

The zref-clever package^{*}

Code documentation

Gustavo Barros[†]

2022-02-07

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	38
5	Configuration	62
5.1	\zcsetup	62
5.2	\zcRefTypeSetup	63
5.3	\zcLanguageSetup	68
6	User interface	81
6.1	\zceref	81
6.2	\zcpageref	82
7	Sorting	83
8	Typesetting	89

^{*}This file describes v0.2.1-alpha, released 2022-02-07.

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

9	Compatibility	124
9.1	appendix	124
9.2	appendices	125
9.3	memoir	126
9.4	KOMA	128
9.5	amsmath	129
9.6	mathtools	132
9.7	breqn	133
9.8	listings	134
9.9	enumitem	134
9.10	subcaption	135
9.11	subfig	136
10	Language files	136
10.1	English	136
10.2	German	140
10.3	French	149
10.4	Portuguese	153
10.5	Spanish	157
10.6	Dutch	162
	Index	166

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} {2022-02-07} {0.2.1-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 `\cl@<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 `\cl@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\cl@<counter>` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`__zrefclever_get_enclosing_counters_value:n`

Recursively generate a *sequence* of “enclosing counters” values, for a given `<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 } { endrange-property-undefined }
211 {
212   Option~'endrange=#1'~requested~\msg_line_context:~
213   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
214 }
215 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
216 {

```

```

217 Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:..~
218 To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
219 '\iow_char:N\zceref'.
220 }
221 \msg_new:nnn { zref-clever } { missing-hyperref }
222 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
223 \msg_new:nnn { zref-clever } { titleref-preamble-only }
224 {
225   Option~'titleref'~only~available~in~the~preamble~\msg_line_context:..~
226   Did~you~mean~'ref=title'?
227 }
228 \msg_new:nnn { zref-clever } { option-preamble-only }
229 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:.. }
230 \msg_new:nnn { zref-clever } { unknown-compat-module }
231 {
232   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
233   Nothing~to~do.
234 }
235 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
236 {
237   The~value~of~option~'#1'~must~be~a~comma~separated~list~
238   of~four~items.~We~received~'#2'~items~\msg_line_context:..~
239   Option~not~set.
240 }
241 \msg_new:nnn { zref-clever } { missing-zref-check }
242 {
243   Option~'check'~requested~\msg_line_context:..~
244   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
245 }
246 \msg_new:nnn { zref-clever } { missing-type }
247 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
248 \msg_new:nnn { zref-clever } { missing-property }
249 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:.. }
250 \msg_new:nnn { zref-clever } { missing-name }
251 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:.. }
252 \msg_new:nnn { zref-clever } { single-element-range }
253 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
254 \msg_new:nnn { zref-clever } { compat-package }
255 { Loaded~support~for~'#1'~package. }
256 \msg_new:nnn { zref-clever } { compat-class }
257 { Loaded~support~for~'#1'~documentclass. }
258 \msg_new:nnn { zref-clever } { option-deprecated }
259 {
260   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
261   Use~'#2'~instead.
262 }

```

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}

```

```

263 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
264 {
265   \exp_args:NNo \exp_args:NNo \tl_set:Nn #1
266   { \zref@extractdefault {#2} {#3} {#4} }
267 }
268 \cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

(End definition for __zrefclever_extract_default:Nnnn.)

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

```

\__zrefclever_extract_unexp:nnn{\langle label \rangle}{\langle prop \rangle}{\langle default \rangle}

269 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
270 {
271   \exp_args:NNo \exp_args:No
272   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
273 }
274 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End definition for __zrefclever_extract_unexp:nnn.)

__zrefclever_extract:nnn An internal version for \zref@extractdefault.

```

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

275 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
276 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for __zrefclever_extract:nnn.)

4.4 Option infra

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

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

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

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

```
277 \cs_new:Npn \_zrefclever_opt_varname_general:nn #1#2
```

```
278 { l\_zrefclever_opt_general_ #1 _ #2 }
```

(End definition for `_zrefclever_opt_varname_general:nn`.)

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

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

```
279 \cs_new:Npn \_zrefclever_opt_varname_type:nnn #1#2#3
```

```
280 { l\_zrefclever_opt_type_ #1 _ #2 _ #3 }
```

```
281 \cs_generate_variant:Nn \_zrefclever_opt_varname_type:nnn { enn , een }
```

(End definition for `_zrefclever_opt_varname_type:nnn`.)

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

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

```
282 \cs_new:Npn \_zrefclever_opt_varname_language:nnn #1#2#3
```

```
283 {
```

```
284   \_zrefclever_language_if_declared:nTF {#1}
```

```
285   {
```

```
286     g\_zrefclever_opt_language_
```

```
287     \tl_use:c { \_zrefclever_language_varname:n {#1} }
```

```
288     _ #2 _ #3
```

```
289   }
```

```
290   { g\_zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
```

```
291 }
```

```
292 \cs_generate_variant:Nn \_zrefclever_opt_varname_language:nnn { enn }
```

(End definition for `_zrefclever_opt_varname_language:nnn`.)

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

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

```
293 \cs_new:Npn \_zrefclever_opt_varname_lang_default:nnn #1#2#3
```

```
294 {
```

```
295   \_zrefclever_language_if_declared:nTF {#1}
```

```
296   {
```

```
297     g\_zrefclever_opt_lang_
```

```
298     \tl_use:c { \_zrefclever_language_varname:n {#1} }
```

```
299     _default_ #2 _ #3
```

```

300     }
301     { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
302   }
303   \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}

304   \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
305   {
306     \__zrefclever_language_if_declared:nTF {#1}
307     {
308       g__zrefclever_opt_lang_
309       \tl_use:c { \__zrefclever_language_varname:n {#1} }
310       _type_ #2 _ #3 _ #4
311     }
312     { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
313   }
314   \cs_generate_variant:Nn
315   \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }

```

(End definition for __zrefclever_opt_varname_lang_type:nnnn.)

__zrefclever_opt_varname_fallback:nn

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

```

    \__zrefclever_opt_varname_fallback:nn {\langle option \rangle} {\langle data type \rangle}

316   \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
317   { 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}

318   \cs_new_protected:Npn \__zrefclever_opt_tl_unset:N #1
319   { \tl_set_eq:NN #1 \c_novalue_tl }
320   \cs_new_protected:Npn \__zrefclever_opt_tl_gunset:N #1
321   { \tl_gset_eq:NN #1 \c_novalue_tl }
322   \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }
323   \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:N $\text{TF}$ 
\__zrefclever_opt_tl_if_set:N(TF) {\langle option tl \rangle} {\langle true \rangle} {\langle false \rangle}
324 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
325 {
326   \bool_lazy_and:nnTF
327     { \tl_if_exist_p:N #1 }
328     { ! \tl_if_novalue_p:n {#1} }
329     { \prg_return_true: }
330     { \prg_return_false: }
331 }

(End definition for \__zrefclever_opt_tl_if_set:N $\text{TF}$ .)

\__zrefclever_opt_tl_gset_if_new:Nn
\__zrefclever_opt_tl_gset_if_new:Nn {\langle option tl \rangle} {\langle value \rangle}
332 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
333 {
334   \__zrefclever_opt_tl_if_set:N $\text{F}$  #1
335   { \tl_gset:Nn #1 {#2} }
336 }
337 \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:N $\text{NTF}$ 
\__zrefclever_opt_tl_get:N $\text{NTF}$  {\langle option tl to get \rangle} {\langle tl var to set \rangle}
{\langle true \rangle} {\langle false \rangle}
338 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:N $\text{NTF}$  #1#2 { F }
339 {
340   \__zrefclever_opt_tl_if_set:N $\text{TF}$  #1
341   {
342     \tl_set_eq:Nn #2 #1
343     \prg_return_true:
344   }
345   { \prg_return_false: }
346 }
347 \prg_generate_conditional_variant:Nnn
348   \__zrefclever_opt_tl_get:N $\text{NTF}$  { cn } { F }

(End definition for \__zrefclever_opt_tl_get:N $\text{NTF}$ .)

\__zrefclever_opt_seq_set_clist_split:Nn
\__zrefclever_opt_seq_gset_clist_split:Nn {\langle option seq \rangle} {\langle value \rangle}
\__zrefclever_opt_seq_gset_clist_split:Nn {\langle option seq \rangle} {\langle value \rangle}
349 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
350 { \seq_set_split:Nnn #1 { , } {#2} }
351 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
352 { \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
\__zrefclever_opt_seq_gunset:N
Unset  $\langle option seq \rangle$ . 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 {\langle option seq \rangle}
\__zrefclever_opt_seq_gunset:N {\langle option seq \rangle}

```

```

353 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
354 { \cs_set_eq:NN #1 \scan_stop: }
355 \cs_new_protected:Npn \__zrefclever_opt_seq_gunset:N #1
356 { \cs_gset_eq:NN #1 \scan_stop: }
357 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }
358 \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 $\text{TF}$  \__zrefclever_opt_seq_if_set:N(TF) {\langle option seq \rangle} {\langle true \rangle} {\langle false \rangle}
359 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
360 { \seq_if_exist:NTF #1 { \prg_return_true: } { \prg_return_false: } }
361 \prg_generate_conditional_variant:Nnn
362 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

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

```

```

\__zrefclever_opt_seq_get:N $\text{NTF}$  \__zrefclever_opt_seq_get:N(TF) {\langle option seq to get \rangle} {\langle seq var to set \rangle}
{\langle true \rangle} {\langle false \rangle}
363 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:N #1#2 { F }
364 {
365 \__zrefclever_opt_seq_if_set:N $\text{TF}$  #1
366 {
367 \seq_set_eq:NN #2 #1
368 \prg_return_true:
369 }
370 { \prg_return_false: }
371 }
372 \prg_generate_conditional_variant:Nnn
373 \__zrefclever_opt_seq_get:N { cN } { F }

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

```

$_zrefclever_opt_bool_unset:N$ $_zrefclever_opt_bool_gunset:N$ 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}
374 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
375 { \cs_set_eq:NN #1 \scan_stop: }
376 \cs_new_protected:Npn \__zrefclever_opt_bool_gunset:N #1
377 { \cs_gset_eq:NN #1 \scan_stop: }
378 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }
379 \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 $\text{TF}$  \__zrefclever_opt_bool_if_set:N(TF) {\langle option bool \rangle} {\langle true \rangle} {\langle false \rangle}
380 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
381 { \bool_if_exist:N $\text{TF}$  #1 { \prg_return_true: } { \prg_return_false: } }
382 \prg_generate_conditional_variant:Nnn
383 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End definition for `_zrefclever_opt_bool_if_set:NTF`.)

```

\_zrefclever_opt_bool_get:NTF      \_zrefclever_opt_bool_get:NN(TF) {\option bool to get} {\bool var to set}
                                   {\true} {\false}
384 \prg_new_protected_conditional:Npnn \_zrefclever_opt_bool_get:NN #1#2 { F }
385 {
386   \_zrefclever_opt_bool_if_set:NTF #1
387   {
388     \bool_set_eq:NN #2 #1
389     \prg_return_true:
390   }
391   { \prg_return_false: }
392 }
393 \prg_generate_conditional_variant:Nnn
394 \_zrefclever_opt_bool_get:NN { cN } { F }

```

(End definition for `_zrefclever_opt_bool_get:NTTF`.)

```

\_zrefclever_opt_bool_if:NTF      \_zrefclever_opt_bool_if:N(TF) {\option bool} {\true} {\false}
395 \prg_new_conditional:Npnn \_zrefclever_opt_bool_if:N #1 { T , F , TF }
396 {
397   \_zrefclever_opt_bool_if_set:NTF #1
398   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
399   { \prg_return_false: }
400 }
401 \prg_generate_conditional_variant:Nnn
402 \_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>. However, Joseph Wright seems to particularly dislike this use and the general idea of a “key with no value” being somehow meaningful for `\lkeys` (e.g. his comments on the previous question, and https://tex.stackexchange.com/q/632157/#comment1576404_632157), which does make it somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the “key with no value” is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value “unset” for this purpose. And similarly for “choice” options. 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_type_tl
  \l__zrefclever_setup_language_tl
  \l__zrefclever_lang_decl_case_tl
\l__zrefclever_lang_declension_seq
  \l__zrefclever_lang_gender_seq
403 \tl_new:N \l__zrefclever_setup_type_tl
404 \tl_new:N \l__zrefclever_setup_language_tl
405 \tl_new:N \l__zrefclever_lang_decl_case_tl
406 \seq_new:N \l__zrefclever_lang_declension_seq
407 \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_bool_maybe_type_specific_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
408 \seq_const_from_clist:Nn
409   \c__zrefclever_rf_opts_tl_not_type_specific_seq
410   {
411     tpairsep ,
412     tlistsep ,
413     tlastsep ,
414     notesep ,
415   }
416 \seq_const_from_clist:Nn
417   \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
418   {
419     namesep ,
420     pairsep ,
421     listsep ,
422     lastsep ,
423     rangesep ,
424     namefont ,
425     reffont ,
426   }
427 \seq_const_from_clist:Nn
428   \c__zrefclever_rf_opts_seq_refbounds_seq
429   {
430     refbounds-first ,
431     refbounds-first-sg ,
432     refbounds-first-pb ,

```

```

433     refbounds-first-rb ,
434     refbounds-mid ,
435     refbounds-mid-rb ,
436     refbounds-mid-re ,
437     refbounds-last ,
438     refbounds-last-pe ,
439     refbounds-last-re ,
440 }
441 \seq_const_from_clist:Nn
442 \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
443 {
444     cap ,
445     abbrev ,
446     rangetopair ,
447 }

```

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:.`

```

448 \seq_const_from_clist:Nn
449 \c__zrefclever_rf_opts_tl_type_names_seq
450 {
451     Name-sg ,
452     name-sg ,
453     Name-pl ,
454     name-pl ,
455     Name-sg-ab ,
456     name-sg-ab ,
457     Name-pl-ab ,
458     name-pl-ab ,
459 }

```

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

```

460 \seq_new:N \c__zrefclever_rf_opts_tl_typesetup_seq
461 \seq_gconcat:NNN \c__zrefclever_rf_opts_tl_typesetup_seq
462 \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
463 \c__zrefclever_rf_opts_tl_type_names_seq
464 \seq_new:N \c__zrefclever_rf_opts_tl_reference_seq
465 \seq_gconcat:NNN \c__zrefclever_rf_opts_tl_reference_seq
466 \c__zrefclever_rf_opts_tl_not_type_specific_seq
467 \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.

```

468 \clist_map_inline:nn
469 {
470     reference ,
471     typesetup ,
472     langsetup ,
473     langfile ,
474 }
475 {
476     \keys_define:nn { zref-clever / #1 }

```

```

477 {
478   +refbounds-first .meta:n =
479   {
480     refbounds-first = {##1} ,
481     refbounds-first-sg = {##1} ,
482     refbounds-first-pb = {##1} ,
483     refbounds-first-rb = {##1} ,
484   } ,
485   +refbounds-first .default:x = \c_novalue_tl ,
486   +refbounds-mid .meta:n =
487   {
488     refbounds-mid = {##1} ,
489     refbounds-mid-rb = {##1} ,
490     refbounds-mid-re = {##1} ,
491   } ,
492   +refbounds-mid .default:x = \c_novalue_tl ,
493   +refbounds-last .meta:n =
494   {
495     refbounds-last = {##1} ,
496     refbounds-last-pe = {##1} ,
497     refbounds-last-re = {##1} ,
498   } ,
499   +refbounds-last .default:x = \c_novalue_tl ,
500   +refbounds-rb .meta:n =
501   {
502     refbounds-first-rb = {##1} ,
503     refbounds-mid-rb = {##1} ,
504   } ,
505   +refbounds-rb .default:x = \c_novalue_tl ,
506   +refbounds-re .meta:n =
507   {
508     refbounds-mid-re = {##1} ,
509     refbounds-last-re = {##1} ,
510   } ,
511   +refbounds-re .default:x = \c_novalue_tl ,
512   +refbounds .meta:n =
513   {
514     +refbounds-first = {##1} ,
515     +refbounds-mid = {##1} ,
516     +refbounds-last = {##1} ,
517   } ,
518   +refbounds .default:x = \c_novalue_tl ,
519   refbounds .meta:n = { +refbounds = {##1} } ,
520   refbounds .default:x = \c_novalue_tl ,
521 }
522 }

```

4.6 Languages

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

```
\_zrefclever_language_varname:n {\language}
```

```

523 \cs_new:Npn \__zrefclever_language_varname:n #1
524 { 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.

