 3544, 3553, 3573, 3586, 3615, 3632, 3669, 3670, 3672, 3674, 3683, 3689, 3690, 3692, 3700, 3704, 3705, 3708, 3721, 3749 \ExplSyntaxOn 25, 712		\int_zero:N 2654, 2655, 2770, 2771, 2772, 2773, 2774, 3372, 3373, 3375, 3376, 3593, 3594 \l_tmpa_int 4515, 4519, 4526, 4527, 4530, 4532, 4534
F		iow commands:
file commands:		\iow_char:N 116, 131, 132, 137, 138, 143, 144, 149, 150, 202, 214 \iow_newline: 255
\file_get:nnNTF 706 \fmtversion 3 \footnote 2, 106, 107		J
G		\jobname 4331
group commands:		K
\group_begin: 102, 533, 698, 1955, 2394, 2408, 2440, 3283, 3300, 3614, 3631, 3656, 3668, 3673, 3688, 3703, 3720, 3748, 4297, 4301, 4305 \group_end: 105, 545, 750, 1983, 2411, 2434, 2444, 3297, 3303, 3628, 3639, 3671, 3681, 3685, 3691, 3706, 3745, 3757, 4299, 4303, 4307		keys commands:
I		\l_keys_choice_tl 586 \keys_define:nn 16, 470, 561, 753, 837, 865, 902, 947, 1031, 1142, 1160, 1187, 1196, 1211, 1220, 1227, 1234, 1240, 1248, 1283, 1292, 1306, 1338, 1377, 1408, 1415, 1427, 1488, 1495, 1497, 1504, 1511, 1518, 1528, 1540, 1549, 1578, 1604, 1628, 1636, 1656, 1685, 1703, 1733, 1767, 1790, 1812, 1824, 1848, 1878, 1918, 1986, 2070, 2104, 2153, 2183, 2235, 2319 \keys_set:nn 22, 25, 41, 42, 63, 542, 674, 737, 1533, 1802, 1807, 1980, 2395
\IfBooleanT 2441 \IfClassLoadedTF 112 \ifdraft 1436 \IfFormatAtLeastTF 3, 4 \ifoptionfinal 1442 \IfPackageLoadedTF 110 \input 24, 25		keyval commands:
int commands:		\keyval_parse:nnn . . . 1121, 1553, 1608 \KOMAClassName 4338, 4360
\int_case:nnTF 2964, 3006, 3068, 3335, 3444, 3501 \int_compare:nNnTF 2586, 2601, 2616, 2628, 2640, 2660, 2662, 2706, 2873, 2984, 3012, 3034, 3089, 3159, 3320, 3322, 3397, 3424, 3482, 4027, 4033, 4053, 4059, 4534 \int_compare_p:nNn 968, 1004, 1749, 1897, 2258, 2294, 2676, 2684, 3360, 3785, 3796, 3825, 3992 \int_eval:n 99 \int_incr:N 3374, 3405, 3414, 3416, 3430, 3432, 3436, 3438, 3450, 3473, 3485, 3519, 3541, 3550, 3570, 3598, 4532 \int_new:N 2453, 2454, 2716, 2717, 2718, 2731, 2732 \int_rand:n 285, 296, 307 \int_set:Nn 2661, 2663, 2667, 2670, 4515 \int_to_roman:n 4519, 4526, 4527, 4530 \int_use:N 47, 50, 54, 60		L
		\label 105, 108, 109, 111, 4284, 4290 \labelformat 3 \language name 38, 1322
		M
		\mainbabelname 38, 1329 \MessageBreak 10 MH commands:
		\MH_if_boolean:nTF 4429 msg commands:
		\msg_info:nnn 740, 793, 856, 909, 4261, 4333, 4360, 4421, 4456, 4488, 4508, 4535 \msg_info:nnnn 766, 773, 805, 981, 1018 \msg_info:nnnnn 787 \msg_line_context: 115, 121, 125, 127, 130, 136, 142, 148, 154, 159, 164, 169, 174, 180, 185, 188, 191, 196, 200, 207, 212, 220, 224, 233, 238, 242, 244, 246, 248, 255 \msg_new:nnn 113, 119, 124, 126, 128, 134, 140,