```

525 \cs_set_eq:NN \zrefclever_language_varname:n
526 \__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>}

```

```

527 \prg_new_conditional:Npnn \__zrefclever_language_if_declared:n #1 { T , F , TF }
528 {
529   \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
530   { \prg_return_true: }
531   { \prg_return_false: }
532 }
533 \prg_generate_conditional_variant:Nnn
534 \__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.

```

535 \prg_set_eq_conditional:Nnn \zrefclever_language_if_declared:n
536 \__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>}

```

```

537 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
538 {
539   \group_begin:
540   \tl_if_empty:nF {#2}
541   {
542     \__zrefclever_language_if_declared:nTF {#2}
543     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
544     {
545       \tl_new:c { \__zrefclever_language_varname:n {#2} }
546       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
547       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
548       \keys_set:nn { zref-clever/declarelang } {#1}
549     }
550   }
551   \group_end:
552 }
553 \@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}

554 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
555 {
556   \tl_if_empty:nF {#1}
557   {
558     \__zrefclever_language_if_declared:nTF {#2}
559     {
560       \tl_gset:cx { \__zrefclever_language_varname:n {#1} }
561       { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
562     }
563     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
564   }
565 }
566 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

567 \keys_define:nn { zref-clever/declarelang }
568 {
569   declension .code:n =
570   {
571     \seq_gset_from_clist:cn
572     {
573       \__zrefclever_opt_varname_language:enn
574       { \l__zrefclever_setup_language_tl } { declension } { seq }
575     }
576     {#1}
577   } ,
578   declension .value_required:n = true ,
579   gender .code:n =
580   {

```

```

581     \seq_gset_from_clist:cn
582     {
583         \__zrefclever_opt_varname_language:enn
584         { \l__zrefclever_setup_language_tl } { gender } { seq }
585     }
586     {#1}
587 },
588 gender .value_required:n = true ,
589 allcaps .choices:nn =
590 { true , false }
591 {
592     \use:c { bool_gset_ \l_keys_choice_tl :c }
593     {
594         \__zrefclever_opt_varname_language:enn
595         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
596     }
597 },
598 allcaps .default:n = true ,
599 }

```

`__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.

```

600 \cs_new_protected:Npn \__zrefclever_process_language_settings:
601 {
602     \__zrefclever_language_if_declared:xTF
603     { \l__zrefclever_ref_language_tl }
604     {

```

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.

```

605     \__zrefclever_opt_seq_get:cNF
606     {
607         \__zrefclever_opt_varname_language:enn
608         { \l__zrefclever_ref_language_tl } { declension } { seq }
609     }
610     \l__zrefclever_lang_declension_seq
611     { \seq_clear:N \l__zrefclever_lang_declension_seq }
612     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
613     {
614         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
615         {
616             \msg_warning:nxxx { zref-clever }
617             { language-no-decl-ref }

```

```

618         { \l__zrefclever_ref_language_tl }
619         { \l__zrefclever_ref_decl_case_tl }
620         \tl_clear:N \l__zrefclever_ref_decl_case_tl
621     }
622 }
623 {
624     \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
625     {
626         \seq_get_left:NN \l__zrefclever_lang_declension_seq
627         \l__zrefclever_ref_decl_case_tl
628     }
629     {
630         \seq_if_in:NVF \l__zrefclever_lang_declension_seq
631         \l__zrefclever_ref_decl_case_tl
632         {
633             \msg_warning:nxxx { zref-clever }
634             { unknown-decl-case }
635             { \l__zrefclever_ref_decl_case_tl }
636             { \l__zrefclever_ref_language_tl }
637             \seq_get_left:NN \l__zrefclever_lang_declension_seq
638             \l__zrefclever_ref_decl_case_tl
639         }
640     }
641 }

```

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.

```

642     \__zrefclever_opt_seq_get:cNF
643     {
644         \__zrefclever_opt_varname_language:enn
645         { \l__zrefclever_ref_language_tl } { gender } { seq }
646     }
647     \l__zrefclever_lang_gender_seq
648     { \seq_clear:N \l__zrefclever_lang_gender_seq }
649     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
650     {
651         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
652         {
653             \msg_warning:nxxxx { zref-clever }
654             { language-no-gender }
655             { \l__zrefclever_ref_language_tl }
656             { g }
657             { \l__zrefclever_ref_gender_tl }
658             \tl_clear:N \l__zrefclever_ref_gender_tl
659         }
660     }
661     {
662         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
663         {
664             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
665             \l__zrefclever_ref_gender_tl
666             {
667                 \msg_warning:nxxx { zref-clever }

```

```

668         { gender-not-declared }
669         { \l__zrefclever_ref_language_tl }
670         { \l__zrefclever_ref_gender_tl }
671         \tl_clear:N \l__zrefclever_ref_gender_tl
672     }
673 }
674 }

```

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

```

675 \__zrefclever_opt_bool_if:cT
676 {
677     \__zrefclever_opt_varname_language:enn
678     { \l__zrefclever_ref_language_tl } { allcaps } { bool }
679 }
680 { \keys_set:nn { zref-clever/reference } { cap = true } }
681 }
682 {

```

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

```

683 \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
684 {
685     \msg_warning:nxxx { zref-clever } { unknown-language-decl }
686     { \l__zrefclever_ref_decl_case_tl }
687     { \l__zrefclever_ref_language_tl }
688     \tl_clear:N \l__zrefclever_ref_decl_case_tl
689 }
690 \tl_if_empty:NF \l__zrefclever_ref_gender_tl
691 {
692     \msg_warning:nxxxx { zref-clever }
693     { language-no-gender }
694     { \l__zrefclever_ref_language_tl }
695     { g }
696     { \l__zrefclever_ref_gender_tl }
697     \tl_clear:N \l__zrefclever_ref_gender_tl
698 }
699 }
700 }

```

(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 `\begin{document}`. 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.

```
701 \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 {\language}

702 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
703 {
704   \group_begin:
705   \@bsphack
706   \__zrefclever_language_if_declared:nT {#1}
707   {
708     \seq_if_in:NxF
709     \g__zrefclever_loaded_langfiles_seq
710     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
711     {
712       \exp_args:Nx \file_get:nnNTF
713       {
714         zref-clever-
715         \tl_use:c { \__zrefclever_language_varname:n {#1} }

```

```

716     .lang
717 }
718 { \ExplSyntaxOn }
719 \l_tmpa_tl
720 {
721   \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
722   \tl_clear:N \l__zrefclever_setup_type_tl
723   \__zrefclever_opt_seq_get:cNF
724   {
725     \__zrefclever_opt_varname_language:nnn
726     {#1} { declension } { seq }
727   }
728   \l__zrefclever_lang_declension_seq
729   { \seq_clear:N \l__zrefclever_lang_declension_seq }
730   \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
731   { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
732   {
733     \seq_get_left:NN \l__zrefclever_lang_declension_seq
734     \l__zrefclever_lang_decl_case_tl
735   }
736   \__zrefclever_opt_seq_get:cNF
737   {
738     \__zrefclever_opt_varname_language:nnn
739     {#1} { gender } { seq }
740   }
741   \l__zrefclever_lang_gender_seq
742   { \seq_clear:N \l__zrefclever_lang_gender_seq }
743   \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
744   \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
745   { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
746   \msg_info:nnx { zref-clever } { langfile-loaded }
747   { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
748 }
749 {

```

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.

```

750   \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
751   { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
752 }
753 }
754 }
755 \@esphack
756 \group_end:
757 }
758 \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.

```

759 \keys_define:nn { zref-clever/langfile }
760 {
761   type .code:n =
762   {
763     \tl_if_empty:NTF {#1}
764     { \tl_clear:N \l__zrefclever_setup_type_tl }
765     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
766   } ,
767
768   case .code:n =
769   {
770     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
771     {
772       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
773       { \l__zrefclever_setup_language_tl } {#1}
774     }
775     {
776       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
777       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
778       {
779         \msg_info:nxxx { zref-clever } { unknown-decl-case }
780         {#1} { \l__zrefclever_setup_language_tl }
781         \seq_get_left:NN \l__zrefclever_lang_declension_seq
782         \l__zrefclever_lang_decl_case_tl
783       }
784     }
785   } ,
786   case .value_required:n = true ,
787
788   gender .default:x = \c_novalue_tl ,
789   gender .code:n =
790   {
791     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
792     {
793       \msg_info:nxxxx { zref-clever } { language-no-gender }
794       { \l__zrefclever_setup_language_tl } { gender } {#1}
795     }
796     {
797       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
798       {
799         \msg_info:nnn { zref-clever }
800         { option-only-type-specific } { gender }
801       }
802       {
803         \tl_if_novalue:nF {#1}
804         {
805           \seq_clear:N \l_tmpa_seq
806           \clist_map_inline:nn {#1}
807           {
808             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
809             { \seq_put_right:Nn \l_tmpa_seq {##1} }
810             {
811               \msg_info:nxxx { zref-clever }
812               { gender-not-declared }

```

```

813         { \l__zrefclever_setup_language_tl } {##1}
814     }
815 }
816 \__zrefclever_opt_seq_if_set:cF
817 {
818     \__zrefclever_opt_varname_lang_type:enn
819     { \l__zrefclever_setup_language_tl }
820     { \l__zrefclever_setup_type_tl }
821     { gender }
822     { seq }
823 }
824 {
825     \seq_gset_eq:cN
826     {
827         \__zrefclever_opt_varname_lang_type:enn
828         { \l__zrefclever_setup_language_tl }
829         { \l__zrefclever_setup_type_tl }
830         { gender }
831         { seq }
832     }
833     \l_tmpa_seq
834 }
835 }
836 }
837 } ,
838 } ,
839 }
840 \seq_map_inline:Nn
841 \c__zrefclever_rf_opts_tl_not_type_specific_seq
842 {
843     \keys_define:nn { zref-clever/langfile }
844     {
845         #1 .default:x = \c_novalue_tl ,
846         #1 .code:n =
847         {
848             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
849             {
850                 \tl_if_novalue:nF {##1}
851                 {
852                     \__zrefclever_opt_tl_gset_if_new:cn
853                     {
854                         \__zrefclever_opt_varname_lang_default:enn
855                         { \l__zrefclever_setup_language_tl }
856                         {#1} { tl }
857                     }
858                     {##1}
859                 }
860             }
861             {
862                 \msg_info:nnn { zref-clever }
863                 { option-not-type-specific } {#1}
864             }
865         } ,
866     }

```

```

867 }
868 \seq_map_inline:Nn
869 \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
870 {
871   \keys_define:nn { zref-clever/langfile }
872   {
873     #1 .default:x = \c_novalue_tl ,
874     #1 .code:n =
875     {
876       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
877       {
878         \tl_if_novalue:nF {##1}
879         {
880           \__zrefclever_opt_tl_gset_if_new:cn
881           {
882             \__zrefclever_opt_varname_lang_default:enn
883             { \l__zrefclever_setup_language_tl }
884             {#1} { tl }
885           }
886           {##1}
887         }
888       }
889       {
890         \tl_if_novalue:nF {##1}
891         {
892           \__zrefclever_opt_tl_gset_if_new:cn
893           {
894             \__zrefclever_opt_varname_lang_type:enn
895             { \l__zrefclever_setup_language_tl }
896             { \l__zrefclever_setup_type_tl }
897             {#1} { tl }
898           }
899           {##1}
900         }
901       }
902     } ,
903   }
904 }
905 \keys_define:nn { zref-clever/langfile }
906 {
907   endrange .code:n =
908   {
909     \str_case:nnF {#1}
910     {
911       { ref }
912       {
913         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
914         {
915           \__zrefclever_opt_tl_gset_if_new:cn
916           {
917             \__zrefclever_opt_varname_lang_default:enn
918             { \l__zrefclever_setup_language_tl }
919             { endrangefunc } { tl }
920           }

```

```

921         { }
922     \_zrefclever_opt_tl_gset_if_new:cn
923     {
924         \_zrefclever_opt_varname_lang_default:enn
925         { \l__zrefclever_setup_language_tl }
926         { endrangeprop } { tl }
927     }
928     { }
929 }
930 {
931     \_zrefclever_opt_tl_gset_if_new:cn
932     {
933         \_zrefclever_opt_varname_lang_type:enn
934         { \l__zrefclever_setup_language_tl }
935         { \l__zrefclever_setup_type_tl }
936         { endrangefunc } { tl }
937     }
938     { }
939     \_zrefclever_opt_tl_gset_if_new:cn
940     {
941         \_zrefclever_opt_varname_lang_type:enn
942         { \l__zrefclever_setup_language_tl }
943         { \l__zrefclever_setup_type_tl }
944         { endrangeprop } { tl }
945     }
946     { }
947 }
948 }
949
950 { stripprefix }
951 {
952     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
953     {
954         \_zrefclever_opt_tl_gset_if_new:cn
955         {
956             \_zrefclever_opt_varname_lang_default:enn
957             { \l__zrefclever_setup_language_tl }
958             { endrangefunc } { tl }
959         }
960         { __zrefclever_get_endrange_stripprefix }
961         \_zrefclever_opt_tl_gset_if_new:cn
962         {
963             \_zrefclever_opt_varname_lang_default:enn
964             { \l__zrefclever_setup_language_tl }
965             { endrangeprop } { tl }
966         }
967         { }
968     }
969     {
970         \_zrefclever_opt_tl_gset_if_new:cn
971         {
972             \_zrefclever_opt_varname_lang_type:enn
973             { \l__zrefclever_setup_language_tl }
974             { \l__zrefclever_setup_type_tl }

```

```

975         { endrangefunc } { t1 }
976     }
977     { __zrefclever_get_endrange_stripprefix }
978     \__zrefclever_opt_tl_gset_if_new:cn
979     {
980         \__zrefclever_opt_varname_lang_type:enn
981         { \l__zrefclever_setup_language_tl }
982         { \l__zrefclever_setup_type_tl }
983         { endrangeprop } { t1 }
984     }
985     { }
986 }
987 }
988
989 { pagecomp }
990 {
991     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
992     {
993         \__zrefclever_opt_tl_gset_if_new:cn
994         {
995             \__zrefclever_opt_varname_lang_default:enn
996             { \l__zrefclever_setup_language_tl }
997             { endrangefunc } { t1 }
998         }
999         { __zrefclever_get_endrange_pagecomp }
1000         \__zrefclever_opt_tl_gset_if_new:cn
1001         {
1002             \__zrefclever_opt_varname_lang_default:enn
1003             { \l__zrefclever_setup_language_tl }
1004             { endrangeprop } { t1 }
1005         }
1006         { }
1007     }
1008     {
1009         \__zrefclever_opt_tl_gset_if_new:cn
1010         {
1011             \__zrefclever_opt_varname_lang_type:enn
1012             { \l__zrefclever_setup_language_tl }
1013             { \l__zrefclever_setup_type_tl }
1014             { endrangefunc } { t1 }
1015         }
1016         { __zrefclever_get_endrange_pagecomp }
1017         \__zrefclever_opt_tl_gset_if_new:cn
1018         {
1019             \__zrefclever_opt_varname_lang_type:enn
1020             { \l__zrefclever_setup_language_tl }
1021             { \l__zrefclever_setup_type_tl }
1022             { endrangeprop } { t1 }
1023         }
1024         { }
1025     }
1026 }
1027
1028 { pagecomp2 }

```

```

1029 {
1030   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1031   {
1032     \__zrefclever_opt_tl_gset_if_new:cn
1033     {
1034       \__zrefclever_opt_varname_lang_default:enn
1035       { \l__zrefclever_setup_language_tl }
1036       { endrangefunc } { t1 }
1037     }
1038     { __zrefclever_get_endrange_pagecomptwo }
1039     \__zrefclever_opt_tl_gset_if_new:cn
1040     {
1041       \__zrefclever_opt_varname_lang_default:enn
1042       { \l__zrefclever_setup_language_tl }
1043       { endrangeprop } { t1 }
1044     }
1045     { }
1046   }
1047   {
1048     \__zrefclever_opt_tl_gset_if_new:cn
1049     {
1050       \__zrefclever_opt_varname_lang_type:enn
1051       { \l__zrefclever_setup_language_tl }
1052       { \l__zrefclever_setup_type_tl }
1053       { endrangefunc } { t1 }
1054     }
1055     { __zrefclever_get_endrange_pagecomptwo }
1056     \__zrefclever_opt_tl_gset_if_new:cn
1057     {
1058       \__zrefclever_opt_varname_lang_type:enn
1059       { \l__zrefclever_setup_language_tl }
1060       { \l__zrefclever_setup_type_tl }
1061       { endrangeprop } { t1 }
1062     }
1063     { }
1064   }
1065 }
1066
1067 { unset }
1068 { }
1069 }
1070 {
1071   \tl_if_empty:nTF {#1}
1072   {
1073     \msg_info:nnn { zref-clever }
1074     { endrange-property-undefined } {#1}
1075   }
1076   {
1077     \zref@ifpropundefined {#1}
1078     {
1079       \msg_info:nnn { zref-clever }
1080       { endrange-property-undefined } {#1}
1081     }
1082     {

```



```

1083 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1084 {
1085   \__zrefclever_opt_tl_gset_if_new:cn
1086   {
1087     \__zrefclever_opt_varname_lang_default:enn
1088     { \l__zrefclever_setup_language_tl }
1089     { endrangefunc } { tl }
1090   }
1091   { __zrefclever_get_endrange_property }
1092   \__zrefclever_opt_tl_gset_if_new:cn
1093   {
1094     \__zrefclever_opt_varname_lang_default:enn
1095     { \l__zrefclever_setup_language_tl }
1096     { endrangeprop } { tl }
1097   }
1098   {#1}
1099 }
1100 {
1101   \__zrefclever_opt_tl_gset_if_new:cn
1102   {
1103     \__zrefclever_opt_varname_lang_type:eenn
1104     { \l__zrefclever_setup_language_tl }
1105     { \l__zrefclever_setup_type_tl }
1106     { endrangefunc } { tl }
1107   }
1108   { __zrefclever_get_endrange_property }
1109   \__zrefclever_opt_tl_gset_if_new:cn
1110   {
1111     \__zrefclever_opt_varname_lang_type:eenn
1112     { \l__zrefclever_setup_language_tl }
1113     { \l__zrefclever_setup_type_tl }
1114     { endrangeprop } { tl }
1115   }
1116   {#1}
1117 }
1118 }
1119 }
1120 }
1121 } ,
1122   endrange .value_required:n = true ,
1123 }
1124 \seq_map_inline:Nn
1125   \c__zrefclever_rf_opts_tl_type_names_seq
1126   {
1127     \keys_define:nn { zref-clever/langfile }
1128     {
1129       #1 .default:x = \c_novalue_tl ,
1130       #1 .code:n =
1131       {
1132         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1133         {
1134           \msg_info:nnn { zref-clever }
1135           { option-only-type-specific } {#1}
1136         }

```