146, 152, 157, 162, 167, 172, 177, 182, 187, 189, 194, 199, 201, 203, 205, 210, 216, 218, 223, 225, 230, 236, 241, 243, 245, 247, 249, 251, 253	\prop_if_in_p:Nn 69
\msg_warning:nn 1280, 1286, 1507, 1544, 3362	\prop_item:Nn 38, 70
\msg_warning:nnn 537, 557, 1148, 1362, 1399, 1411, 1473, 1484, 1610, 1660, 1670, 1816, 1982, 2026, 2095, 2190, 2857, 3206, 3814, 3830	\prop_new:N 1548, 1603
\msg_warning:nnnn 610, 627, 661, 679, 1311, 1555, 1708, 1714, 1720, 1726, 1756, 1853, 1859, 1865, 1871, 1907, 1999, 2006, 2048, 2158, 2164, 2170, 2176, 2271, 2307, 2865, 3899, 3959	\prop_put:Nnn 1139
\msg_warning:nnnnn 647, 686, 2020, 3849	\prop_remove:Nn 1138
\msg_warning:nnnnnn 3856	\providecommand 3
N	\ProvidesExplPackage 14
\newcounter 5, 4207, 4208	\ProvidesFile 24
\NewDocumentCommand 531, 548, 1799, 1804, 1953, 2390, 2438	R
\newfloat 106	\refstepcounter 3, 105, 108–110, 112, 113
\newsubfloat 106	\relax 15
\nobreakspace 1129, 4592, 4593, 4595, 4596, 4598, 4600, 4797, 4798, 4800, 4801, 4803, 4805, 5246, 5247, 5249, 5250, 5252, 5254, 5462, 5463, 5465, 5466, 5468, 5470, 5687, 5688, 5690, 5691, 5693, 5695, 5900, 5901, 5903, 5904, 5906, 5908	\renewlist 113
P	\RequirePackage 16, 17, 18, 19, 20, 1276, 1499, 1520
\PackageError 7	S
\pagenote 106, 107	scan commands:
\pagenumbering 7	\scan_stop: 15, 349, 351, 370, 372
\pageref 64	\scantokens 103
\paragraph 44, 45	seq commands:
prg commands:	\seq_clear:N 605, 642, 723, 736, 799, 1207, 1744, 1892, 1966, 1979, 2042, 2458, 4145
\prg_generate_conditional_	\seq_const_from_clist:Nn 403, 411, 422, 436, 442
variant:Nnn 342, 356, 367, 377, 388, 396, 527	\seq_count:N 969, 983, 1005, 1020, 1749, 1758, 1897, 1909, 2259, 2273, 2295, 2309
\prg_new_conditional:Npnn 109, 111, 319, 354, 375, 390, 521	\seq_gclear:N 962, 998, 2252, 2288
\prg_new_protected_conditional:Npnn 333, 358, 379	\seq_gconcat:NNN 455, 459
\prg_return_false: 110, 112, 325, 340, 355, 365, 376, 386, 393, 394, 525	\seq_get_left:NN 620, 631, 727, 775, 1970, 2008, 2803
\prg_return_true: 110, 112, 324, 338, 355, 363, 376, 384, 393, 524	\seq_gput_right:Nn 738, 744, 1647
\prg_set_eq_conditional:NNn 529	\seq_gremove_all:Nn 1679
\ProcessKeysOptions 1798	\seq_gset_eq:NN 819, 972, 1008, 2053, 2262, 2298
prop commands:	\seq_gset_from_clist:Nn 565, 575
\prop_if_in:NnTF 35	\seq_gset_split:Nnn 347
	\seq_if_empty:NTF 606, 643, 724, 764, 785, 1967, 1997, 2018, 2797, 3847
	\seq_if_exist:NTF 355
	\seq_if_in:NnTF 624, 658, 702, 770, 802, 1584, 1645, 1678, 2003, 2045, 2502, 3843
	\seq_item:Nn 3616, 3622, 3625, 3627, 3633, 3634, 3637, 3638, 3675, 3676, 3680, 3684, 3722, 3735, 3740, 3743, 3750, 3751, 3755, 3756
	\seq_map_break:n 90, 2695, 2698
	\seq_map_function:NN 2461
	\seq_map_indexed_inline:Nn 35, 2656
	\seq_map_inline:Nn 834, 862, 899, 944, 1028, 1669, 1682, 1730,

1764, 1809, 1821, 1875, 1915, 2067, 2101, 2180, 2232, 2316, 2692, 4435	
\seq_map_tokens:Nn	72
\seq_new:N	401, 402, 454, 458, 695, 1195, 1577, 1635, 2436, 2455, 2713, 2730, 2748, 2749, 2750, 2751, 2752, 2753, 2754, 2755, 2756, 2757, 2758
\seq_pop_left:NN	2795
\seq_put_right:Nn	803, 1586, 2046, 2505
\seq_reverse:N	1201
\seq_set_eq:NN	362, 1751, 1899, 2763, 2975, 2986, 2997, 3047, 3083, 3119, 3153, 3169, 3229, 3240, 3251, 3467, 3512, 3534, 3563
\seq_set_from_clist:Nn	1200, 2396
\seq_set_split:Nnn	345
\seq_sort:Nn	66, 2464
\seq_use:Nn	3861
\g_tmpa_seq	962, 964, 969, 978, 983, 998, 1000, 1005, 1015, 1020, 2252, 2254, 2259, 2268, 2273, 2288, 2290, 2295, 2304, 2309
\l_tmpa_seq	799, 803, 827, 1744, 1746, 1749, 1753, 1758, 1892, 1894, 1897, 1904, 1909, 2042, 2046, 2061
\setcounter	4209, 4210, 4226, 4241, 4245
\sidefootnote	106, 107
sort commands:	
\sort_return_same:	66, 70, 2471, 2476, 2550, 2570, 2591, 2606, 2620, 2645, 2680, 2695, 2711
\sort_return_swapped:	66, 70, 2484, 2560, 2569, 2590, 2605, 2621, 2644, 2688, 2698, 2710
\stepcounter	112, 4225, 4244
str commands:	
\str_case:nnTF	1344, 1381, 1457
\str_compare:nNnTF	2566
\str_if_eq:nnTF	89
\str_if_eq_p:nn	3977, 3983, 3985, 3989
\str_new:N	1291
\str_set:Nn	1296, 1298, 1300, 1302
\string	4442, 4450
\subbottom	105
\subcaption	105
\subcaptionref	105
\subref	114
\subsubsection	44, 45
\subsubsection	45
\subtop	105
T	
\tag	110, 112

TeX and L^AT_EX 2_ε commands:

\@@sidecaption	105
\@Alph	103
\@addtoreset	5
\@auxout	4441, 4449
\@bsphack	699, 4434
\@capttype	108, 4309, 4342, 4343, 4347, 4556
\@chapapp	103
\@currentcounter	2, 3, 5, 46, 106, 108–110, 113, 28, 29, 49, 50, 1632, 4343, 4355
\@currentlabel	3, 106, 108, 113
\@currenvir	4350
\@elt	5
\@esphack	749, 4454
\@ifl@t@r	3
\@mem@scap@afterhook	105
\@memsubcaption	106
\@onlypreamble	547, 560, 1985
\@raw@opt@⟨package⟩.sty	43
\bbl@loaded	39
\bbl@main@language	38, 1323
\c@lstnumber	113
\c@page	7, 103
\caption@subtypehook	4557
\hyper@link	90, 3286, 3618, 3658, 3725
\lst@AddToHook	4504, 4506
\lst@Init	113
\lst@label	4505
\lst@MakeCaption	113
\ltx@gobble	108
\ltx@label	109, 4376, 4377, 4381, 4382
\m@mscaplabel	105
\MT@newlabel	4442, 4450
\protected@write	4441, 4449
\zref@addprop	22, 32, 43, 53, 55, 97, 108
\zref@default	90, 3601, 3603
\zref@extractdefault	10, 11, 98, 261, 267, 271
\zref@ifpropundefined	34, 1146, 4001
\zref@ifrefcontainsprop	34, 2860, 3606, 3652, 3710, 4004
\zref@ifrefundefined	2466, 2468, 2480, 2828, 2830, 2835, 2852, 3203, 3212, 3388, 3647, 3766
\zref@label	108, 4370
\zref@localaddprop	4311, 4558
\ZREF@mainlist	22, 32, 43, 53, 55, 97, 108, 4311, 4558
\zref@newprop	5, 7, 21, 23, 33, 44, 54, 92, 107, 4308, 4555
\zref@refused	2850
\zref@wrapper@babel	62, 108, 2391, 4370

<code>\textendash</code>		<code>\tl_new:N</code> 98, 398, 399, 400, 539, 1141, 1315, 1316, 1317, 1407, 1426, 1494, 1510, 1627, 2446, 2447, 2448, 2449, 2450, 2451, 2719, 2720, 2721, 2722, 2723, 2724, 2725, 2728, 2729, 2733, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744, 2745, 4312, 4345
<code>\textup</code>	110	<code>\tl_put_left:Nn</code> . 3262, 3269, 3313, 3883, 3884, 3926, 3928, 3930, 3932
<code>\thechapter</code>	103	<code>\tl_put_right:Nn</code>
<code>\thelstnumber</code>	113	2991, 3014, 3022, 3040, 3053, 3092, 3101, 3123, 3131, 3138, 3162, 3176, 3185, 3451, 3474, 3486, 3520, 3542, 3551, 3571, 3584, 3791, 3792, 3803, 4557
<code>\thepage</code>	7, 104	<code>\tl_reverse:N</code>
<code>\thesection</code>	103	<code>\tl_set:Nn</code>
tl commands:		260, 541, 715, 759, 771, 1149, 1151, 1322, 1323, 1328, 1329, 1332, 1333, 1337, 1348, 1354, 1359, 1385, 1391, 1396, 1696, 1806, 1838, 1959, 1992, 2004, 2633, 2635, 2809, 2810, 2971, 2973, 3223, 3234, 3279, 3401, 3403, 3428, 3787, 3788, 3801, 4313, 4315
<code>\c_novalue_tl</code>	314, 316, 479, 486, 493, 499, 505, 512, 514, 782, 839, 867, 904, 949, 1687, 1735, 1826, 1880, 2015, 2072, 2237	<code>\tl_set_eq:NN</code> 314, 337, 3366, 4343, 4354
<code>\tl_clear:N</code>	614, 652, 665, 682, 691, 716, 725, 758, 1958, 1968, 1991, 2765, 2766, 2767, 2768, 2769, 2799, 3368, 3369, 3370, 3371, 3413, 3768, 3775, 3805, 3897, 3956, 4112	<code>\tl_show:N</code>
<code>\tl_const:Nn</code>	1118	<code>\tl_tail:N</code>
<code>\tl_gclear:N</code>	4358	<code>\tl_use:N</code>
<code>\tl_gset:Nn</code>	104, 330, 540, 554, 2086, 2120, 2140, 2207, 2217	282, 293, 304, 555, 704, 709, 739, 741, 745
<code>\tl_gset_eq:NN</code>	316, 4347	<code>\l_tmpa_tl</code>
<code>\tl_head:N</code>		713, 737, 2417, 2418
..	2604, 2617, 2629, 2631, 2641, 2643	
<code>\tl_if_empty:NTF</code>	80, 608, 618, 645, 656, 677, 684, 791, 842, 870, 907, 913, 952, 1036, 1074, 2024, 2075, 2109, 2188, 2205, 2240, 2324, 2345, 2366, 2412, 3201, 3707, 3790, 3812, 3870, 3881, 3924, 4352, 4505	U
<code>\tl_if_empty:nTF</code>	534, 550, 757, 1137, 1640, 1990, 3081, 3117, 3151, 3465, 3510, 3532, 3561	<code>\upshape</code>
<code>\tl_if_empty_p:N</code>		4420
.....	3823, 3837, 3976, 3986, 3990	use commands:
<code>\tl_if_empty_p:n</code>	966, 1002, 1748, 1896, 2256, 2292, 2545, 2546, 2555, 2556, 2581, 2582, 2597, 2612	<code>\use:N</code>
<code>\tl_if_eq:NNTF</code>	2516, 2539, 2839, 4017	26, 29, 586, 4178
<code>\tl_if_eq:NnTF</code>		
...	2459, 2491, 2666, 2669, 2694, 2697, 2807, 2855, 3772, 4021, 4350	V
<code>\tl_if_eq:nnTF</code>	2658, 4023, 4045, 4049, 4066, 4437, 4445	<code>\value</code>
<code>\tl_if_exist:NTF</code>	523, 4342	4226, 4245
<code>\tl_if_exist_p:N</code>	322	<code>\verbfootnote</code>
<code>\tl_if_novalue:nTF</code>		106, 107
..	797, 844, 872, 884, 915, 928, 954, 989, 1690, 1738, 1829, 1883, 2030, 2077, 2111, 2129, 2194, 2242, 2278	
<code>\tl_if_novalue_p:n</code>	323	Z
<code>\tl_map_break:n</code>	90	<code>\zcDeclareLanguage</code>
<code>\tl_map_tokens:Nn</code>	82	12, 20, 531, 4582, 4786, 5239, 5456, 5684, 5897
		<code>\zcDeclareLanguageAlias</code>
	