```

1137 {
1138   \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1139   {
1140     \tl_if_novalue:nF {##1}
1141     {
1142       \__zrefclever_opt_tl_gset_if_new:cn
1143       {
1144         \__zrefclever_opt_varname_lang_type:een
1145         { \l__zrefclever_setup_language_tl }
1146         { \l__zrefclever_setup_type_tl }
1147         {#1} { tl }
1148       }
1149       {##1}
1150     }
1151   }
1152   {
1153     \tl_if_novalue:nF {##1}
1154     {
1155       \__zrefclever_opt_tl_gset_if_new:cn
1156       {
1157         \__zrefclever_opt_varname_lang_type:een
1158         { \l__zrefclever_setup_language_tl }
1159         { \l__zrefclever_setup_type_tl }
1160         { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
1161       }
1162       {##1}
1163     }
1164   }
1165 } ,
1166 }
1167 }
1168 }
1169 \seq_map_inline:Nn
1170 \c__zrefclever_rf_opts_seq_refbounds_seq
1171 {
1172   \keys_define:nn { zref-clever/langfile }
1173   {
1174     #1 .default:x = \c_novalue_tl ,
1175     #1 .code:n =
1176     {
1177       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1178       {
1179         \tl_if_novalue:nF {##1}
1180         {
1181           \__zrefclever_opt_seq_if_set:cF
1182           {
1183             \__zrefclever_opt_varname_lang_default:enn
1184             { \l__zrefclever_setup_language_tl } {#1} { seq }
1185           }
1186           {
1187             \seq_gclear:N \g_tmpa_seq
1188             \__zrefclever_opt_seq_gset_clist_split:Nn
1189             \g_tmpa_seq {##1}
1190             \bool_lazy_or:nnTF

```

```

1191         { \tl_if_empty_p:n {##1} }
1192     {
1193         \int_compare_p:nNn
1194             { \seq_count:N \g_tmpa_seq } = { 4 }
1195     }
1196     {
1197         \seq_gset_eq:cN
1198             {
1199                 \__zrefclever_opt_varname_lang_default:enn
1200                 { \l__zrefclever_setup_language_tl }
1201                 {#1} { seq }
1202             }
1203         \g_tmpa_seq
1204     }
1205     {
1206         \msg_info:nxxx { zref-clever }
1207         { refbounds-must-be-four }
1208         {#1} { \seq_count:N \g_tmpa_seq }
1209     }
1210 }
1211 }
1212 {
1213 {
1214     \tl_if_novalue:nF {##1}
1215     {
1216         \__zrefclever_opt_seq_if_set:cF
1217         {
1218             \__zrefclever_opt_varname_lang_type:eenn
1219             { \l__zrefclever_setup_language_tl }
1220             { \l__zrefclever_setup_type_tl } {#1} { seq }
1221         }
1222         {
1223             \seq_gclear:N \g_tmpa_seq
1224             \__zrefclever_opt_seq_gset_clist_split:Nn
1225             \g_tmpa_seq {##1}
1226             \bool_lazy_or:nnTF
1227                 { \tl_if_empty_p:n {##1} }
1228                 {
1229                     \int_compare_p:nNn
1230                         { \seq_count:N \g_tmpa_seq } = { 4 }
1231                 }
1232                 {
1233                     \seq_gset_eq:cN
1234                     {
1235                         \__zrefclever_opt_varname_lang_type:eenn
1236                         { \l__zrefclever_setup_language_tl }
1237                         { \l__zrefclever_setup_type_tl }
1238                         {#1} { seq }
1239                     }
1240                     \g_tmpa_seq
1241                 }
1242             {
1243                 \msg_info:nxxx { zref-clever }
1244                 { refbounds-must-be-four }

```

```

1245         {#1} { \seq_count:N \g_tmpa_seq }
1246     }
1247 }
1248 }
1249 } ,
1250 } ,
1251 }
1252 }
1253 \seq_map_inline:Nn
1254 \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
1255 {
1256     \keys_define:nn { zref-clever/langfile }
1257     {
1258         #1 .choice: ,
1259         #1 / true .code:n =
1260         {
1261             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1262             {
1263                 \__zrefclever_opt_bool_if_set:cF
1264                 {
1265                     \__zrefclever_opt_varname_lang_default:enn
1266                     { \l__zrefclever_setup_language_tl }
1267                     {#1} { bool }
1268                 }
1269                 {
1270                     \bool_gset_true:c
1271                     {
1272                         \__zrefclever_opt_varname_lang_default:enn
1273                         { \l__zrefclever_setup_language_tl }
1274                         {#1} { bool }
1275                     }
1276                 }
1277             }
1278             {
1279                 \__zrefclever_opt_bool_if_set:cF
1280                 {
1281                     \__zrefclever_opt_varname_lang_type:eenn
1282                     { \l__zrefclever_setup_language_tl }
1283                     { \l__zrefclever_setup_type_tl }
1284                     {#1} { bool }
1285                 }
1286                 {
1287                     \bool_gset_true:c
1288                     {
1289                         \__zrefclever_opt_varname_lang_type:eenn
1290                         { \l__zrefclever_setup_language_tl }
1291                         { \l__zrefclever_setup_type_tl }
1292                         {#1} { bool }
1293                     }
1294                 }
1295             }
1296         } ,
1297         #1 / false .code:n =
1298         {

```

```

1299 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1300 {
1301   \__zrefclever_opt_bool_if_set:cF
1302   {
1303     \__zrefclever_opt_varname_lang_default:enn
1304     { \l__zrefclever_setup_language_tl }
1305     {#1} { bool }
1306   }
1307   {
1308     \bool_gset_false:c
1309     {
1310       \__zrefclever_opt_varname_lang_default:enn
1311       { \l__zrefclever_setup_language_tl }
1312       {#1} { bool }
1313     }
1314   }
1315 }
1316 {
1317   \__zrefclever_opt_bool_if_set:cF
1318   {
1319     \__zrefclever_opt_varname_lang_type:enn
1320     { \l__zrefclever_setup_language_tl }
1321     { \l__zrefclever_setup_type_tl }
1322     {#1} { bool }
1323   }
1324   {
1325     \bool_gset_false:c
1326     {
1327       \__zrefclever_opt_varname_lang_type:enn
1328       { \l__zrefclever_setup_language_tl }
1329       { \l__zrefclever_setup_type_tl }
1330       {#1} { bool }
1331     }
1332   }
1333 }
1334 } ,
1335 #1 / unset .code:n = { } ,
1336 #1 .default:n = true ,
1337 no #1 .meta:n = { #1 = false } ,
1338 no #1 .value_forbidden:n = true ,
1339 }
1340 }

```

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.

```

1341 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1342 {
1343   \tl_const:cn
1344   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1345 }

```

```

1346 \keyval_parse:nnn
1347 { }
1348 { \__zrefclever_opt_tl_cset_fallback:nn }
1349 {
1350   tpairsep = {,~} ,
1351   tlistsep = {,~} ,
1352   tlastsep = {,~} ,
1353   notesep  = {~} ,
1354   namesep  = {\nobreakspace} ,
1355   pairsep  = {,~} ,
1356   listsep  = {,~} ,
1357   lastsep  = {,~} ,
1358   rangesep = {\textendash} ,
1359 }

```

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>}

1360 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1361 {
1362   \tl_if_empty:nTF {#3}
1363     { \prop_remove:Nn #1 {#2} }
1364     { \prop_put:Nnn #1 {#2} {#3} }
1365 }

```

(End definition for `__zrefclever_prop_put_non_empty:Nnn`.)

ref option

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

```

1366 \tl_new:N \l__zrefclever_ref_property_tl
1367 \keys_define:nn { zref-clever/reference }
1368 {
1369   ref .code:n =
1370   {
1371     \tl_if_empty:nTF {#1}
1372     {
1373       \msg_warning:nnn { zref-clever }
1374       { zref-property-undefined } {#1}
1375       \tl_set:Nn \l__zrefclever_ref_property_tl { default }

```

```

1376     }
1377     {
1378         \zref@ifpropundefined {#1}
1379         {
1380             \msg_warning:nnn { zref-clever }
1381             { zref-property-undefined } {#1}
1382             \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1383         }
1384         { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1385     }
1386 } ,
1387 ref .initial:n = default ,
1388 ref .value_required:n = true ,
1389 page .meta:n = { ref = page },
1390 page .value_forbidden:n = true ,
1391 }

```

typeset option

```

1392 \bool_new:N \l__zrefclever_typeset_ref_bool
1393 \bool_new:N \l__zrefclever_typeset_name_bool
1394 \keys_define:nn { zref-clever/reference }
1395 {
1396     typeset .choice: ,
1397     typeset / both .code:n =
1398     {
1399         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1400         \bool_set_true:N \l__zrefclever_typeset_name_bool
1401     } ,
1402     typeset / ref .code:n =
1403     {
1404         \bool_set_true:N \l__zrefclever_typeset_ref_bool
1405         \bool_set_false:N \l__zrefclever_typeset_name_bool
1406     } ,
1407     typeset / name .code:n =
1408     {
1409         \bool_set_false:N \l__zrefclever_typeset_ref_bool
1410         \bool_set_true:N \l__zrefclever_typeset_name_bool
1411     } ,
1412     typeset .initial:n = both ,
1413     typeset .value_required:n = true ,
1414
1415     noname .meta:n = { typeset = ref } ,
1416     noname .value_forbidden:n = true ,
1417     noref .meta:n = { typeset = name } ,
1418     noref .value_forbidden:n = true ,
1419 }

```

sort option

```

1420 \bool_new:N \l__zrefclever_typeset_sort_bool
1421 \keys_define:nn { zref-clever/reference }
1422 {
1423     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1424     sort .initial:n = true ,

```

```

1425     sort .default:n = true ,
1426     nosort .meta:n = { sort = false },
1427     nosort .value_forbidden:n = true ,
1428 }

```

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`.

```

1429 \seq_new:N \l__zrefclever_typesort_seq
1430 \keys_define:nn { zref-clever/reference }
1431 {
1432     typesort .code:n =
1433     {
1434         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1435         \seq_reverse:N \l__zrefclever_typesort_seq
1436     } ,
1437     typesort .initial:n =
1438     { part , chapter , section , paragraph },
1439     typesort .value_required:n = true ,
1440     notypesort .code:n =
1441     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1442     notypesort .value_forbidden:n = true ,
1443 }

```

comp option

```

1444 \bool_new:N \l__zrefclever_typeset_compress_bool
1445 \keys_define:nn { zref-clever/reference }
1446 {
1447     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1448     comp .initial:n = true ,
1449     comp .default:n = true ,
1450     nocomp .meta:n = { comp = false },
1451     nocomp .value_forbidden:n = true ,
1452 }

```

endrange option

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

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

may set $\langle tl\ var\ to\ set \rangle$ to the special value `zc@missingproperty`, to signal a missing property for `__zrefclever_get_ref_endrange:nnN`.

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

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

```

1453 \NewHook { zref-clever/endrange-setup }
1454 \keys_define:nn { zref-clever/reference }
1455 {
1456   endrange .code:n =
1457   {
1458     \str_case:nnF {#1}
1459     {
1460       { ref }
1461       {
1462         \tl_clear:c
1463         {
1464           \__zrefclever_opt_varname_general:nn
1465             { endrangefunc } { t1 }
1466         }
1467         \tl_clear:c
1468         {
1469           \__zrefclever_opt_varname_general:nn
1470             { endrangeprop } { t1 }
1471         }
1472       }
1473     }
1474   { stripprefix }
1475   {
1476     \tl_set:cn
1477     {
1478       \__zrefclever_opt_varname_general:nn
1479         { endrangefunc } { t1 }
1480     }
1481     { \__zrefclever_get_endrange_stripprefix }
1482     \tl_clear:c
1483     {
1484       \__zrefclever_opt_varname_general:nn
1485         { endrangeprop } { t1 }
1486     }
1487   }

```

```

1487     }
1488
1489     { pagecomp }
1490     {
1491         \tl_set:cn
1492         {
1493             \__zrefclever_opt_varname_general:nn
1494             { endrangefunc } { t1 }
1495         }
1496         { __zrefclever_get_endrange_pagecomp }
1497         \tl_clear:c
1498         {
1499             \__zrefclever_opt_varname_general:nn
1500             { endrangeprop } { t1 }
1501         }
1502     }
1503
1504     { pagecomp2 }
1505     {
1506         \tl_set:cn
1507         {
1508             \__zrefclever_opt_varname_general:nn
1509             { endrangefunc } { t1 }
1510         }
1511         { __zrefclever_get_endrange_pagecomptwo }
1512         \tl_clear:c
1513         {
1514             \__zrefclever_opt_varname_general:nn
1515             { endrangeprop } { t1 }
1516         }
1517     }
1518
1519     { unset }
1520     {
1521         \__zrefclever_opt_t1_unset:c
1522         {
1523             \__zrefclever_opt_varname_general:nn
1524             { endrangefunc } { t1 }
1525         }
1526         \__zrefclever_opt_t1_unset:c
1527         {
1528             \__zrefclever_opt_varname_general:nn
1529             { endrangeprop } { t1 }
1530         }
1531     }
1532 }
1533 {
1534     \tl_if_empty:nTF {#1}
1535     {
1536         \msg_warning:nnn { zref-clever }
1537         { endrange-property-undefined } {#1}
1538     }
1539     {
1540         \zref@ifpropundefined {#1}

```

```

1541         {
1542             \msg_warning:nnn { zref-clever }
1543             { endrange-property-undefined } {#1}
1544         }
1545         {
1546             \tl_set:cn
1547             {
1548                 \__zrefclever_opt_varname_general:nn
1549                 { endrangefunc } { tl }
1550             }
1551             { \__zrefclever_get_endrange_property }
1552             \tl_set:cn
1553             {
1554                 \__zrefclever_opt_varname_general:nn
1555                 { endrangeprop } { tl }
1556             }
1557             {#1}
1558         }
1559     }
1560 }
1561 },
1562 endrange .value_required:n = true ,
1563 }
1564 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1565 {
1566     \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1567     {
1568         \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1569         {
1570             \__zrefclever_extract_default:Nnvn #3
1571             {#2} { \l__zrefclever_ref_property_tl } { }
1572         }
1573         { \tl_set:Nn #3 { zc@missingproperty } }
1574     }
1575     {
1576         \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1577         {

```

If the range came about by normal compression, we already know the beginning and the end references share the same “form” and “prefix” (this is ensured at `__zrefclever_labels_in_sequence:nn`), but the same is not true if the `range` option is being used, in which case, we have to check the replacement `\l__zrefclever_ref_property_tl` by `\l__zrefclever_endrangeprop_tl` is really granted.

```

1578     \bool_if:NTF \l__zrefclever_typeset_range_bool
1579     {
1580         \group_begin:
1581         \bool_set_false:N \l_tmpa_bool
1582         \exp_args:Nxx \tl_if_eq:nnT
1583         {
1584             \__zrefclever_extract_unexp:nnn
1585             {#1} { externaldocument } { }
1586         }
1587         {
1588             \__zrefclever_extract_unexp:nnn

```

```

1589         {#2} { externaldocument } { }
1590     }
1591     {
1592         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1593         {
1594             \exp_args:Nxx \tl_if_eq:nnT
1595             {
1596                 \__zrefclever_extract_unexp:nnn
1597                 {#1} { zc@pgfmt } { }
1598             }
1599             {
1600                 \__zrefclever_extract_unexp:nnn
1601                 {#2} { zc@pgfmt } { }
1602             }
1603             { \bool_set_true:N \l_tmpa_bool }
1604         }
1605         {
1606             \exp_args:Nxx \tl_if_eq:nnT
1607             {
1608                 \__zrefclever_extract_unexp:nnn
1609                 {#1} { zc@counter } { }
1610             }
1611             {
1612                 \__zrefclever_extract_unexp:nnn
1613                 {#2} { zc@counter } { }
1614             }
1615             {
1616                 \exp_args:Nxx \tl_if_eq:nnT
1617                 {
1618                     \__zrefclever_extract_unexp:nnn
1619                     {#1} { zc@enclval } { }
1620                 }
1621                 {
1622                     \__zrefclever_extract_unexp:nnn
1623                     {#2} { zc@enclval } { }
1624                 }
1625                 { \bool_set_true:N \l_tmpa_bool }
1626             }
1627         }
1628     }
1629     \bool_if:NTF \l_tmpa_bool
1630     {
1631         \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1632         {#2} { l__zrefclever_endrangeprop_tl } { }
1633     }
1634     {
1635         \zref@ifrefcontainsprop
1636         {#2} { \l__zrefclever_ref_property_tl }
1637         {
1638             \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1639             {#2} { l__zrefclever_ref_property_tl } { }
1640         }
1641         { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1642     }

```

```

1643         \exp_args:NNNV
1644         \group_end:
1645         \tl_set:Nn #3 \l_tmpb_tl
1646     }
1647     {
1648         \__zrefclever_extract_default:Nnvn #3
1649         {#2} { \l__zrefclever_endrangeprop_tl } { }
1650     }
1651 }
1652 {
1653     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1654     {
1655         \__zrefclever_extract_default:Nnvn #3
1656         {#2} { \l__zrefclever_ref_property_tl } { }
1657     }
1658     { \tl_set:Nn #3 { zc@missingproperty } }
1659 }
1660 }
1661 }
1662 \cs_generate_variant:Nn \__zrefclever_get_endrange_property:nnN { VVN }

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

1663 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1664 {
1665     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1666     {
1667         \group_begin:
1668         \UseHook { zref-clever/endrange-setup }
1669         \tl_set:Nx \l_tmpa_tl
1670         {
1671             \__zrefclever_extract:nnn
1672             {#1} { \l__zrefclever_ref_property_tl } { }
1673         }
1674         \tl_set:Nx \l_tmpb_tl
1675         {
1676             \__zrefclever_extract:nnn
1677             {#2} { \l__zrefclever_ref_property_tl } { }
1678         }
1679         \bool_set_false:N \l_tmpa_bool
1680         \bool_until_do:Nn \l_tmpa_bool
1681         {
1682             \exp_args:Nxx \tl_if_eq:nnTF
1683             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1684             {
1685                 \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1686                 \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1687                 \tl_if_empty:NT \l_tmpb_tl
1688                 { \bool_set_true:N \l_tmpa_bool }
1689             }
1690             { \bool_set_true:N \l_tmpa_bool }
1691         }
1692     }
1693     \exp_args:NNNV
1694     \group_end:

```

```

1694         \tl_set:Nn #3 \l_tmpb_tl
1695     }
1696     { \tl_set:Nn #3 { zc@missingproperty } }
1697 }
1698 \cs_generate_variant:Nn \__zrefclever_get_endrange_stripprefix:nnN { VVN }

```

_zrefclever_is_integer_rgx:n Test if argument is composed only of digits (adapted from <https://tex.stackexchange.com/a/427559>).

```

1699 \prg_new_protected_conditional:Npnn
1700   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1701 {
1702   \regex_match:nnTF { \A\d+\Z } {#1}
1703   { \prg_return_true: }
1704   { \prg_return_false: }
1705 }
1706 \prg_generate_conditional_variant:Nnn
1707   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End definition for _zrefclever_is_integer_rgx:n.)

```

1708 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1709 {
1710   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1711   {
1712     \group_begin:
1713     \UseHook { zref-clever/endrange-setup }
1714     \tl_set:Nx \l_tmpa_tl
1715     {
1716       \__zrefclever_extract:nnn
1717       {#1} { \l__zrefclever_ref_property_tl } { }
1718     }
1719     \tl_set:Nx \l_tmpb_tl
1720     {
1721       \__zrefclever_extract:nnn
1722       {#2} { \l__zrefclever_ref_property_tl } { }
1723     }
1724     \bool_set_false:N \l_tmpa_bool
1725     \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1726     {
1727       \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1728       { \bool_set_true:N \l_tmpa_bool }
1729     }
1730     { \bool_set_true:N \l_tmpa_bool }
1731     \bool_until_do:Nn \l_tmpa_bool
1732     {
1733       \exp_args:Nxx \tl_if_eq:nnTF
1734       { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1735       {
1736         \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1737         \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1738         \tl_if_empty:NT \l_tmpb_tl
1739         { \bool_set_true:N \l_tmpa_bool }
1740       }
1741       { \bool_set_true:N \l_tmpa_bool }
1742     }
1743   }

```

```

1743         \exp_args:NNNV
1744         \group_end:
1745         \tl_set:Nn #3 \l_tmpb_tl
1746     }
1747     { \tl_set:Nn #3 { zc@missingproperty } }
1748 }
1749 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1750 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1751 {
1752     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1753     {
1754         \group_begin:
1755         \UseHook { zref-clever/endrange-setup }
1756         \tl_set:Nx \l_tmpa_tl
1757         {
1758             \__zrefclever_extract:nnn
1759             {#1} { \l__zrefclever_ref_property_tl } { }
1760         }
1761         \tl_set:Nx \l_tmpb_tl
1762         {
1763             \__zrefclever_extract:nnn
1764             {#2} { \l__zrefclever_ref_property_tl } { }
1765         }
1766         \bool_set_false:N \l_tmpa_bool
1767         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1768         {
1769             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1770             { \bool_set_true:N \l_tmpa_bool }
1771         }
1772         { \bool_set_true:N \l_tmpa_bool }
1773         \bool_until_do:Nn \l_tmpa_bool
1774         {
1775             \exp_args:Nxx \tl_if_eq:nnTF
1776             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1777             {
1778                 \bool_lazy_or:nnTF
1779                 { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1780                 { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1781                 {
1782                     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1783                     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1784                 }
1785                 { \bool_set_true:N \l_tmpa_bool }
1786             }
1787             { \bool_set_true:N \l_tmpa_bool }
1788         }
1789         \exp_args:NNNV
1790         \group_end:
1791         \tl_set:Nn #3 \l_tmpb_tl
1792     }
1793     { \tl_set:Nn #3 { zc@missingproperty } }
1794 }
1795 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomptwo:nnN { VVN }

```

range and rangetopair options

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

```
1796 \bool_new:N \l__zrefclever_typeset_range_bool
1797 \keys_define:nn { zref-clever/reference }
1798 {
1799     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1800     range .initial:n = false ,
1801     range .default:n = true ,
1802 }
```

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`.

```
1803 \bool_new:N \l__zrefclever_capfirst_bool
1804 \keys_define:nn { zref-clever/reference }
1805 {
1806     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1807     capfirst .initial:n = false ,
1808     capfirst .default:n = true ,
1809 }
```

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`.

```
1810 \bool_new:N \l__zrefclever_noabbrev_first_bool
1811 \keys_define:nn { zref-clever/reference }
1812 {
1813     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1814     noabbrevfirst .initial:n = false ,
1815     noabbrevfirst .default:n = true ,
1816 }
```

S option

```
1817 \keys_define:nn { zref-clever/reference }
1818 {
1819     S .meta:n =
1820         { capfirst = {#1} , noabbrevfirst = {#1} },
1821     S .default:n = true ,
1822 }
```

hyperref option

```
1823 \bool_new:N \l__zrefclever_hyperlink_bool
1824 \bool_new:N \l__zrefclever_hyperref_warn_bool
1825 \keys_define:nn { zref-clever/reference }
1826 {
1827     hyperref .choice: ,
1828     hyperref / auto .code:n =
1829     {
```



```

1830     \bool_set_true:N \l__zrefclever_hyperlink_bool
1831     \bool_set_false:N \l__zrefclever_hyperref_warn_bool
1832   } ,
1833   hyperref / true .code:n =
1834   {
1835     \bool_set_true:N \l__zrefclever_hyperlink_bool
1836     \bool_set_true:N \l__zrefclever_hyperref_warn_bool
1837   } ,
1838   hyperref / false .code:n =
1839   {
1840     \bool_set_false:N \l__zrefclever_hyperlink_bool
1841     \bool_set_false:N \l__zrefclever_hyperref_warn_bool
1842   } ,
1843   hyperref .initial:n = auto ,
1844   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.

```

1845     nohyperref .meta:n = { hyperref = false } ,
1846     nohyperref .value_forbidden:n = true ,
1847   }
1848   \AddToHook { begindocument }
1849   {
1850     \__zrefclever_if_package_loaded:nTF { hyperref }
1851     {
1852       \bool_if:NT \l__zrefclever_hyperlink_bool
1853       { \RequirePackage { zref-hyperref } }
1854     }
1855     {
1856       \bool_if:NT \l__zrefclever_hyperref_warn_bool
1857       { \msg_warning:nn { zref-clever } { missing-hyperref } }
1858       \bool_set_false:N \l__zrefclever_hyperlink_bool
1859     }
1860     \keys_define:nn { zref-clever/reference }
1861     {
1862       hyperref .code:n =
1863       { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
1864       nohyperref .code:n =
1865       { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
1866     }
1867   }

```

nameinlink option

```

1868   \str_new:N \l__zrefclever_nameinlink_str
1869   \keys_define:nn { zref-clever/reference }
1870   {
1871     nameinlink .choice: ,
1872     nameinlink / true .code:n =
1873     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
1874     nameinlink / false .code:n =
1875     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,

```

```

1876     nameinlink / single .code:n =
1877       { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
1878     nameinlink / tsingle .code:n =
1879       { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
1880     nameinlink .initial:n = tsingle ,
1881     nameinlink .default:n = true ,
1882   }

```

preposinlink option (deprecated)

```

1883 \keys_define:nn { zref-clever/reference }
1884 {
1885   preposinlink .code:n =
1886     {
1887       % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
1888       \msg_warning:nnnn { zref-clever }{ option-deprecated }
1889       { preposinlink } { refbounds }
1890     } ,
1891 }

```

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`.

```

1892 \tl_new:N \l__zrefclever_ref_language_tl
1893 \tl_new:N \l__zrefclever_current_language_tl
1894 \tl_new:N \l__zrefclever_main_language_tl
1895 \AddToHook { begindocument }
1896 {
1897   \__zrefclever_if_package_loaded:nTF { babel }

```

```

1898 {
1899   \tl_set:Nn \l__zrefclever_current_language_tl { \language }
1900   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
1901 }
1902 {
1903   \__zrefclever_if_package_loaded:nTF { polyglossia }
1904   {
1905     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
1906     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
1907   }
1908   {
1909     \tl_set:Nn \l__zrefclever_current_language_tl { english }
1910     \tl_set:Nn \l__zrefclever_main_language_tl { english }
1911   }
1912 }
1913 }

```

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

```

1914 \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 ??.)

```

1915 \keys_define:nn { zref-clever/reference }
1916 {
1917   lang .code:n =
1918   {
1919     \AddToHook { begindocument }
1920     {
1921       \str_case:nnF {#1}
1922       {
1923         { current }
1924         {
1925           \tl_set:Nn \l__zrefclever_ref_language_tl
1926             { \l__zrefclever_current_language_tl }
1927         }
1928
1929         { main }
1930         {
1931           \tl_set:Nn \l__zrefclever_ref_language_tl
1932             { \l__zrefclever_main_language_tl }
1933         }
1934       }
1935     {
1936       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1937       \__zrefclever_language_if_declared:nF {#1}
1938       {
1939         \msg_warning:nnn { zref-clever }
1940           { unknown-language-opt } {#1}
1941       }
1942     }
1943     \__zrefclever_provide_langfile:x
1944       { \l__zrefclever_ref_language_tl }
1945   }
1946 },
1947 lang .initial:n = current ,

```

```

1948     lang .value_required:n = true ,
1949   }

1950   \AddToHook { begindocument / before }
1951   {
1952     \AddToHook { begindocument }
1953     {

```

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_zceref:nnn` already ensures it.

```

1954     \keys_define:nn { zref-clever/reference }
1955     {
1956       lang .code:n =
1957       {
1958         \str_case:nnF {#1}
1959         {
1960           { current }
1961           {
1962             \tl_set:Nn \l__zrefclever_ref_language_tl
1963             { \l__zrefclever_current_language_tl }
1964           }
1965
1966           { main }
1967           {
1968             \tl_set:Nn \l__zrefclever_ref_language_tl
1969             { \l__zrefclever_main_language_tl }
1970           }
1971         }
1972       }
1973       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
1974       \__zrefclever_language_if_declared:nF {#1}
1975       {
1976         \msg_warning:nnn { zref-clever }
1977         { unknown-language-opt } {#1}
1978       }
1979     }
1980   } ,
1981 }
1982 }
1983 }

```

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.

```

1984 \tl_new:N \l__zrefclever_ref_decl_case_tl
1985 \keys_define:nn { zref-clever/reference }
1986 {
1987   d .code:n =

```

```

1988     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1989   }
1990   \AddToHook { begindocument }
1991     {
1992     \keys_define:nn { zref-clever/reference }
1993     {

```

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

```

1994         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1995         d .value_required:n = true ,
1996     }
1997   }

```

nudge & co. options

```

1998   \bool_new:N \l__zrefclever_nudge_enabled_bool
1999   \bool_new:N \l__zrefclever_nudge_multitype_bool
2000   \bool_new:N \l__zrefclever_nudge_comptosing_bool
2001   \bool_new:N \l__zrefclever_nudge_singular_bool
2002   \bool_new:N \l__zrefclever_nudge_gender_bool
2003   \tl_new:N \l__zrefclever_ref_gender_tl
2004   \keys_define:nn { zref-clever/reference }
2005   {
2006     nudge .choice: ,
2007     nudge / true .code:n =
2008       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2009     nudge / false .code:n =
2010       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2011     nudge / ifdraft .code:n =
2012       {
2013         \ifdraft
2014           { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2015           { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2016         } ,
2017     nudge / iffinal .code:n =
2018       {
2019         \ifoptionfinal
2020           { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2021           { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2022         } ,
2023     nudge .initial:n = false ,
2024     nudge .default:n = true ,
2025     nonnudge .meta:n = { nudge = false } ,
2026     nonnudge .value_forbidden:n = true ,
2027     nudgeif .code:n =
2028       {
2029         \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2030         \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2031         \bool_set_false:N \l__zrefclever_nudge_gender_bool
2032         \clist_map_inline:nn {#1}
2033         {
2034           \str_case:nnF {##1}
2035           {

```

```

2036         { multitype }
2037         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2038         { comptosing }
2039         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2040         { gender }
2041         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2042         { all }
2043         {
2044             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2045             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2046             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2047         }
2048     }
2049     {
2050         \msg_warning:nnn { zref-clever }
2051         { nudgeif-unknown-value } {##1}
2052     }
2053 }
2054 },
2055 nudgeif .value_required:n = true ,
2056 nudgeif .initial:n = all ,
2057 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2058 sg .initial:n = false ,
2059 sg .default:n = true ,
2060 g .code:n =
2061     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2062 }
2063 \AddToHook { begindocument }
2064 {
2065     \keys_define:nn { zref-clever/reference }
2066     {

```

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

```

2067         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2068         g .value_required:n = true ,
2069     }
2070 }

```

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 `\zceref` 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>).

```

2071 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2072 \keys_define:nn { zref-clever/reference }
2073 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2074 \keys_define:nn { zref-clever/reference }
2075 {
2076     titleref .code:n = { \RequirePackage { zref-titleref } } ,

```

```

2077     titleref .value_forbidden:n = true ,
2078   }
2079   \AddToHook { begindocument }
2080   {
2081     \keys_define:nn { zref-clever/reference }
2082     {
2083       titleref .code:n =
2084       { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
2085     }
2086   }

```

vario option

```

2087 \keys_define:nn { zref-clever/reference }
2088 {
2089   vario .code:n = { \RequirePackage { zref-vario } } ,
2090   vario .value_forbidden:n = true ,
2091 }
2092 \AddToHook { begindocument }
2093 {
2094   \keys_define:nn { zref-clever/reference }
2095   {
2096     vario .code:n =
2097     {
2098       \msg_warning:nnn { zref-clever }
2099       { option-preamble-only } { vario }
2100     }
2101   }
2102 }

```

note option

```

2103 \tl_new:N \l__zrefclever_zcref_note_tl
2104 \keys_define:nn { zref-clever/reference }
2105 {
2106   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2107   note .value_required:n = true ,
2108 }

```

check option

Integration with zref-check.

```

2109 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2110 \bool_new:N \l__zrefclever_zcref_with_check_bool
2111 \keys_define:nn { zref-clever/reference }
2112 {
2113   check .code:n = { \RequirePackage { zref-check } } ,
2114   check .value_forbidden:n = true ,
2115 }
2116 \AddToHook { begindocument }
2117 {
2118   \__zrefclever_if_package_loaded:nTF { zref-check }
2119   {
2120     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2121     \keys_define:nn { zref-clever/reference }
2122     {

```

```

2123         check .code:n =
2124             {
2125                 \bool_set_true:N \l__zrefclever_zceref_with_check_bool
2126                 \keys_set:nn { zref-check / zcheck } {#1}
2127             } ,
2128         check .value_required:n = true ,
2129     }
2130 }
2131 {
2132     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2133     \keys_define:nn { zref-clever/reference }
2134     {
2135         check .value_forbidden:n = false ,
2136         check .code:n =
2137             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2138     }
2139 }
2140 }

```

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`.

```

2141 \prop_new:N \l__zrefclever_counter_type_prop
2142 \keys_define:nn { zref-clever/label }
2143 {
2144     countertype .code:n =
2145     {
2146         \keyval_parse:nnn
2147         {
2148             \msg_warning:nnnn { zref-clever }
2149             { key-requires-value } { countertype }
2150         }
2151         {
2152             \__zrefclever_prop_put_non_empty:Nnn
2153             \l__zrefclever_counter_type_prop
2154         }
2155         {#1}
2156     } ,
2157     countertype .value_required:n = true ,
2158     countertype .initial:n =
2159     {
2160         subsection    = section ,
2161         subsubsection = section ,
2162         subparagraph  = paragraph ,
2163         enumi          = item ,
2164         enumii         = item ,
2165         enumiii        = item ,
2166         enumiv         = item ,
2167         mpfootnote     = footnote ,
2168     } ,

```


2169 }

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

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclval` property, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resetters_seq` with the `counterresetby` option.

```

2170 \seq_new:N \l__zrefclever_counter_resetters_seq
2171 \keys_define:nn { zref-clever/label }
2172 {
2173   counterresetters .code:n =
2174   {
2175     \clist_map_inline:nn {#1}
2176     {
2177       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2178       {
2179         \seq_put_right:Nn
2180         \l__zrefclever_counter_resetters_seq {##1}
2181       }
2182     }
2183   } ,
2184   counterresetters .initial:n =
2185   {
2186     part ,
2187     chapter ,
2188     section ,
2189     subsection ,
2190     subsubsection ,
2191     paragraph ,
2192     subparagraph ,
2193   } ,
2194   counterresetters .value_required:n = true ,
2195 }
```

counterresetby option

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

```
2196 \prop_new:N \l__zrefclever_counter_resetby_prop
2197 \keys_define:nn { zref-clever/label }
2198 {
2199     counterresetby .code:n =
2200     {
2201         \keyval_parse:nnn
2202         {
2203             \msg_warning:nnn { zref-clever }
2204             { key-requires-value } { counterresetby }
2205         }
2206         {
2207             \__zrefclever_prop_put_non_empty:Nnn
2208             \l__zrefclever_counter_resetby_prop
2209         }
2210         {#1}
2211     } ,
2212     counterresetby .value_required:n = true ,
2213     counterresetby .initial:n =
2214     {
```

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.

```
2215     enumii = enumi ,
2216     enumiii = enumii ,
2217     enumiv = enumiii ,
2218 } ,
2219 }
```

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.