		21, 548, 4583, 4584, 4585, 4586, 4587, 4588, 4589, 4789, 4790, 4791, 4792, 4793, 4794, 5240, 5241, 5242, 5243, 5457, 5458, 5459
		<code>\zcLanguageSetup</code> . 17, 24, 25, 51, 54, 1953
		<code>\zcpageref</code>
		64, 2438
		<code>\zceref</code>
		43, 48, 50, 62, 64–66, 72, 74, 111, 2390, 2443
		<code>\zcRefTypeSetup</code>
		17, 51, 1804, 4419
		<code>\zcsetup</code>
		38, 43, 48, 50, 1799
		<code>\zlabel</code> 105, 108, 109, 111, 113, 4288, 4505

zrefcheck commands:

- \zrefcheck_zceref_beg_label: .. 2403
- \zrefcheck_zceref_end_label_-
maybe: 2425
- \zrefcheck_zceref_run_checks_on_-
labels:n 2426

zrefclever commands:

- \zrefclever_language_if_declared:nTF
..... 529
- \zrefclever_language_varname:n . 519
- \l_zrefclever_ref_language_tl . 1337

zrefclever internal commands:

- \l__zrefclever_abbrev_bool
..... 2736, 2910, 3794
- \l__zrefclever_cap_bool
..... 2736, 2906, 3782
- \l__zrefclever_capfirst_bool ...
..... 1226, 1229, 3784
- __zrefclever_compat_module:nn ..
..... 47,
1673, 4185, 4203, 4264, 4336, 4363,
4425, 4460, 4491, 4511, 4538, 4561
- __zrefclever_counter_reset_by:n
. 6, 45, 46, 58, 60, 62, 66, 4391, 4471
- __zrefclever_counter_reset_by_-
aux:nn 73, 76
- __zrefclever_counter_reset_by_-
auxi:nnn 83, 87
- \l__zrefclever_counter_resetby_-
prop 5, 45, 69, 70, 1603, 1615
- \l__zrefclever_counter_resettters_-
seq . 5, 45, 46, 72, 1577, 1584, 1587
- \l__zrefclever_counter_type_prop
..... 4, 44, 35, 38, 1548, 1560
- \l__zrefclever_current_counter_-
tl 3, 5, 46, 21, 25,
26, 36, 39, 41, 46, 47, 95, 1627, 1630
- \l__zrefclever_current_language_-
tl 38,
1316, 1322, 1328, 1332, 1349, 1386
- __zrefclever_extract:nnn
..... 11, 270, 2587, 2589,
2602, 2619, 2707, 2709, 4028, 4030,
4034, 4036, 4054, 4056, 4060, 4062
- __zrefclever_extract_default:Nnnn
..... 10,
258, 2500, 2511, 2513, 2524, 2527,
2530, 2532, 2813, 2816, 4012, 4014
- __zrefclever_extract_unexp:nnn .
11, 98, 264, 3292, 3620, 3623, 3635,
3664, 3677, 3731, 3737, 3752, 4002,
4005, 4006, 4024, 4025, 4046, 4047,
4050, 4051, 4068, 4072, 4438, 4446
- __zrefclever_extract_url_-
unexp:n 3288, 3619, 3660, 3727, 3999
- __zrefclever_get_enclosing_-
counters_value:n . 5, 6, 56, 61, 94
- __zrefclever_get_ref:nN
..... 90, 91, 2994, 3017,
3025, 3043, 3056, 3060, 3095, 3104,
3126, 3134, 3141, 3165, 3179, 3188,
3226, 3237, 3271, 3454, 3477, 3489,
3523, 3545, 3554, 3574, 3587, 3604
- __zrefclever_get_ref_first: ...
..... 90, 94, 3263, 3314, 3645
- __zrefclever_get_rf_opt_bool:nN 102
- __zrefclever_get_rf_opt_-
bool:nnnnN ... 16, 2903, 2907, 4152
- __zrefclever_get_rf_opt_-
seq:nnnN 16, 101,
2911, 2915, 2919, 2923, 2927, 2931,
2935, 2939, 2943, 2947, 3839, 4119
- __zrefclever_get_rf_opt_tl:nnnN
..... 16, 18, 100,
2414, 2778, 2782, 2786, 2875, 2879,
2883, 2887, 2891, 2895, 2899, 4086
- \l__zrefclever_hyperlink_bool ...
... 1246, 1253, 1258, 1263, 1275,
1281, 1288, 2442, 3610, 3716, 3974
- \l__zrefclever_hyperref_warn_-
bool ... 1247, 1254, 1259, 1264, 1279
- __zrefclever_if_class_loaded:n 109
- __zrefclever_if_class_loaded:nTF
..... 4266
- __zrefclever_if_package_-
loaded:n 109
- __zrefclever_if_package_-
loaded:nTF 1273,
1320, 1326, 1525, 4205, 4365, 4372,
4427, 4462, 4493, 4513, 4540, 4563
- \g__zrefclever_koma_captionofbeside_-
captype_tl 4345
- \g__zrefclever_koma_captype_tl ..
..... 4347, 4352, 4355, 4358
- \l__zrefclever_label_a_tl
..... 71, 2719, 2796, 2815, 2828,
2850, 2852, 2858, 2861, 2867, 2972,
2994, 3017, 3025, 3060, 3126, 3141,
3188, 3217, 3226, 3237, 3388, 3392,
3402, 3429, 3454, 3490, 3554, 3587
- \l__zrefclever_label_b_tl
..... 71, 2719,
2799, 2804, 2818, 2830, 2835, 3392
- \l__zrefclever_label_count_int ...
..... 72, 2716, 2770,
2873, 2964, 3372, 3397, 3598, 3826