```
2220 \tl_new:N \l__zrefclever_current_counter_tl
2221 \keys_define:nn { zref-clever/label }
2222 {
2223     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2224     currentcounter .value_required:n = true ,
2225     currentcounter .initial:n = \@currentcounter ,
2226 }
```

nocompat option

```

2227 \bool_new:N \g__zrefclever_nocompat_bool
2228 \seq_new:N \g__zrefclever_nocompat_modules_seq
2229 \keys_define:nn { zref-clever/reference }
2230 {
2231   nocompat .code:n =
2232   {
2233     \tl_if_empty:nTF {#1}
2234     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2235     {
2236       \clist_map_inline:nn {#1}
2237       {
2238         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2239         {
2240           \seq_gput_right:Nn
2241             \g__zrefclever_nocompat_modules_seq {##1}
2242         }
2243       }
2244     }
2245   },
2246 }
2247 \AddToHook { begindocument }
2248 {
2249   \keys_define:nn { zref-clever/reference }
2250   {
2251     nocompat .code:n =
2252     {
2253       \msg_warning:nnn { zref-clever }
2254       { option-preamble-only } { nocompat }
2255     }
2256   }
2257 }
2258 \AtEndOfPackage
2259 {
2260   \AddToHook { begindocument }
2261   {
2262     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2263     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2264   }
2265 }

```

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

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

```

2266 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2267 {
2268   \AddToHook { begindocument }
2269   {
2270     \bool_if:NF \g__zrefclever_nocompat_bool
2271     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2272     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2273   }
2274 }

```

(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.

```

2275 \seq_map_inline:Nn
2276   \c__zrefclever_rf_opts_tl_reference_seq
2277 {
2278   \keys_define:nn { zref-clever/reference }
2279   {
2280     #1 .default:x = \c_novalue_tl ,
2281     #1 .code:n =
2282     {
2283       \tl_if_novalue:nTF {##1}
2284       {
2285         \__zrefclever_opt_tl_unset:c
2286         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2287       }
2288       {
2289         \tl_set:cn
2290         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2291         {##1}
2292       }
2293     } ,
2294   }
2295 }
2296 \keys_define:nn { zref-clever/reference }
2297 {
2298   refpre .code:n =
2299   {
2300     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2301     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2302     { refpre } { refbounds }
2303   } ,
2304   refpos .code:n =
2305   {
2306     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2307     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2308     { refpos } { refbounds }
2309   } ,
2310   preref .code:n =

```

```

2311     {
2312         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2313         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2314         { preref } { refbounds }
2315     } ,
2316     postref .code:n =
2317     {
2318         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2319         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2320         { postref } { refbounds }
2321     } ,
2322 }
2323 \seq_map_inline:Nn
2324   \c__zrefclever_rf_opts_seq_refbounds_seq
2325   {
2326     \keys_define:nn { zref-clever/reference }
2327     {
2328       #1 .default:x = \c_novalue_tl ,
2329       #1 .code:n =
2330       {
2331         \tl_if_novalue:nTF {##1}
2332         {
2333           \__zrefclever_opt_seq_unset:c
2334           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2335         }
2336         {
2337           \seq_clear:N \l_tmpa_seq
2338           \__zrefclever_opt_seq_set_clist_split:Nn
2339             \l_tmpa_seq {##1}
2340           \bool_lazy_or:nnTF
2341             { \tl_if_empty_p:n {##1} }
2342             { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2343             {
2344               \seq_set_eq:cN
2345                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2346                 \l_tmpa_seq
2347             }
2348             {
2349               \msg_warning:nxxx { zref-clever }
2350               { refbounds-must-be-four }
2351               {#1} { \seq_count:N \l_tmpa_seq }
2352             }
2353           }
2354         } ,
2355       }
2356     }
2357 \seq_map_inline:Nn
2358   \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
2359   {
2360     \keys_define:nn { zref-clever/reference }
2361     {
2362       #1 .choice: ,
2363       #1 / true .code:n =
2364       {

```

```

2365         \bool_set_true:c
2366         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2367     } ,
2368     #1 / false .code:n =
2369     {
2370         \bool_set_false:c
2371         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2372     } ,
2373     #1 / unset .code:n =
2374     {
2375         \__zrefclever_opt_bool_unset:c
2376         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2377     } ,
2378     #1 .default:n = true ,
2379     no #1 .meta:n = { #1 = false } ,
2380     no #1 .value_forbidden:n = true ,
2381 }
2382 }

```

Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: `label` and `reference`. Currently, the only use of this selection is the ability to exclude label related options from `\zcref`'s options. Anyway, for 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.

```

2383 \keys_define:nn { }
2384 {
2385     zref-clever/zcsetup .inherit:n =
2386     {
2387         zref-clever/label ,
2388         zref-clever/reference ,
2389     }
2390 }

```

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

```

2391 \ProcessKeysOptions { zref-clever/zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

```

```

2392 \NewDocumentCommand \zcsetup { m }
2393 { \__zrefclever_zcsetup:n {#1} }

```

(End definition for `\zcsetup`.)

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

```

\__zrefclever_zcsetup:n{<options>}

```

```

2394 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2395 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2396 \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>}

2397 \NewDocumentCommand \zcRefTypeSetup { m m }
2398 {
2399   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2400   \keys_set:nn { zref-clever/typesetup } {#2}
2401   \tl_clear:N \l__zrefclever_setup_type_tl
2402 }

(End definition for \zcRefTypeSetup.)

2403 \seq_map_inline:Nn
2404 \c__zrefclever_rf_opts_tl_not_type_specific_seq
2405 {
2406   \keys_define:nn { zref-clever/typesetup }
2407   {
2408     #1 .code:n =
2409     {
2410       \msg_warning:nnn { zref-clever }
2411       { option-not-type-specific } {#1}
2412     } ,
2413   }
2414 }
2415 \seq_map_inline:Nn
2416 \c__zrefclever_rf_opts_tl_typesetup_seq
2417 {
2418   \keys_define:nn { zref-clever/typesetup }
2419   {
2420     #1 .default:x = \c_novalue_tl ,
2421     #1 .code:n =
2422     {
2423       \tl_if_novalue:nTF {##1}
2424       {
2425         \__zrefclever_opt_tl_unset:c
2426         {
2427           \__zrefclever_opt_varname_type:enn
2428           { \l__zrefclever_setup_type_tl } {#1} { tl }
2429         }
2430       }
2431     }

```

```

2432         \tl_set:cn
2433         {
2434             \__zrefclever_opt_varname_type:enn
2435             { \l__zrefclever_setup_type_tl } {#1} { t1 }
2436         }
2437         {##1}
2438     }
2439 } ,
2440 }
2441 }
2442 \keys_define:nn { zref-clever/typesetup }
2443 {
2444     endrange .code:n =
2445     {
2446         \str_case:nnF {#1}
2447         {
2448             { ref }
2449             {
2450                 \tl_clear:c
2451                 {
2452                     \__zrefclever_opt_varname_type:enn
2453                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2454                 }
2455                 \tl_clear:c
2456                 {
2457                     \__zrefclever_opt_varname_type:enn
2458                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2459                 }
2460             }
2461         }
2462     { stripprefix }
2463     {
2464         \tl_set:cn
2465         {
2466             \__zrefclever_opt_varname_type:enn
2467             { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2468         }
2469         { __zrefclever_get_endrange_stripprefix }
2470         \tl_clear:c
2471         {
2472             \__zrefclever_opt_varname_type:enn
2473             { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2474         }
2475     }
2476     { pagecomp }
2477     {
2478         \tl_set:cn
2479         {
2480             \__zrefclever_opt_varname_type:enn
2481             { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2482         }
2483         { __zrefclever_get_endrange_pagecomp }
2484         \tl_clear:c
2485     }

```



```

2486         {
2487             \__zrefclever_opt_varname_type:enn
2488             { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2489         }
2490     }
2491
2492     { pagecomp2 }
2493     {
2494         \tl_set:cn
2495         {
2496             \__zrefclever_opt_varname_type:enn
2497             { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2498         }
2499         { __zrefclever_get_endrange_pagecomptwo }
2500         \tl_clear:c
2501         {
2502             \__zrefclever_opt_varname_type:enn
2503             { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2504         }
2505     }
2506
2507     { unset }
2508     {
2509         \__zrefclever_opt_t1_unset:c
2510         {
2511             \__zrefclever_opt_varname_type:enn
2512             { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2513         }
2514         \__zrefclever_opt_t1_unset:c
2515         {
2516             \__zrefclever_opt_varname_type:enn
2517             { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2518         }
2519     }
2520 }
2521 {
2522     \tl_if_empty:nTF {#1}
2523     {
2524         \msg_warning:nnn { zref-clever }
2525         { endrange-property-undefined } {#1}
2526     }
2527     {
2528         \zref@ifpropundefined {#1}
2529         {
2530             \msg_warning:nnn { zref-clever }
2531             { endrange-property-undefined } {#1}
2532         }
2533         {
2534             \tl_set:cn
2535             {
2536                 \__zrefclever_opt_varname_type:enn
2537                 { \l__zrefclever_setup_type_t1 }
2538                 { endrangefunc } { t1 }
2539             }

```

```

2540         { __zrefclever_get_endrange_property }
2541     \tl_set:cn
2542     {
2543         \__zrefclever_opt_varname_type:enn
2544         { \l__zrefclever_setup_type_tl }
2545         { endrangeprop } { tl }
2546     }
2547     {#1}
2548 }
2549 }
2550 }
2551 },
2552 endrange .value_required:n = true ,
2553 }
2554 \keys_define:nn { zref-clever/typesetup }
2555 {
2556     refpre .code:n =
2557     {
2558         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2559         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2560         { refpre } { refbounds }
2561     } ,
2562     refpos .code:n =
2563     {
2564         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2565         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2566         { refpos } { refbounds }
2567     } ,
2568     preref .code:n =
2569     {
2570         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2571         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2572         { preref } { refbounds }
2573     } ,
2574     postref .code:n =
2575     {
2576         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2577         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2578         { postref } { refbounds }
2579     } ,
2580 }
2581 \seq_map_inline:Nn
2582 \c__zrefclever_rf_opts_seq_refbounds_seq
2583 {
2584     \keys_define:nn { zref-clever/typesetup }
2585     {
2586         #1 .default:x = \c_novalue_tl ,
2587         #1 .code:n =
2588         {
2589             \tl_if_novalue:nTF {##1}
2590             {
2591                 \__zrefclever_opt_seq_unset:c
2592                 {
2593                     \__zrefclever_opt_varname_type:enn

```

```

2594         { \l__zrefclever_setup_type_tl } {#1} { seq }
2595     }
2596 }
2597 {
2598     \seq_clear:N \l_tmpa_seq
2599     \__zrefclever_opt_seq_set_clist_split:Nn
2600         \l_tmpa_seq {##1}
2601     \bool_lazy_or:nnTF
2602         { \tl_if_empty_p:n {##1} }
2603         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2604     {
2605         \seq_set_eq:cN
2606             {
2607                 \__zrefclever_opt_varname_type:enn
2608                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2609             }
2610         \l_tmpa_seq
2611     }
2612     {
2613         \msg_warning:nxxx { zref-clever }
2614         { refbounds-must-be-four }
2615         {#1} { \seq_count:N \l_tmpa_seq }
2616     }
2617 }
2618 } ,
2619 }
2620 }
2621 \seq_map_inline:Nn
2622     \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
2623     {
2624         \keys_define:nn { zref-clever/typesetup }
2625         {
2626             #1 .choice: ,
2627             #1 / true .code:n =
2628             {
2629                 \bool_set_true:c
2630                 {
2631                     \__zrefclever_opt_varname_type:enn
2632                         { \l__zrefclever_setup_type_tl }
2633                         {#1} { bool }
2634                 }
2635             } ,
2636             #1 / false .code:n =
2637             {
2638                 \bool_set_false:c
2639                 {
2640                     \__zrefclever_opt_varname_type:enn
2641                         { \l__zrefclever_setup_type_tl }
2642                         {#1} { bool }
2643                 }
2644             } ,
2645             #1 / unset .code:n =
2646             {
2647                 \__zrefclever_opt_bool_unset:c

```

```

2648         {
2649             \__zrefclever_opt_varname_type:enn
2650             { \l__zrefclever_setup_type_tl }
2651             {#1} { bool }
2652         }
2653     } ,
2654     #1 .default:n = true ,
2655     no #1 .meta:n = { #1 = false } ,
2656     no #1 .value_forbidden:n = true ,
2657 }
2658 }

```

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>}
2659 \NewDocumentCommand \zcLanguageSetup { m m }
2660 {
2661     \group_begin:
2662     \__zrefclever_language_if_declared:NTF {#1}
2663     {
2664         \tl_clear:N \l__zrefclever_setup_type_tl
2665         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2666         \__zrefclever_opt_seq_get:cNF
2667         {
2668             \__zrefclever_opt_varname_language:nnn
2669             {#1} { declension } { seq }
2670         }
2671         \l__zrefclever_lang_declension_seq
2672         { \seq_clear:N \l__zrefclever_lang_declension_seq }
2673         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2674         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2675         {
2676             \seq_get_left:NN \l__zrefclever_lang_declension_seq
2677             \l__zrefclever_lang_decl_case_tl
2678         }
2679         \__zrefclever_opt_seq_get:cNF
2680         {
2681             \__zrefclever_opt_varname_language:nnn
2682             {#1} { gender } { seq }
2683         }
2684         \l__zrefclever_lang_gender_seq
2685         { \seq_clear:N \l__zrefclever_lang_gender_seq }
2686         \keys_set:nn { zref-clever/langsetup } {#2}
2687     }
2688     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }

```

```

2689 \group_end:
2690 }
2691 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2692 \keys_define:nn { zref-clever/langsetup }
2693 {
2694   type .code:n =
2695   {
2696     \tl_if_empty:NTF {#1}
2697     { \tl_clear:N \l__zrefclever_setup_type_tl }
2698     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2699   } ,
2700
2701   case .code:n =
2702   {
2703     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2704     {
2705       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2706       { \l__zrefclever_setup_language_tl } {#1}
2707     }
2708     {
2709       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2710       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2711       {
2712         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2713         {#1} { \l__zrefclever_setup_language_tl }
2714         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2715         \l__zrefclever_lang_decl_case_tl
2716       }
2717     }
2718   } ,
2719   case .value_required:n = true ,
2720
2721   gender .default:x = \c_novalue_tl ,
2722   gender .code:n =
2723   {
2724     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2725     {
2726       \msg_warning:nxxx { zref-clever } { language-no-gender }
2727       { \l__zrefclever_setup_language_tl } { gender } {#1}
2728     }
2729     {
2730       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2731       {
2732         \msg_warning:nnn { zref-clever }
2733         { option-only-type-specific } { gender }
2734       }
2735       {
2736         \tl_if_novalue:NTF {#1}
2737         {
2738           \__zrefclever_opt_seq_gunset:c

```

```

2739         {
2740             \__zrefclever_opt_varname_lang_type:enn
2741             { \l__zrefclever_setup_language_tl }
2742             { \l__zrefclever_setup_type_tl }
2743             { gender }
2744             { seq }
2745         }
2746     }
2747     {
2748         \seq_clear:N \l_tmpa_seq
2749         \clist_map_inline:nn {#1}
2750         {
2751             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2752             { \seq_put_right:Nn \l_tmpa_seq {##1} }
2753             {
2754                 \msg_warning:nxxx { zref-clever }
2755                 { gender-not-declared }
2756                 { \l__zrefclever_setup_language_tl } {##1}
2757             }
2758         }
2759         \seq_gset_eq:cN
2760         {
2761             \__zrefclever_opt_varname_lang_type:enn
2762             { \l__zrefclever_setup_language_tl }
2763             { \l__zrefclever_setup_type_tl }
2764             { gender }
2765             { seq }
2766         }
2767         \l_tmpa_seq
2768     }
2769 } ,
2770 }
2771 } ,
2772 }
2773 \seq_map_inline:Nn
2774 \c__zrefclever_rf_opts_tl_not_type_specific_seq
2775 {
2776     \keys_define:nn { zref-clever/langsetup }
2777     {
2778         #1 .default:x = \c_novalue_tl ,
2779         #1 .code:n =
2780         {
2781             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2782             {
2783                 \tl_if_novalue:NTF {##1}
2784                 {
2785                     \__zrefclever_opt_tl_gunset:c
2786                     {
2787                         \__zrefclever_opt_varname_lang_default:enn
2788                         { \l__zrefclever_setup_language_tl } {##1} { tl }
2789                     }
2790                 }
2791             }
2792             \tl_gset:cn

```

```

2793         {
2794             \__zrefclever_opt_varname_lang_default:enn
2795             { \l__zrefclever_setup_language_tl } {#1} { tl }
2796         }
2797         {##1}
2798     }
2799 }
2800 {
2801     \msg_warning:nnn { zref-clever }
2802     { option-not-type-specific } {#1}
2803 }
2804 },
2805 }
2806 }
2807 \seq_map_inline:Nn
2808   \c__zrefclever_rf_opts_tl_maybe_type_specific_seq
2809   {
2810     \keys_define:nn { zref-clever/langsetup }
2811     {
2812       #1 .value_required:n = true ,
2813       #1 .code:n =
2814       {
2815         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2816         {
2817           \tl_if_novalue:NTF {##1}
2818           {
2819             \__zrefclever_opt_tl_gunset:c
2820             {
2821               \__zrefclever_opt_varname_lang_default:enn
2822               { \l__zrefclever_setup_language_tl } {#1} { tl }
2823             }
2824           }
2825           {
2826             \tl_gset:cn
2827             {
2828               \__zrefclever_opt_varname_lang_default:enn
2829               { \l__zrefclever_setup_language_tl } {#1} { tl }
2830             }
2831             {##1}
2832           }
2833         }
2834       }
2835       \tl_if_novalue:NTF {##1}
2836       {
2837         \__zrefclever_opt_tl_gunset:c
2838         {
2839           \__zrefclever_opt_varname_lang_type:eenn
2840           { \l__zrefclever_setup_language_tl }
2841           { \l__zrefclever_setup_type_tl }
2842           {#1} { tl }
2843         }
2844       }
2845       {
2846         \tl_gset:cn

```

```

2847         {
2848             \_zrefclever_opt_varname_lang_type:eenn
2849             { \l_zrefclever_setup_language_tl }
2850             { \l_zrefclever_setup_type_tl }
2851             {#1} { t1 }
2852         }
2853         {##1}
2854     }
2855 } ,
2856 } ,
2857 }
2858 }
2859 \keys_define:nn { zref-clever/langsetup }
2860 {
2861     endrange .code:n =
2862     {
2863         \str_case:nnF {#1}
2864         {
2865             { ref }
2866             {
2867                 \tl_if_empty:NTF \l_zrefclever_setup_type_tl
2868                 {
2869                     \tl_gclear:c
2870                     {
2871                         \_zrefclever_opt_varname_lang_default:enn
2872                         { \l_zrefclever_setup_language_tl }
2873                         { endrangefunc } { t1 }
2874                     }
2875                     \tl_gclear:c
2876                     {
2877                         \_zrefclever_opt_varname_lang_default:enn
2878                         { \l_zrefclever_setup_language_tl }
2879                         { endrangeprop } { t1 }
2880                     }
2881                 }
2882             }
2883             \tl_gclear:c
2884             {
2885                 \_zrefclever_opt_varname_lang_type:eenn
2886                 { \l_zrefclever_setup_language_tl }
2887                 { \l_zrefclever_setup_type_tl }
2888                 { endrangefunc } { t1 }
2889             }
2890             \tl_gclear:c
2891             {
2892                 \_zrefclever_opt_varname_lang_type:eenn
2893                 { \l_zrefclever_setup_language_tl }
2894                 { \l_zrefclever_setup_type_tl }
2895                 { endrangeprop } { t1 }
2896             }
2897         }
2898     }
2899 }
2900 { stripprefix }

```



```

2901 {
2902   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2903   {
2904     \tl_gset:cn
2905     {
2906       \__zrefclever_opt_varname_lang_default:enn
2907       { \l__zrefclever_setup_language_tl }
2908       { endrangefunc } { tl }
2909     }
2910     { __zrefclever_get_endrange_stripprefix }
2911   \tl_gclear:c
2912   {
2913     \__zrefclever_opt_varname_lang_default:enn
2914     { \l__zrefclever_setup_language_tl }
2915     { endrangeprop } { tl }
2916   }
2917 }
2918 {
2919   \tl_gset:cn
2920   {
2921     \__zrefclever_opt_varname_lang_type:enn
2922     { \l__zrefclever_setup_language_tl }
2923     { \l__zrefclever_setup_type_tl }
2924     { endrangefunc } { tl }
2925   }
2926   { __zrefclever_get_endrange_stripprefix }
2927   \tl_gclear:c
2928   {
2929     \__zrefclever_opt_varname_lang_type:enn
2930     { \l__zrefclever_setup_language_tl }
2931     { \l__zrefclever_setup_type_tl }
2932     { endrangeprop } { tl }
2933   }
2934 }
2935 }
2936
2937 { pagecomp }
2938 {
2939   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2940   {
2941     \tl_gset:cn
2942     {
2943       \__zrefclever_opt_varname_lang_default:enn
2944       { \l__zrefclever_setup_language_tl }
2945       { endrangefunc } { tl }
2946     }
2947     { __zrefclever_get_endrange_pagecomp }
2948   \tl_gclear:c
2949   {
2950     \__zrefclever_opt_varname_lang_default:enn
2951     { \l__zrefclever_setup_language_tl }
2952     { endrangeprop } { tl }
2953   }
2954 }

```

```

2955 {
2956   \tl_gset:cn
2957   {
2958     \__zrefclever_opt_varname_lang_type:een
2959     { \l__zrefclever_setup_language_tl }
2960     { \l__zrefclever_setup_type_tl }
2961     { endrangefunc } { tl }
2962   }
2963   { __zrefclever_get_endrange_pagecomp }
2964   \tl_gclear:c
2965   {
2966     \__zrefclever_opt_varname_lang_type:een
2967     { \l__zrefclever_setup_language_tl }
2968     { \l__zrefclever_setup_type_tl }
2969     { endrangeprop } { tl }
2970   }
2971 }
2972 }
2973
2974 { pagecomp2 }
2975 {
2976   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2977   {
2978     \tl_gset:cn
2979     {
2980       \__zrefclever_opt_varname_lang_default:enn
2981       { \l__zrefclever_setup_language_tl }
2982       { endrangefunc } { tl }
2983     }
2984     { __zrefclever_get_endrange_pagecomptwo }
2985     \tl_gclear:c
2986     {
2987       \__zrefclever_opt_varname_lang_default:enn
2988       { \l__zrefclever_setup_language_tl }
2989       { endrangeprop } { tl }
2990     }
2991   }
2992   {
2993     \tl_gset:cn
2994     {
2995       \__zrefclever_opt_varname_lang_type:een
2996       { \l__zrefclever_setup_language_tl }
2997       { \l__zrefclever_setup_type_tl }
2998       { endrangefunc } { tl }
2999     }
3000     { __zrefclever_get_endrange_pagecomptwo }
3001     \tl_gclear:c
3002     {
3003       \__zrefclever_opt_varname_lang_type:een
3004       { \l__zrefclever_setup_language_tl }
3005       { \l__zrefclever_setup_type_tl }
3006       { endrangeprop } { tl }
3007     }
3008   }

```

```

3009     }
3010
3011 { unset }
3012 {
3013   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3014   {
3015     \__zrefclever_opt_tl_gunset:c
3016     {
3017       \__zrefclever_opt_varname_lang_default:enn
3018       { \l__zrefclever_setup_language_tl }
3019       { endrangefunc } { tl }
3020     }
3021     \__zrefclever_opt_tl_gunset:c
3022     {
3023       \__zrefclever_opt_varname_lang_default:enn
3024       { \l__zrefclever_setup_language_tl }
3025       { endrangeprop } { tl }
3026     }
3027   }
3028   {
3029     \__zrefclever_opt_tl_gunset:c
3030     {
3031       \__zrefclever_opt_varname_lang_type:enn
3032       { \l__zrefclever_setup_language_tl }
3033       { \l__zrefclever_setup_type_tl }
3034       { endrangefunc } { tl }
3035     }
3036     \__zrefclever_opt_tl_gunset:c
3037     {
3038       \__zrefclever_opt_varname_lang_type:enn
3039       { \l__zrefclever_setup_language_tl }
3040       { \l__zrefclever_setup_type_tl }
3041       { endrangeprop } { tl }
3042     }
3043   }
3044 }
3045 }
3046 {
3047   \tl_if_empty:nTF {#1}
3048   {
3049     \msg_warning:nnn { zref-clever }
3050     { endrange-property-undefined } {#1}
3051   }
3052   {
3053     \zref@ifpropundefined {#1}
3054     {
3055       \msg_warning:nnn { zref-clever }
3056       { endrange-property-undefined } {#1}
3057     }
3058     {
3059       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3060       {
3061         \tl_gset:cn
3062         {

```

```

3063         \_zrefclever_opt_varname_lang_default:enn
3064         { \l__zrefclever_setup_language_tl }
3065         { endrangefunc } { tl }
3066     }
3067     { __zrefclever_get_endrange_property }
3068     \tl_gset:cn
3069     {
3070         \_zrefclever_opt_varname_lang_default:enn
3071         { \l__zrefclever_setup_language_tl }
3072         { endrangeprop } { tl }
3073     }
3074     {#1}
3075 }
3076 {
3077     \tl_gset:cn
3078     {
3079         \_zrefclever_opt_varname_lang_type:eenn
3080         { \l__zrefclever_setup_language_tl }
3081         { \l__zrefclever_setup_type_tl }
3082         { endrangefunc } { tl }
3083     }
3084     { __zrefclever_get_endrange_property }
3085     \tl_gset:cn
3086     {
3087         \_zrefclever_opt_varname_lang_type:eenn
3088         { \l__zrefclever_setup_language_tl }
3089         { \l__zrefclever_setup_type_tl }
3090         { endrangeprop } { tl }
3091     }
3092     {#1}
3093 }
3094 }
3095 }
3096 }
3097 },
3098     endrange .value_required:n = true ,
3099 }
3100 \keys_define:nn { zref-clever/langsetup }
3101 {
3102     refpre .code:n =
3103     {
3104         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3105         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3106         { refpre } { refbounds }
3107     } ,
3108     refpos .code:n =
3109     {
3110         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3111         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3112         { refpos } { refbounds }
3113     } ,
3114     preref .code:n =
3115     {
3116         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.

```

```

3117         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3118         { preref } { refbounds }
3119     } ,
3120     postref .code:n =
3121     {
3122         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3123         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3124         { postref } { refbounds }
3125     } ,
3126 }
3127 \seq_map_inline:Nn
3128 \c__zrefclever_rf_opts_tl_type_names_seq
3129 {
3130     \keys_define:nn { zref-clever/langsetup }
3131     {
3132         #1 .value_required:n = true ,
3133         #1 .code:n =
3134         {
3135             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3136             {
3137                 \msg_warning:nnn { zref-clever }
3138                 { option-only-type-specific } {#1}
3139             }
3140             {
3141                 \tl_if_novalue:NTF {##1}
3142                 {
3143                     \__zrefclever_opt_tl_gunset:c
3144                     {
3145                         \__zrefclever_opt_varname_lang_type:eenn
3146                         { \l__zrefclever_setup_language_tl }
3147                         { \l__zrefclever_setup_type_tl }
3148                         {#1} { tl }
3149                     }
3150                 }
3151             }
3152             \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3153             {
3154                 \tl_gset:cn
3155                 {
3156                     \__zrefclever_opt_varname_lang_type:eenn
3157                     { \l__zrefclever_setup_language_tl }
3158                     { \l__zrefclever_setup_type_tl }
3159                     {#1} { tl }
3160                 }
3161                 {##1}
3162             }
3163             {
3164                 \tl_gset:cn
3165                 {
3166                     \__zrefclever_opt_varname_lang_type:eeen
3167                     { \l__zrefclever_setup_language_tl }
3168                     { \l__zrefclever_setup_type_tl }
3169                     { \l__zrefclever_lang_decl_case_tl - #1 }
3170                     { tl }

```

```

3171         }
3172         {##1}
3173     }
3174 }
3175 }
3176 },
3177 }
3178 }
3179 \seq_map_inline:Nn
3180 \c__zrefclever_rf_opts_seq_refbounds_seq
3181 {
3182     \keys_define:nn { zref-clever/langsetup }
3183     {
3184         #1 .default:x = \c_novalue_tl ,
3185         #1 .code:n =
3186         {
3187             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3188             {
3189                 \tl_if_novalue:nTF {##1}
3190                 {
3191                     \__zrefclever_opt_seq_gunset:c
3192                     {
3193                         \__zrefclever_opt_varname_lang_default:enn
3194                         { \l__zrefclever_setup_language_tl }
3195                         {#1} { seq }
3196                     }
3197                 }
3198             }
3199             \seq_gclear:N \g_tmpa_seq
3200             \__zrefclever_opt_seq_gset_clist_split:Nn
3201             \g_tmpa_seq {##1}
3202             \bool_lazy_or:nnTF
3203             { \tl_if_empty_p:n {##1} }
3204             {
3205                 \int_compare_p:nNn
3206                 { \seq_count:N \g_tmpa_seq } = { 4 }
3207             }
3208             {
3209                 \seq_gset_eq:cN
3210                 {
3211                     \__zrefclever_opt_varname_lang_default:enn
3212                     { \l__zrefclever_setup_language_tl }
3213                     {#1} { seq }
3214                 }
3215                 \g_tmpa_seq
3216             }
3217             {
3218                 \msg_warning:nnxx { zref-clever }
3219                 { refbounds-must-be-four }
3220                 {#1} { \seq_count:N \g_tmpa_seq }
3221             }
3222         }
3223     }
3224 {

```

```

3225         \tl_if_novalue:nTF {##1}
3226         {
3227             \__zrefclever_opt_seq_gunset:c
3228             {
3229                 \__zrefclever_opt_varname_lang_type:eenn
3230                 { \l__zrefclever_setup_language_tl }
3231                 { \l__zrefclever_setup_type_tl } {#1} { seq }
3232             }
3233         }
3234         {
3235             \seq_gclear:N \g_tmpa_seq
3236             \__zrefclever_opt_seq_gset_clist_split:Nn
3237             \g_tmpa_seq {##1}
3238             \bool_lazy_or:nnTF
3239             { \tl_if_empty_p:n {##1} }
3240             {
3241                 \int_compare_p:nNn
3242                 { \seq_count:N \g_tmpa_seq } = { 4 }
3243             }
3244             {
3245                 \seq_gset_eq:cN
3246                 {
3247                     \__zrefclever_opt_varname_lang_type:eenn
3248                     { \l__zrefclever_setup_language_tl }
3249                     { \l__zrefclever_setup_type_tl } {#1} { seq }
3250                 }
3251                 \g_tmpa_seq
3252             }
3253             {
3254                 \msg_warning:nxxx { zref-clever }
3255                 { refbounds-must-be-four }
3256                 {#1} { \seq_count:N \g_tmpa_seq }
3257             }
3258         }
3259     } ,
3260 }
3261 }
3262 }
3263 \seq_map_inline:Nn
3264 \c__zrefclever_rf_opts_bool_maybe_type_specific_seq
3265 {
3266     \keys_define:nn { zref-clever/langsetup }
3267     {
3268         #1 .choice: ,
3269         #1 / true .code:n =
3270         {
3271             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3272             {
3273                 \bool_gset_true:c
3274                 {
3275                     \__zrefclever_opt_varname_lang_default:enn
3276                     { \l__zrefclever_setup_language_tl }
3277                     {#1} { bool }
3278                 }

```

```

3279         }
3280         {
3281             \bool_gset_true:c
3282             {
3283                 \__zrefclever_opt_varname_lang_type:enn
3284                 { \l__zrefclever_setup_language_tl }
3285                 { \l__zrefclever_setup_type_tl }
3286                 {#1} { bool }
3287             }
3288         }
3289     } ,
3290     #1 / false .code:n =
3291     {
3292         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3293         {
3294             \bool_gset_false:c
3295             {
3296                 \__zrefclever_opt_varname_lang_default:enn
3297                 { \l__zrefclever_setup_language_tl }
3298                 {#1} { bool }
3299             }
3300         }
3301         {
3302             \bool_gset_false:c
3303             {
3304                 \__zrefclever_opt_varname_lang_type:enn
3305                 { \l__zrefclever_setup_language_tl }
3306                 { \l__zrefclever_setup_type_tl }
3307                 {#1} { bool }
3308             }
3309         }
3310     } ,
3311     #1 / unset .code:n =
3312     {
3313         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3314         {
3315             \__zrefclever_opt_bool_gunset:c
3316             {
3317                 \__zrefclever_opt_varname_lang_default:enn
3318                 { \l__zrefclever_setup_language_tl }
3319                 {#1} { bool }
3320             }
3321         }
3322         {
3323             \__zrefclever_opt_bool_gunset:c
3324             {
3325                 \__zrefclever_opt_varname_lang_type:enn
3326                 { \l__zrefclever_setup_language_tl }
3327                 { \l__zrefclever_setup_type_tl }
3328                 {#1} { bool }
3329             }
3330         }
3331     } ,
3332     #1 .default:n = true ,

```



```

3333         no #1 .meta:n = { #1 = false } ,
3334         no #1 .value_forbidden:n = true ,
3335     }
3336 }

```

6 User interface

6.1 `\zcref`

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

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

```

3337 \NewDocumentCommand \zcref { s O { } m }
3338 { \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>}
```

```

3339 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3340 {
3341     \group_begin:

```

Set options.

```
3342     \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```

3343     \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3344     \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.

```
3345     \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3346     \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```

3347     \bool_lazy_and:nnT
3348     { \l__zrefclever_zrefcheck_available_bool }
3349     { \l__zrefclever_zcref_with_check_bool }
3350     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

3351     \bool_lazy_or:nnT
3352     { \l__zrefclever_typeset_sort_bool }
3353     { \l__zrefclever_typeset_range_bool }
3354     { \__zrefclever_sort_labels: }

```

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

```

3355     \group_begin:
3356     \l__zrefclever_ref_typeset_font_tl
3357     \__zrefclever_typeset_refs:
3358     \group_end:

```

Typeset note.

```

3359     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3360     {
3361         \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3362         { \l__zrefclever_label_type_a_tl }
3363         { \l__zrefclever_ref_language_tl }
3364         \l_tmpa_tl
3365         \l_tmpa_tl
3366         \l__zrefclever_zcref_note_tl
3367     }

```

Integration with zref-check.

```

3368     \bool_lazy_and:nnT
3369     { \l__zrefclever_zrefcheck_available_bool }
3370     { \l__zrefclever_zcref_with_check_bool }
3371     {
3372         \zrefcheck_zcref_end_label_maybe:
3373         \zrefcheck_zcref_run_checks_on_labels:n
3374         { \l__zrefclever_zcref_labels_seq }
3375     }

```

Integration with mathtools.

```

3376     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3377     {
3378         \__zrefclever_mathtools_showonlyrefs:n
3379         { \l__zrefclever_zcref_labels_seq }
3380     }
3381     \group_end:
3382 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3383 \seq_new:N \l__zrefclever_zcref_labels_seq
3384 \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>}

```

3385 \NewDocumentCommand \zcpageref { s O { } m }
3386 {
3387     \group_begin:
3388     \IfBooleanT {#1}

```

```

3389     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3390     \zcref [#2, ref = page] {#3}
3391     \group_end:
3392 }

```

(End definition for \zcpageref.)