`\l_zrefclever_label_enclval_a_-`
`tl 2446, 2524, 2526, 2581,`
`2597, 2617, 2629, 2633, 2634, 2641`
`\l_zrefclever_label_enclval_b_-`
`tl 2446, 2527, 2529, 2582,`
`2604, 2612, 2631, 2635, 2636, 2643`
`\l_zrefclever_label_extdoc_a_tl`
`. 2446, 2530,`
`2540, 2545, 2555, 2568, 4012, 4018`
`\l_zrefclever_label_extdoc_b_tl`
`. 2446, 2532,`
`2541, 2546, 2556, 2567, 4014, 4019`
`\l_zrefclever_label_type_a_tl . .`
`. 2415,`
`2446, 2501, 2503, 2506, 2512, 2517,`
`2666, 2694, 2779, 2783, 2787, 2809,`
`2814, 2840, 2855, 2876, 2880, 2884,`
`2888, 2892, 2896, 2900, 2904, 2908,`
`2912, 2916, 2920, 2924, 2928, 2932,`
`2936, 2940, 2944, 2948, 2974, 3404`
`\l_zrefclever_label_type_b_tl . .`
`. 2446, 2514,`
`2518, 2669, 2697, 2810, 2817, 2841`
`_zrefclever_label_type_put_-`
`new_right:n 65, 66, 2462, 2498`
`\l_zrefclever_label_types_seq . .`
`. . . . 66, 2455, 2458, 2502, 2505, 2692`
`_zrefclever_labels_in_sequence:nn`
`. 72, 99, 3215, 3391, 4010`
`\l_zrefclever_lang_decl_case_tl`
`. . . . 398, 725, 728, 771, 776, 913,`
`935, 1968, 1971, 2004, 2009, 2205, 2222`
`\l_zrefclever_lang_declension_-`
`seq 398,`
`604, 605, 606, 620, 624, 631, 722,`
`723, 724, 727, 764, 770, 775, 1965,`
`1966, 1967, 1970, 1997, 2003, 2008`
`\l_zrefclever_lang_gender_seq . .`
`. . . . 398, 641, 642, 643, 658, 735,`
`736, 785, 802, 1978, 1979, 2018, 2045`
`_zrefclever_language_if_-`
`declared:n 20, 530`
`_zrefclever_language_if_-`
`declared:n(TF) 20`
`_zrefclever_language_if_-`
`declared:nTF 279, 290, 301, 521,`
`536, 552, 596, 700, 1360, 1397, 1956`
`_zrefclever_language_varname:n`
`. 19, 20,`
`282, 293, 304, 517, 520, 523, 539,`
`540, 554, 555, 704, 709, 739, 741, 745`
`\l_zrefclever_last_of_type_bool`
`. 71, 2713, 2826,`
`2831, 2832, 2836, 2842, 2843, 2954`
`\l_zrefclever_lastsep_tl . 2736,`
`2890, 3024, 3059, 3103, 3140, 3178`
`\l_zrefclever_link_star_bool . . .`
`. 2397, 2436, 3611, 3717, 3975`
`\l_zrefclever_listsep_tl`
`. . . 2736, 2886, 3055, 3133, 3453,`
`3476, 3488, 3522, 3544, 3553, 3573`
`\g_zrefclever_loaded_langfiles_-`
`seq 695, 703, 738, 744`
`_zrefclever_ltxlabel:n`
`. 109, 4367, 4377, 4382`
`\l_zrefclever_main_language_tl .`
`38, 1317, 1323, 1329, 1333, 1355, 1392`
`_zrefclever_mathtools_showonlyrefs:n`
`. 2431, 4432`
`\l_zrefclever_mathtools_-`
`showonlyrefs_bool 2429, 4424, 4431`
`_zrefclever_memoir_both_-`
`labels:`
`. . 4282, 4293, 4295, 4297, 4301, 4305`
`\l_zrefclever_memoir_footnote_-`
`type_tl 4312, 4313, 4315, 4319`
`_zrefclever_memoir_label_and_-`
`zlabel:n 4285, 4290`
`_zrefclever_memoir_orig_-`
`label:n 4284, 4287`
`_zrefclever_name_default:`
`. 3600, 3699`
`\l_zrefclever_name_format_-`
`fallback_tl 2725, 3801,`
`3805, 3870, 3919, 3931, 3933, 3951`
`\l_zrefclever_name_format_tl . . .`
`. . . 2725, 3787, 3788, 3791, 3792,`
`3802, 3803, 3876, 3883, 3884, 3892,`
`3900, 3910, 3927, 3928, 3941, 3961`
`\l_zrefclever_name_in_link_bool`
`. 91,`
`94, 2725, 3281, 3650, 3979, 3995, 3996`
`\l_zrefclever_namefont_tl 2736,`
`2898, 3284, 3301, 3669, 3689, 3704`
`\l_zrefclever_nameinlink_str . . .`
`. 1291, 1296, 1298,`
`1300, 1302, 3977, 3983, 3985, 3989`
`\l_zrefclever_namesep_tl`
`. . 2736, 2878, 3672, 3692, 3700, 3708`
`\l_zrefclever_next_is_same_bool`
`. 72, 99, 2731,`
`3385, 3415, 3431, 3437, 4039, 4077`
`\l_zrefclever_next_maybe_range_-`
`bool`
`. . 72, 99, 2731, 3211, 3221, 3384,`
`3411, 3421, 4031, 4038, 4057, 4065`
`\l_zrefclever_noabbrev_first_-`
`bool 1233, 1236, 3798`