7 Sorting

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

```

\l__zrefclever_label_type_a_tl Auxiliary variables, for use in sorting, and some also in typesetting. Used to store refer-
\l__zrefclever_label_type_b_tl ence information – label properties – of the “current” (a) and “next” (b) labels.
\l__zrefclever_label_enclval_a_tl 3393 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_enclval_b_tl 3394 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_extdoc_a_tl 3395 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_extdoc_b_tl 3396 \tl_new:N \l__zrefclever_label_enclval_b_tl
3397 \tl_new:N \l__zrefclever_label_extdoc_a_tl
3398 \tl_new:N \l__zrefclever_label_extdoc_b_tl

```

(End definition for \l__zrefclever_label_type_a_tl and others.)

```

\l__zrefclever_sort_decided_bool Auxiliary variable for \__zrefclever_sort_default_same_type:nn, signals if the sort-
ing between two labels has been decided or not.
3399 \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.
3400 \int_new:N \l__zrefclever_sort_prior_a_int
3401 \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.
3402 \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.

```
3403 \cs_new_protected:Npn \__zrefclever_sort_labels:
3404 {
```

Store label types sequence.

```
3405   \seq_clear:N \l__zrefclever_label_types_seq
3406   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
3407   {
3408     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3409     \__zrefclever_label_type_put_new_right:n
3410   }
```

Sort.

```
3411   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3412   {
3413     \zref@ifrefundefined {##1}
3414     {
3415       \zref@ifrefundefined {##2}
3416       {
3417         % Neither label is defined.
3418         \sort_return_same:
3419       }
3420       {
3421         % The second label is defined, but the first isn't, leave the
3422         % undefined first (to be more visible).
3423         \sort_return_same:
3424       }
3425     }
3426     {
3427       \zref@ifrefundefined {##2}
3428       {
3429         % The first label is defined, but the second isn't, bring the
3430         % second forward.
3431         \sort_return_swapped:
3432       }
3433       {
3434         % The interesting case: both labels are defined. References
3435         % to the "default" property or to the "page" are quite
3436         % different with regard to sorting, so we branch them here to
3437         % specialized functions.
3438         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3439         { \__zrefclever_sort_page:nn {##1} {##2} }
3440         { \__zrefclever_sort_default:nn {##1} {##2} }
3441       }
3442     }
3443   }
3444 }
```

(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}

3445 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3446 {
3447   \__zrefclever_extract_default:Nnnn
3448   \l__zrefclever_label_type_a_tl {#1} {zc@type} {}
3449   \seq_if_in:NVF \l__zrefclever_label_types_seq
3450   \l__zrefclever_label_type_a_tl
3451   {
3452     \seq_put_right:NV \l__zrefclever_label_types_seq
3453     \l__zrefclever_label_type_a_tl
3454   }
3455 }

```

(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}

3456 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3457 {
3458   \__zrefclever_extract_default:Nnnn
3459   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3460   \__zrefclever_extract_default:Nnnn
3461   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3462
3463   \tl_if_eq:NNTF
3464   \l__zrefclever_label_type_a_tl
3465   \l__zrefclever_label_type_b_tl
3466   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3467   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3468 }

```

(End definition for _zrefclever_sort_default:nn.)

_zrefclever_sort_default_same_type:nn

```

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

3469 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3470 {
3471   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3472   {#1} {zc@enclval} {}

```

```

3473 \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3474 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3475 {#2} { zc@enclval } { }
3476 \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3477 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3478 {#1} { externaldocument } { }
3479 \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3480 {#2} { externaldocument } { }
3481
3482 \bool_set_false:N \l__zrefclever_sort_decided_bool
3483
3484 % First we check if there's any "external document" difference (coming
3485 % from 'zref-xr') and, if so, sort based on that.
3486 \tl_if_eq:NNF
3487   \l__zrefclever_label_extdoc_a_tl
3488   \l__zrefclever_label_extdoc_b_tl
3489   {
3490     \bool_if:nTF
3491     {
3492       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3493       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3494     }
3495     {
3496       \bool_set_true:N \l__zrefclever_sort_decided_bool
3497       \sort_return_same:
3498     }
3499     {
3500       \bool_if:nTF
3501       {
3502         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3503         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3504       }
3505       {
3506         \bool_set_true:N \l__zrefclever_sort_decided_bool
3507         \sort_return_swapped:
3508       }
3509       {
3510         \bool_set_true:N \l__zrefclever_sort_decided_bool
3511         % Two different "external documents": last resort, sort by the
3512         % document name itself.
3513         \str_compare:eNeTF
3514         { \l__zrefclever_label_extdoc_b_tl } <
3515         { \l__zrefclever_label_extdoc_a_tl }
3516         { \sort_return_swapped: }
3517         { \sort_return_same: }
3518       }
3519     }
3520   }
3521
3522 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3523 {
3524   \bool_if:nTF
3525   {
3526     % Both are empty: neither label has any (further) "enclosing

```

```

3527 % counters" (left).
3528 \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3529 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3530 }
3531 {
3532   \bool_set_true:N \l__zrefclever_sort_decided_bool
3533   \int_compare:nNnTF
3534     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3535     >
3536     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3537     { \sort_return_swapped: }
3538     { \sort_return_same: }
3539 }
3540 {
3541   \bool_if:nTF
3542     {
3543       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
3544       \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3545     }
3546     {
3547       \bool_set_true:N \l__zrefclever_sort_decided_bool
3548       \int_compare:nNnTF
3549         { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3550         >
3551         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3552         { \sort_return_swapped: }
3553         { \sort_return_same: }
3554     }
3555   {
3556     \bool_if:nTF
3557       {
3558         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3559         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3560       }
3561       {
3562         \bool_set_true:N \l__zrefclever_sort_decided_bool
3563         \int_compare:nNnTF
3564           { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3565           <
3566           { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3567           { \sort_return_same: }
3568           { \sort_return_swapped: }
3569       }
3570     {
3571       % Neither is empty: we can compare the values of the
3572       % current enclosing counter in the loop, if they are
3573       % equal, we are still in the loop, if they are not, a
3574       % sorting decision can be made directly.
3575       \int_compare:nNnTF
3576         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3577         =
3578         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3579         {
3580           \tl_set:Nx \l__zrefclever_label_enclval_a_tl

```

```

3581         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3582         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3583         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3584     }
3585     {
3586         \bool_set_true:N \l__zrefclever_sort_decided_bool
3587         \int_compare:nNnTF
3588         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3589         >
3590         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3591         { \sort_return_swapped: }
3592         { \sort_return_same: }
3593     }
3594 }
3595 }
3596 }
3597 }
3598 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

```

__zrefclever_sort_default_different_types:nn      \__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
3599 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3600 {

```

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

```

3601     \int_zero:N \l__zrefclever_sort_prior_a_int
3602     \int_zero:N \l__zrefclever_sort_prior_b_int
3603     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3604     {
3605         \tl_if_eq:nnTF {##2} {{othertypes}}
3606         {
3607             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3608             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3609             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3610             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3611         }
3612         {
3613             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3614             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3615             {
3616                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3617                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3618             }
3619         }
3620     }

```

Then do the actual sorting.

```

3621     \bool_if:nTF
3622     {
3623         \int_compare_p:nNn
3624         { \l__zrefclever_sort_prior_a_int } <

```



```

3625         { \l__zrefclever_sort_prior_b_int }
3626     }
3627     { \sort_return_same: }
3628     {
3629         \bool_if:nTF
3630         {
3631             \int_compare_p:nNn
3632             { \l__zrefclever_sort_prior_a_int } >
3633             { \l__zrefclever_sort_prior_b_int }
3634         }
3635         { \sort_return_swapped: }
3636         {
3637             % Sort priorities are equal: the type that occurs first in
3638             % ‘labels’, as given by the user, is kept (or brought) forward.
3639             \seq_map_inline:Nn \l__zrefclever_label_types_seq
3640             {
3641                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3642                 { \seq_map_break:n { \sort_return_same: } }
3643                 {
3644                     \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3645                     { \seq_map_break:n { \sort_return_swapped: } }
3646                 }
3647             }
3648         }
3649     }
3650 }

```

(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>}

3651 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3652 {
3653     \int_compare:nNnTF
3654     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3655     >
3656     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3657     { \sort_return_swapped: }
3658     { \sort_return_same: }
3659 }

```

(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

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

```
\l_zrefclever_typeset_last_bool 3660 \seq_new:N \l_zrefclever_typeset_labels_seq
\l_zrefclever_last_of_type_bool 3661 \bool_new:N \l_zrefclever_typeset_last_bool
3662 \bool_new:N \l_zrefclever_last_of_type_bool
```

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

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

```
\l_zrefclever_label_count_int 3663 \int_new:N \l_zrefclever_type_count_int
\l_zrefclever_ref_count_int 3664 \int_new:N \l_zrefclever_label_count_int
3665 \int_new:N \l_zrefclever_ref_count_int
```

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

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

```
\l_zrefclever_label_b_tl 3666 \tl_new:N \l_zrefclever_label_a_tl
\l_zrefclever_typeset_queue_prev_tl 3667 \tl_new:N \l_zrefclever_label_b_tl
\l_zrefclever_typeset_queue_curr_tl 3668 \tl_new:N \l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_type_first_label_tl 3669 \tl_new:N \l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_type_tl 3670 \tl_new:N \l_zrefclever_type_first_label_tl
3671 \tl_new:N \l_zrefclever_type_first_label_type_tl
```

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

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

```
\l_zrefclever_name_in_link_bool 3672 \tl_new:N \l_zrefclever_type_name_tl
\l_zrefclever_type_name_missing_bool 3673 \bool_new:N \l_zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl 3674 \bool_new:N \l_zrefclever_type_name_missing_bool
\l_zrefclever_name_format_fallback_tl 3675 \tl_new:N \l_zrefclever_name_format_tl
\l_zrefclever_type_name_gender_seq 3676 \tl_new:N \l_zrefclever_name_format_fallback_tl
3677 \seq_new:N \l_zrefclever_type_name_gender_seq
```

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

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

Auxiliary variables for $\backslash_zrefclever_typeset_refs$: range handling.

```

3678 \int_new:N \l__zrefclever_range_count_int
3679 \int_new:N \l__zrefclever_range_same_count_int
3680 \tl_new:N \l__zrefclever_range_beg_label_tl
3681 \bool_new:N \l__zrefclever_range_beg_is_first_bool
3682 \tl_new:N \l__zrefclever_range_end_ref_tl
3683 \bool_new:N \l__zrefclever_next_maybe_range_bool
3684 \bool_new:N \l__zrefclever_next_is_same_bool

```

(End definition for $\backslash l_zrefclever_range_count_int$ and others.)

$\backslash l_zrefclever_tpairsep_tl$
 $\backslash l_zrefclever_tlistsep_tl$
 $\backslash l_zrefclever_tlastsep_tl$
 $\backslash l_zrefclever_namesep_tl$
 $\backslash l_zrefclever_pairsep_tl$
 $\backslash l_zrefclever_listsep_tl$
 $\backslash l_zrefclever_lastsep_tl$
 $\backslash l_zrefclever_rangesep_tl$
 $\backslash l_zrefclever_namefont_tl$
 $\backslash l_zrefclever_reffont_tl$
 $\backslash l_zrefclever_endrangefunc_tl$
 $\backslash l_zrefclever_endrangeprop_tl$
 $\backslash l_zrefclever_cap_bool$
 $\backslash l_zrefclever_abbrev_bool$
 $\backslash l_zrefclever_rangetopair_bool$

Auxiliary variables for $\backslash_zrefclever_typeset_refs$: separators, and font and other options.

```

3685 \tl_new:N \l__zrefclever_tpairsep_tl
3686 \tl_new:N \l__zrefclever_tlistsep_tl
3687 \tl_new:N \l__zrefclever_tlastsep_tl
3688 \tl_new:N \l__zrefclever_namesep_tl
3689 \tl_new:N \l__zrefclever_pairsep_tl
3690 \tl_new:N \l__zrefclever_listsep_tl
3691 \tl_new:N \l__zrefclever_lastsep_tl
3692 \tl_new:N \l__zrefclever_rangesep_tl
3693 \tl_new:N \l__zrefclever_namefont_tl
3694 \tl_new:N \l__zrefclever_reffont_tl
3695 \tl_new:N \l__zrefclever_endrangefunc_tl
3696 \tl_new:N \l__zrefclever_endrangeprop_tl
3697 \bool_new:N \l__zrefclever_cap_bool
3698 \bool_new:N \l__zrefclever_abbrev_bool
3699 \bool_new:N \l__zrefclever_rangetopair_bool

```

(End definition for $\backslash l_zrefclever_tpairsep_tl$ and others.)

$\backslash l_zrefclever_refbounds_first_seq$
 $\backslash l_zrefclever_refbounds_first_sg_seq$
 $\backslash l_zrefclever_refbounds_first_pb_seq$
 $\backslash l_zrefclever_refbounds_first_rb_seq$
 $\backslash l_zrefclever_refbounds_mid_seq$
 $\backslash l_zrefclever_refbounds_mid_rb_seq$
 $\backslash l_zrefclever_refbounds_mid_re_seq$
 $\backslash l_zrefclever_refbounds_last_seq$
 $\backslash l_zrefclever_refbounds_last_pe_seq$
 $\backslash l_zrefclever_refbounds_last_re_seq$
 $\backslash l_zrefclever_type_first_refbounds_seq$
 $\backslash l_zrefclever_type_first_refbounds_set_bool$

Auxiliary variables for $\backslash_zrefclever_typeset_refs$: advanced reference format options.

```

3700 \seq_new:N \l__zrefclever_refbounds_first_seq
3701 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3702 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3703 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3704 \seq_new:N \l__zrefclever_refbounds_mid_seq
3705 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3706 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3707 \seq_new:N \l__zrefclever_refbounds_last_seq
3708 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3709 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3710 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3711 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

(End definition for $\backslash l_zrefclever_refbounds_first_seq$ and others.)

$\backslash l_zrefclever_verbose_testing_bool$

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 $\backslash l_zrefclever_typeset_queue_curr_tl$.

```

3712 \bool_new:N \l__zrefclever_verbose_testing_bool

```

(End definition for $\backslash l_zrefclever_verbose_testing_bool$.)

Main functions

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

```

3713 \cs_new_protected:Npn \_zrefclever_typeset_refs:
3714 {
3715   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3716   \l__zrefclever_zcref_labels_seq
3717   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3718   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3719   \tl_clear:N \l__zrefclever_type_first_label_tl
3720   \tl_clear:N \l__zrefclever_type_first_label_type_tl
3721   \tl_clear:N \l__zrefclever_range_beg_label_tl
3722   \tl_clear:N \l__zrefclever_range_end_ref_tl
3723   \int_zero:N \l__zrefclever_label_count_int
3724   \int_zero:N \l__zrefclever_type_count_int
3725   \int_zero:N \l__zrefclever_ref_count_int
3726   \int_zero:N \l__zrefclever_range_count_int
3727   \int_zero:N \l__zrefclever_range_same_count_int
3728   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3729   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3730
3731   % Get type block options (not type-specific).
3732   \_zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3733   { \l__zrefclever_label_type_a_tl }
3734   { \l__zrefclever_ref_language_tl }
3735   \l__zrefclever_tpairsep_tl
3736   \_zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3737   { \l__zrefclever_label_type_a_tl }
3738   { \l__zrefclever_ref_language_tl }
3739   \l__zrefclever_tlistsep_tl
3740   \_zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3741   { \l__zrefclever_label_type_a_tl }
3742   { \l__zrefclever_ref_language_tl }
3743   \l__zrefclever_tlastsep_tl
3744
3745   % Process label stack.
3746   \bool_set_false:N \l__zrefclever_typeset_last_bool
3747   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3748   {
3749     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3750     \l__zrefclever_label_a_tl
3751     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3752     {
3753       \tl_clear:N \l__zrefclever_label_b_tl
3754       \bool_set_true:N \l__zrefclever_typeset_last_bool
3755     }
3756     {
3757       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3758       \l__zrefclever_label_b_tl
3759     }
3760
3761     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3762     {
3763       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }

```

```

3764 \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3765 }
3766 {
3767   \__zrefclever_extract_default:Nvnn
3768   \l__zrefclever_label_type_a_tl
3769   \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3770   \__zrefclever_extract_default:Nvnn
3771   \l__zrefclever_label_type_b_tl
3772   \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3773 }
3774
3775 % First, we establish whether the "current label" (i.e. 'a') is the
3776 % last one of its type. This can happen because the "next label"
3777 % (i.e. 'b') is of a different type (or different definition status),
3778 % or because we are at the end of the list.
3779 \bool_if:NTF \l__zrefclever_typeset_last_bool
3780 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3781 {
3782   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3783   {
3784     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3785     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3786     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3787   }
3788   {
3789     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3790     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3791     {
3792       % Neither is undefined, we must check the types.
3793       \tl_if_eq:NNTF
3794       \l__zrefclever_label_type_a_tl
3795       \l__zrefclever_label_type_b_tl
3796       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3797       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3798     }
3799   }
3800 }
3801
3802 % Handle warnings in case of reference or type undefined.
3803 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3804 \zref@refused { \l__zrefclever_label_a_tl }
3805 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3806 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3807 {}
3808 {
3809   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3810   {
3811     \msg_warning:nx { zref-clever } { missing-type }
3812     { \l__zrefclever_label_a_tl }
3813   }
3814   \zref@ifrefcontainsprop
3815   { \l__zrefclever_label_a_tl }
3816   { \l__zrefclever_ref_property_tl }
3817   { }