\g_zrefclever_nocompat_bool ...	_zrefclever_opt_tl_cset_-
..... 1634, 1641, 1677	fallback:nn 1116, 1123
\l_zrefclever_nocompat_bool ... 47	_zrefclever_opt_tl_get:NN(TF) . 14
\g_zrefclever_nocompat_modules_-	_zrefclever_opt_tl_get:NNTF ...
seq 1635, 1645, 1648, 1669, 1678, 1679	333, 3872, 3887, 3906, 3915, 3936,
\l_zrefclever_nocompat_modules_-	3946, 4089, 4094, 4099, 4104, 4109
seq 47	_zrefclever_opt_tl_gset_if_-
\l_zrefclever_nudge_comptosing_-	new:Nn 14, 327, 846, 874, 886, 917, 930
bool ... 1423, 1453, 1462, 1468, 3822	_zrefclever_opt_tl_gunset:N ...
\l_zrefclever_nudge_enabled_- 13, 313, 2079, 2113, 2131, 2196
bool 1421, 1431, 1433,	_zrefclever_opt_tl_if_set:N ... 13
1437, 1438, 1443, 1444, 3358, 3808	_zrefclever_opt_tl_if_set:N(TF)
\l_zrefclever_nudge_gender_bool 13
..... 1425, 1454, 1464, 1469, 3836	_zrefclever_opt_tl_if_set:NNTF .
\l_zrefclever_nudge_multitype_- 319, 329, 335
bool ... 1422, 1452, 1460, 1467, 3359	_zrefclever_opt_tl_unset:N ...
\l_zrefclever_nudge_singular_- 13, 313, 1692, 1831
bool 1424, 1480, 3810	_zrefclever_opt_varname_-
_zrefclever_opt_bool_get:NN(TF)	fallback:nn 13, 311, 1119, 4110, 4143, 4176
..... 16	_zrefclever_opt_varname_-
_zrefclever_opt_bool_get:NNTF .	general:nn 11, 272, 1693, 1697, 1741, 1752,
... 379, 4155, 4160, 4165, 4170, 4175	1773, 1778, 1783, 4090, 4123, 4156
_zrefclever_opt_bool_gunset:N .	_zrefclever_opt_varname_lang_-
..... 15, 369, 2368, 2376	default:nnn . 12, 288, 848, 876,
_zrefclever_opt_bool_if:N 15	958, 974, 1040, 1047, 1078, 1085,
_zrefclever_opt_bool_if:N(TF) . 16	2081, 2088, 2115, 2122, 2246, 2264,
_zrefclever_opt_bool_if:NNTF ...	2328, 2349, 2370, 4105, 4138, 4171
..... 390, 669	_zrefclever_opt_varname_lang_-
_zrefclever_opt_bool_if_set:N . 15	type:nnnn 13,
_zrefclever_opt_bool_if_-	299, 812, 821, 888, 919, 932, 993,
set:N(TF) 15	1010, 1056, 1064, 1094, 1102, 2034,
_zrefclever_opt_bool_if_-	2055, 2133, 2142, 2198, 2209, 2219,
set:NNTF 375,	2282, 2300, 2336, 2357, 2378,
381, 392, 1038, 1054, 1076, 1092	3889, 3938, 3948, 4100, 4133, 4166
_zrefclever_opt_bool_unset:N .	_zrefclever_opt_varname_-
..... 15, 369, 1782, 1941	language:nnn . 12, 277, 567, 577,
_zrefclever_opt_seq_get:NN(TF) 15	588, 601, 638, 671, 719, 732, 1962, 1975
_zrefclever_opt_seq_get:NNTF .	_zrefclever_opt_varname_-
... 358, 599, 636, 717, 730, 1960,	type:nnn 12, 274, 1833,
1973, 4122, 4127, 4132, 4137, 4142	1840, 1887, 1901, 1925, 1934, 1943,
_zrefclever_opt_seq_gset_-	3874, 3908, 3917, 4095, 4128, 4161
clist_split:Nn 14, 344, 963, 999, 2253, 2289	_zrefclever_orig_ltxlabel:n ...
_zrefclever_opt_seq_gunset:N 4369, 4376, 4381
..... 14, 348, 2032, 2244, 2280	_zrefclever_page_format_aux: .
_zrefclever_opt_seq_if_set:N . 14 99, 103
_zrefclever_opt_seq_if_-	\g_zrefclever_page_format_tl ...
set:N(TF) 15 7, 98, 104, 107
_zrefclever_opt_seq_if_set:NNTF	\l_zrefclever_pairsep_tl 2736, 2882, 2993,
..... 354, 360, 810, 956, 991	3016, 3042, 3094, 3125, 3164, 3225
_zrefclever_opt_seq_set_clist_-	_zrefclever_process_language_-
split:Nn 14, 344, 1745, 1893	settings: 41, 42, 594, 2399
_zrefclever_opt_seq_unset:N ...	
..... 14, 348, 1740, 1885	


```

\__zrefclever_prop_put_non-
  empty:Nnn . . . . . 34, 1135, 1559, 1614
\__zrefclever_provide_langfile:n
  . . . . . 17, 25, 26, 63, 696, 1366, 2398
\l_zrefclever_range_beg_label_-
  tl 72, 2731, 2769, 3044, 3057, 3081,
    3096, 3105, 3117, 3135, 3151, 3166,
    3180, 3371, 3413, 3428, 3465, 3478,
    3510, 3524, 3532, 3546, 3561, 3575
\l_zrefclever_range_count_int . .
  . . . . . 72,
    2731, 2773, 3006, 3070, 3375, 3414,
    3425, 3430, 3436, 3444, 3503, 3593
\l_zrefclever_range_same_count_-
  int . . . . . 72,
    2731, 2774, 2984, 3035, 3071, 3376,
    3416, 3432, 3438, 3483, 3504, 3594
\l_zrefclever_rangeseq_tl . . . .
  . . . . . 2736, 2894, 3187, 3236, 3586
\l_zrefclever_ref_count_int . . .
  . . . . . 2716, 2772,
    3012, 3090, 3160, 3373, 3405, 3450,
    3473, 3485, 3519, 3541, 3550, 3570
\l_zrefclever_ref_decl_case_tl .
  . . . . . 22, 608, 613, 614, 618, 621,
    625, 629, 632, 677, 680, 682, 1407,
    1417, 3881, 3885, 3924, 3929, 3934
\__zrefclever_ref_default: . . . .
  . . . . . 3600, 3642, 3648, 3693, 3760
\l_zrefclever_ref_gender_tl . . .
  . . . . . 23, 645, 651,
    652, 656, 659, 664, 665, 684, 690,
    691, 1426, 1490, 3837, 3845, 3851, 3859
\l_zrefclever_ref_language_tl . .
  . . . . . 22, 38, 39,
    597, 602, 612, 630, 639, 649, 663,
    672, 681, 688, 1315, 1337, 1348,
    1354, 1359, 1367, 1385, 1391, 1396,
    2398, 2416, 2780, 2784, 2788, 2877,
    2881, 2885, 2889, 2893, 2897, 2901,
    2905, 2909, 2913, 2917, 2921, 2925,
    2929, 2933, 2937, 2941, 2945, 2949,
    3841, 3853, 3864, 3890, 3939, 3949
\l_zrefclever_ref_property_tl . .
  . 34, 1141, 1149, 1151, 2491, 2807,
    2862, 2866, 3606, 3654, 3712, 4021
\l_zrefclever_ref_propserity_tl 2459
\l_zrefclever_ref_typeset_font_-
  tl . . . . . 1494, 1496, 2409
\l_zrefclever_refbounds_first_-
  pb_seq . . . . . 2748,
    2922, 2998, 3048, 3120, 3171, 3230
\l_zrefclever_refbounds_first_-
  rb_seq . 2748, 2926, 3154, 3241, 3565
\l_zrefclever_refbounds_first_-
  seq 2748, 2914, 3252, 3468, 3514, 3536
\l_zrefclever_refbounds_first_-
  sg_seq . 2748, 2918, 2976, 2987, 3084
\l_zrefclever_refbounds_last_-
  pe_seq . . . . . 2748, 2946,
    2995, 3018, 3045, 3097, 3127, 3227
\l_zrefclever_refbounds_last_-
  re_seq . . . . . 2748, 2950, 3189, 3238
\l_zrefclever_refbounds_last_-
  seq 2748, 2942, 3026, 3061, 3106, 3142
\l_zrefclever_refbounds_mid_rb_-
  seq . . . 2748, 2934, 3167, 3181, 3576
\l_zrefclever_refbounds_mid_re_-
  seq . . . . . 2748, 2938, 3588
\l_zrefclever_refbounds_mid_seq
  . . . . . 2748, 2930, 3058, 3136,
    3455, 3479, 3491, 3525, 3547, 3555
\l_zrefclever_reffont_tl . . . .
  . . . . . 2736, 2902,
    3615, 3632, 3674, 3683, 3721, 3749
\c_zrefclever_rf_opts_bool_-
  maybe_type_specific_seq . . . .
  . . . . . 36, 403, 1029, 1765, 1916, 2317
\c_zrefclever_rf_opts_seq_-
  refbounds_seq . . . . .
  . . . . . 403, 945, 1731, 1876, 2233
\c_zrefclever_rf_opts_tl_font_-
  seq . . . . . 403
\c_zrefclever_rf_opts_tl_maybe_-
  type_specific_seq . 403, 863, 2102
\c_zrefclever_rf_opts_tl_not_-
  type_specific_seq . . . . .
  . . . . . 403, 835, 1810, 2068
\c_zrefclever_rf_opts_tl_-
  reference_seq . . . . . 403, 1683
\c_zrefclever_rf_opts_tl_type_-
  names_seq . . . . . 403, 900, 2181
\c_zrefclever_rf_opts_tl_-
  typesetup_seq . . . . . 403, 1822
\l_zrefclever_setup_language_tl
  . . . . . 398, 541, 568, 578, 589,
    715, 767, 774, 788, 807, 813, 822,
    849, 877, 889, 920, 933, 959, 975,
    994, 1011, 1041, 1048, 1057, 1065,
    1079, 1086, 1095, 1103, 1959, 2000,
    2007, 2021, 2035, 2050, 2056, 2082,
    2089, 2116, 2123, 2134, 2143, 2199,
    2210, 2220, 2247, 2265, 2283, 2301,
    2329, 2337, 2350, 2358, 2371, 2379
\l_zrefclever_setup_type_tl . . .
  . . . . . 398, 716, 758,
    759, 791, 814, 823, 842, 870, 890,
    907, 921, 934, 952, 995, 1012, 1036,

```

1058, 1066, 1074, 1096, 1104, 1806,
 1834, 1841, 1888, 1902, 1926, 1935,
 1944, 1958, 1991, 1992, 2024, 2036,
 2057, 2075, 2109, 2135, 2144, 2188,
 2200, 2211, 2221, 2240, 2284, 2302,
 2324, 2338, 2345, 2359, 2366, 2380
 \l_zrefclever_sort_decided_bool
 2452, 2535, 2549, 2559,
 2563, 2575, 2585, 2600, 2615, 2639
 _zrefclever_sort_default:nn ...
 66, 2493, 2509
 _zrefclever_sort_default_-
 different_types:nn
 35, 65, 69, 2520, 2652
 _zrefclever_sort_default_same_-
 type:nn 64, 67, 2519, 2522
 _zrefclever_sort_labels:
 65, 66, 70, 2407, 2456
 _zrefclever_sort_page:nn
 70, 2492, 2704
 \l_zrefclever_sort_prior_a_int .
 2453,
 2654, 2660, 2661, 2667, 2677, 2685
 \l_zrefclever_sort_prior_b_int .
 2453,
 2655, 2662, 2663, 2670, 2678, 2686
 \l_zrefclever_tlastsep_tl
 2736, 2789, 3352
 \l_zrefclever_tlistsep_tl
 2736, 2785, 3323
 \l_zrefclever_tpairsep_tl
 2736, 2781, 3345
 \l_zrefclever_type_count_int ...
 .. 72, 94, 2716, 2771, 3320, 3322,
 3335, 3360, 3374, 3785, 3797, 3992
 \l_zrefclever_type_first_label_-
 tl 71, 91, 2719, 2767, 2971, 3203,
 3212, 3216, 3271, 3289, 3293, 3369,
 3401, 3647, 3653, 3661, 3665, 3678,
 3711, 3728, 3732, 3738, 3753, 3766
 \l_zrefclever_type_first_label_-
 type_tl ... 72, 94, 2719, 2768,
 2973, 3207, 3370, 3403, 3773, 3816,
 3832, 3840, 3852, 3858, 3875, 3891,
 3901, 3909, 3918, 3940, 3950, 3962
 \l_zrefclever_type_first_-
 refbounds_seq
 ... 2748, 2975, 2986, 2997, 3047,
 3083, 3119, 3153, 3170, 3229, 3240,
 3251, 3272, 3467, 3513, 3535, 3564,
 3675, 3676, 3680, 3684, 3723, 3736,
 3741, 3744, 3750, 3751, 3755, 3756
 \l_zrefclever_type_first_-
 refbounds_set_bool 2748,
 2775, 2977, 2988, 2999, 3050, 3086,
 3122, 3156, 3173, 3231, 3242,
 3249, 3377, 3470, 3516, 3538, 3567
 \l_zrefclever_type_name_gender_-
 seq ... 2725, 3842, 3844, 3847, 3862
 \l_zrefclever_type_name_-
 missing_bool
 .. 2725, 3697, 3769, 3776, 3898, 3958
 _zrefclever_type_name_setup: ..
 16, 18, 91, 3258, 3764
 \l_zrefclever_type_name_tl
 91, 94,
 2725, 3296, 3302, 3670, 3690, 3705,
 3707, 3768, 3775, 3879, 3895, 3897,
 3913, 3922, 3944, 3954, 3956, 3976
 \l_zrefclever_typeset_compress_-
 bool 1210, 1213, 3386
 \l_zrefclever_typeset_labels_-
 seq 71, 2713, 2763, 2795, 2797, 2803
 \l_zrefclever_typeset_last_bool
 71, 2713,
 2792, 2793, 2800, 2825, 3332, 3991
 \l_zrefclever_typeset_name_bool
 .. 1159, 1166, 1171, 1176, 3260, 3276
 \l_zrefclever_typeset_queue_-
 curr_tl 72, 74, 90,
 94, 2719, 2766, 2991, 3014, 3022,
 3040, 3053, 3092, 3101, 3123, 3131,
 3138, 3162, 3176, 3185, 3201, 3223,
 3234, 3262, 3269, 3279, 3313, 3329,
 3340, 3346, 3353, 3367, 3368, 3451,
 3474, 3486, 3520, 3542, 3551, 3571,
 3584, 3790, 3812, 3823, 3986, 3990
 \l_zrefclever_typeset_queue_-
 prev_tl . 72, 2719, 2765, 3324, 3366
 \l_zrefclever_typeset_range_-
 bool 1219, 1222, 2406, 3199
 \l_zrefclever_typeset_ref_bool .
 .. 1158, 1165, 1170, 1175, 3260, 3266
 _zrefclever_typeset_refs:
 71–73, 2410, 2761
 _zrefclever_typeset_refs_last_-
 of_type: . 78, 90, 91, 94, 2956, 2961
 _zrefclever_typeset_refs_not_-
 last_of_type:
 72, 78, 90, 99, 2958, 3380
 \l_zrefclever_typeset_sort_bool
 1186, 1189, 2405
 \l_zrefclever_typesort_seq
 . 35, 69, 1195, 1200, 1201, 1207, 2656
 \l_zrefclever_verbose_testing_-
 bool 2760, 3328
 _zrefclever_zcref:nnn
 22, 40, 2391, 2392

_zrefclever_zcref:nnnn	62, 65, <u>2392</u>	_zrefclever_zcsetup:n
\l_zrefclever_zcref_labels_seq	50, 1800, <u>1801</u> , 4189,
.....	65, 66, 2396,	4213, 4219, 4227, 4249, 4268, 4318,	
2427, 2432, <u>2436</u> , 2461, 2464, 2764		4322, 4323, 4332, 4386, 4417, 4466,	
\l_zrefclever_zcref_note_tl	...	4486, 4495, 4507, 4522, 4542, 4565	
.....	1510, 1513, 2412, 2419	\l_zrefclever_zrefcheck_-	
\l_zrefclever_zcref_with_check_-		available_bool
bool 1517, 1532, 2402, 2423	1516, 1527, 1539, 2401, 2422