```

```

3818         {
3819             \msg_warning:nxxx { zref-clever } { missing-property }
3820             { \l__zrefclever_ref_property_tl }
3821             { \l__zrefclever_label_a_tl }
3822         }
3823     }
3824
3825 % Get possibly type-specific separators, refbounds, font and other
3826 % options, once per type.
3827 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
3828 {
3829     \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3830     { \l__zrefclever_label_type_a_tl }
3831     { \l__zrefclever_ref_language_tl }
3832     \l__zrefclever_namesep_tl
3833     \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3834     { \l__zrefclever_label_type_a_tl }
3835     { \l__zrefclever_ref_language_tl }
3836     \l__zrefclever_pairsep_tl
3837     \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3838     { \l__zrefclever_label_type_a_tl }
3839     { \l__zrefclever_ref_language_tl }
3840     \l__zrefclever_listsep_tl
3841     \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3842     { \l__zrefclever_label_type_a_tl }
3843     { \l__zrefclever_ref_language_tl }
3844     \l__zrefclever_lastsep_tl
3845     \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3846     { \l__zrefclever_label_type_a_tl }
3847     { \l__zrefclever_ref_language_tl }
3848     \l__zrefclever_rangesep_tl
3849     \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3850     { \l__zrefclever_label_type_a_tl }
3851     { \l__zrefclever_ref_language_tl }
3852     \l__zrefclever_namefont_tl
3853     \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3854     { \l__zrefclever_label_type_a_tl }
3855     { \l__zrefclever_ref_language_tl }
3856     \l__zrefclever_reffont_tl
3857     \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3858     { \l__zrefclever_label_type_a_tl }
3859     { \l__zrefclever_ref_language_tl }
3860     \l__zrefclever_endrangefunc_tl
3861     \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3862     { \l__zrefclever_label_type_a_tl }
3863     { \l__zrefclever_ref_language_tl }
3864     \l__zrefclever_endrangeprop_tl
3865     \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3866     { \l__zrefclever_label_type_a_tl }
3867     { \l__zrefclever_ref_language_tl }
3868     \l__zrefclever_cap_bool
3869     \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3870     { \l__zrefclever_label_type_a_tl }
3871     { \l__zrefclever_ref_language_tl }

```

```

3872         \l__zrefclever_abbrev_bool
3873     \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3874     { \l__zrefclever_label_type_a_tl }
3875     { \l__zrefclever_ref_language_tl }
3876     \l__zrefclever_rangetopair_bool
3877     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3878     { \l__zrefclever_label_type_a_tl }
3879     { \l__zrefclever_ref_language_tl }
3880     \l__zrefclever_refbounds_first_seq
3881     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }
3882     { \l__zrefclever_label_type_a_tl }
3883     { \l__zrefclever_ref_language_tl }
3884     \l__zrefclever_refbounds_first_sg_seq
3885     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3886     { \l__zrefclever_label_type_a_tl }
3887     { \l__zrefclever_ref_language_tl }
3888     \l__zrefclever_refbounds_first_pb_seq
3889     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3890     { \l__zrefclever_label_type_a_tl }
3891     { \l__zrefclever_ref_language_tl }
3892     \l__zrefclever_refbounds_first_rb_seq
3893     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3894     { \l__zrefclever_label_type_a_tl }
3895     { \l__zrefclever_ref_language_tl }
3896     \l__zrefclever_refbounds_mid_seq
3897     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3898     { \l__zrefclever_label_type_a_tl }
3899     { \l__zrefclever_ref_language_tl }
3900     \l__zrefclever_refbounds_mid_rb_seq
3901     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3902     { \l__zrefclever_label_type_a_tl }
3903     { \l__zrefclever_ref_language_tl }
3904     \l__zrefclever_refbounds_mid_re_seq
3905     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3906     { \l__zrefclever_label_type_a_tl }
3907     { \l__zrefclever_ref_language_tl }
3908     \l__zrefclever_refbounds_last_seq
3909     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3910     { \l__zrefclever_label_type_a_tl }
3911     { \l__zrefclever_ref_language_tl }
3912     \l__zrefclever_refbounds_last_pe_seq
3913     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3914     { \l__zrefclever_label_type_a_tl }
3915     { \l__zrefclever_ref_language_tl }
3916     \l__zrefclever_refbounds_last_re_seq
3917 }
3918
3919 % Here we send this to a couple of auxiliary functions.
3920 \bool_if:NTF \l__zrefclever_last_of_type_bool
3921 % There exists no next label of the same type as the current.
3922 { \__zrefclever_typeset_refs_last_of_type: }
3923 % There exists a next label of the same type as the current.
3924 { \__zrefclever_typeset_refs_not_last_of_type: }
3925 }

```


3926 }

(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.

```

3927 \cs_new_protected:Npn \_zrefclever_typeset_refs_last_of_type:
3928 {
3929   % Process the current label to the current queue.
3930   \int_case:nnF { \l__zrefclever_label_count_int }
3931   {
3932     % It is the last label of its type, but also the first one, and that's
3933     % what matters here: just store it.
3934     % Test: 'zc-typeset01.lvt': "Last of type: single"
3935     { 0 }
3936     {
3937       \tl_set:NV \l__zrefclever_type_first_label_tl
3938       \l__zrefclever_label_a_tl
3939       \tl_set:NV \l__zrefclever_type_first_label_type_tl
3940       \l__zrefclever_label_type_a_tl
3941       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3942       \l__zrefclever_refbounds_first_sg_seq
3943       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3944     }
3945
3946     % The last is the second: we have a pair (if not repeated).
3947     % Test: 'zc-typeset01.lvt': "Last of type: pair"
3948     { 1 }
3949     {
3950       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
3951       {
3952         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3953         \l__zrefclever_refbounds_first_sg_seq
3954         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3955       }
3956       {
3957         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3958         {
3959           \exp_not:V \l__zrefclever_pairsep_tl
3960           \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3961           \l__zrefclever_refbounds_last_pe_seq
3962         }
3963         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3964         \l__zrefclever_refbounds_first_pb_seq
3965         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3966       }

```

```

3967     }
3968   }
3969   % Last is third or more of its type: without repetition, we'd have the
3970   % last element on a list, but control for possible repetition.
3971   {
3972     \int_case:nnF { \l__zrefclever_range_count_int }
3973     {
3974       % There was no range going on.
3975       % Test: 'zc-typeset01.lvt': "Last of type: not range"
3976       { 0 }
3977       {
3978         \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
3979         {
3980           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3981             {
3982               \exp_not:V \l__zrefclever_pairsep_tl
3983               \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3984               \l__zrefclever_refbounds_last_pe_seq
3985             }
3986         }
3987         {
3988           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3989             {
3990               \exp_not:V \l__zrefclever_lastsep_tl
3991               \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3992               \l__zrefclever_refbounds_last_seq
3993             }
3994         }
3995       }
3996       % Last in the range is also the second in it.
3997       % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
3998       { 1 }
3999       {
4000         \int_compare:nNnTF
4001         { \l__zrefclever_range_same_count_int } = { 1 }
4002         {
4003           % We know 'range_beg_is_first_bool' is false, since this is
4004           % the second element in the range, but the third or more in
4005           % the type list.
4006           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4007             {
4008               \exp_not:V \l__zrefclever_pairsep_tl
4009               \__zrefclever_get_ref:VN
4010               \l__zrefclever_range_beg_label_tl
4011               \l__zrefclever_refbounds_last_pe_seq
4012             }
4013           \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4014             \l__zrefclever_refbounds_first_pb_seq
4015           \bool_set_true:N
4016             \l__zrefclever_type_first_refbounds_set_bool
4017         }
4018         {
4019           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4020             {

```

```

4021         \exp_not:V \l__zrefclever_listsep_tl
4022         \__zrefclever_get_ref:VN
4023         \l__zrefclever_range_beg_label_tl
4024         \l__zrefclever_refbounds_mid_seq
4025         \exp_not:V \l__zrefclever_lastsep_tl
4026         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4027         \l__zrefclever_refbounds_last_seq
4028     }
4029 }
4030 }
4031 }
4032 % Last in the range is third or more in it.
4033 {
4034     \int_case:nnF
4035     {
4036         \l__zrefclever_range_count_int -
4037         \l__zrefclever_range_same_count_int
4038     }
4039     {
4040         % Repetition, not a range.
4041         % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4042         { 0 }
4043         {
4044             % If 'range_beg_is_first_bool' is true, it means it was also
4045             % the first of the type, and hence its typesetting was
4046             % already handled, and we just have to set refbounds.
4047             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4048             {
4049                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4050                 \l__zrefclever_refbounds_first_sg_seq
4051                 \bool_set_true:N
4052                 \l__zrefclever_type_first_refbounds_set_bool
4053             }
4054             {
4055                 \int_compare:nNnTF
4056                 { \l__zrefclever_ref_count_int } < { 2 }
4057                 {
4058                     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4059                     {
4060                         \exp_not:V \l__zrefclever_pairsep_tl
4061                         \__zrefclever_get_ref:VN
4062                         \l__zrefclever_range_beg_label_tl
4063                         \l__zrefclever_refbounds_last_pe_seq
4064                     }
4065                 }
4066                 {
4067                     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4068                     {
4069                         \exp_not:V \l__zrefclever_lastsep_tl
4070                         \__zrefclever_get_ref:VN
4071                         \l__zrefclever_range_beg_label_tl
4072                         \l__zrefclever_refbounds_last_seq
4073                     }
4074                 }

```

```

4075     }
4076 }
4077 % A 'range', but with no skipped value, treat as pair if range
4078 % started with first of type, otherwise as list.
4079 % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4080 { 1 }
4081 {
4082     % Ditto.
4083     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4084     {
4085         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4086         \l__zrefclever_refbounds_first_pb_seq
4087         \bool_set_true:N
4088         \l__zrefclever_type_first_refbounds_set_bool
4089         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4090         {
4091             \exp_not:V \l__zrefclever_pairsep_tl
4092             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4093             \l__zrefclever_refbounds_last_pe_seq
4094         }
4095     }
4096     {
4097         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4098         {
4099             \exp_not:V \l__zrefclever_listsep_tl
4100             \__zrefclever_get_ref:VN
4101             \l__zrefclever_range_beg_label_tl
4102             \l__zrefclever_refbounds_mid_seq
4103         }
4104         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4105         {
4106             \exp_not:V \l__zrefclever_lastsep_tl
4107             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4108             \l__zrefclever_refbounds_last_seq
4109         }
4110     }
4111 }
4112 }
4113 {
4114     % An actual range.
4115     % Test: 'zc-typeset01.lvt': "Last of type: range"
4116     % Ditto.
4117     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4118     {
4119         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4120         \l__zrefclever_refbounds_first_rb_seq
4121         \bool_set_true:N
4122         \l__zrefclever_type_first_refbounds_set_bool
4123     }
4124     {
4125         \int_compare:nNnTF
4126         { \l__zrefclever_ref_count_int } < { 2 }
4127         {
4128             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

4129         {
4130             \exp_not:V \l__zrefclever_pairsep_tl
4131             \__zrefclever_get_ref:VN
4132             \l__zrefclever_range_beg_label_tl
4133             \l__zrefclever_refbounds_mid_rb_seq
4134         }
4135         \seq_set_eq:NN
4136         \l__zrefclever_type_first_refbounds_seq
4137         \l__zrefclever_refbounds_first_pb_seq
4138         \bool_set_true:N
4139         \l__zrefclever_type_first_refbounds_set_bool
4140     }
4141     {
4142         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4143         {
4144             \exp_not:V \l__zrefclever_lastsep_tl
4145             \__zrefclever_get_ref:VN
4146             \l__zrefclever_range_beg_label_tl
4147             \l__zrefclever_refbounds_mid_rb_seq
4148         }
4149     }
4150 }
4151 \bool_lazy_and:nnTF
4152 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4153 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4154 {
4155     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4156     \l__zrefclever_range_beg_label_tl
4157     \l__zrefclever_label_a_tl
4158     \l__zrefclever_range_end_ref_tl
4159     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4160     {
4161         \exp_not:V \l__zrefclever_rangesep_tl
4162         \__zrefclever_get_ref_endrange:VVN
4163         \l__zrefclever_label_a_tl
4164         \l__zrefclever_range_end_ref_tl
4165         \l__zrefclever_refbounds_last_re_seq
4166     }
4167 }
4168 {
4169     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4170     {
4171         \exp_not:V \l__zrefclever_rangesep_tl
4172         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4173         \l__zrefclever_refbounds_last_re_seq
4174     }
4175 }
4176 }
4177 }
4178 }
4179
4180 % Handle "range" option. The idea is simple: if the queue is not empty,
4181 % we replace it with the end of the range (or pair). We can still
4182 % retrieve the end of the range from 'label_a' since we know to be

```

```

4183 % processing the last label of its type at this point.
4184 \bool_if:NT \l__zrefclever_typeset_range_bool
4185 {
4186   \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4187   {
4188     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4189     { }
4190     {
4191       \msg_warning:nxx { zref-clever } { single-element-range }
4192       { \l__zrefclever_type_first_label_type_tl }
4193     }
4194   }
4195   {
4196     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4197     \bool_if:NT \l__zrefclever_rangetopair_bool
4198     {
4199       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4200       { }
4201       {
4202         \__zrefclever_labels_in_sequence:nn
4203         { \l__zrefclever_type_first_label_tl }
4204         { \l__zrefclever_label_a_tl }
4205       }
4206     }
4207     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4208     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4209     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4210     {
4211       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4212       {
4213         \exp_not:V \l__zrefclever_pairsep_tl
4214         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4215         \l__zrefclever_refbounds_last_pe_seq
4216       }
4217       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4218       \l__zrefclever_refbounds_first_pb_seq
4219       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4220     }
4221     {
4222       \bool_lazy_and:nnTF
4223       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4224       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4225       {
4226         % We must get 'type_first_label_tl' instead of
4227         % 'range_beg_label_tl' here, since it is not necessary
4228         % that the first of type was actually starting a range for
4229         % the 'range' option to be used.
4230         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4231         \l__zrefclever_type_first_label_tl
4232         \l__zrefclever_label_a_tl
4233         \l__zrefclever_range_end_ref_tl
4234         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4235         {
4236           \exp_not:V \l__zrefclever_rangesep_tl

```

```

4237         \_zrefclever_get_ref_endrange:VVN
4238         \l__zrefclever_label_a_tl
4239         \l__zrefclever_range_end_ref_tl
4240         \l__zrefclever_refbounds_last_re_seq
4241     }
4242 }
4243 {
4244     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4245     {
4246         \exp_not:V \l__zrefclever_rangeseq_tl
4247         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4248         \l__zrefclever_refbounds_last_re_seq
4249     }
4250 }
4251 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4252 \l__zrefclever_refbounds_first_rb_seq
4253 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4254 }
4255 }
4256 }
4257
4258 % If none of the special cases for the first of type refbounds have been
4259 % set, do it.
4260 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4261 {
4262     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4263     \l__zrefclever_refbounds_first_seq
4264 }
4265
4266 % Now that the type block is finished, we can add the name and the first
4267 % ref to the queue. Also, if "typeset" option is not "both", handle it
4268 % here as well.
4269 \_zrefclever_type_name_setup:
4270 \bool_if:NTF
4271 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4272 {
4273     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4274     { \_zrefclever_get_ref_first: }
4275 }
4276 {
4277     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4278     {
4279         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4280         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4281         {
4282             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4283             \l__zrefclever_type_first_refbounds_seq
4284         }
4285     }
4286     {
4287         \bool_if:NTF \l__zrefclever_typeset_name_bool
4288         {
4289             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4290             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

4291 {
4292   \bool_if:NTF \l__zrefclever_name_in_link_bool
4293   {
4294     \exp_not:N \group_begin:
4295     \exp_not:V \l__zrefclever_namefont_tl
4296     % It's two '@s', but escaped for DocStrip.
4297     \exp_not:N \hyper@@link
4298     {
4299       \__zrefclever_extract_url_unexp:V
4300       \l__zrefclever_type_first_label_tl
4301     }
4302     {
4303       \__zrefclever_extract_unexp:Vnn
4304       \l__zrefclever_type_first_label_tl
4305       { anchor } { }
4306     }
4307     { \exp_not:V \l__zrefclever_type_name_tl }
4308     \exp_not:N \group_end:
4309   }
4310   {
4311     \exp_not:N \group_begin:
4312     \exp_not:V \l__zrefclever_namefont_tl
4313     \exp_not:V \l__zrefclever_type_name_tl
4314     \exp_not:N \group_end:
4315   }
4316 }
4317 }
4318 {
4319   % Logically, this case would correspond to "typeset=none", but
4320   % it should not occur, given that the options are set up to
4321   % typeset either "ref" or "name". Still, leave here a
4322   % sensible fallback, equal to the behavior of "both".
4323   % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4324   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4325   { \__zrefclever_get_ref_first: }
4326 }
4327 }
4328 }
4329
4330 % Typeset the previous type block, if there is one.
4331 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4332 {
4333   \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4334   { \l__zrefclever_tlistsep_tl }
4335   \l__zrefclever_typeset_queue_prev_tl
4336 }
4337
4338 % Extra log for testing.
4339 \bool_if:NT \l__zrefclever_verbose_testing_bool
4340 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4341
4342 % Wrap up loop, or prepare for next iteration.
4343 \bool_if:NTF \l__zrefclever_typeset_last_bool
4344 {

```



```

4345 % We are finishing, typeset the current queue.
4346 \int_case:nnF { \l__zrefclever_type_count_int }
4347 {
4348   % Single type.
4349   % Test: 'zc-typeset01.lvt': "Last of type: single type"
4350   { 0 }
4351   { \l__zrefclever_typeset_queue_curr_tl }
4352   % Pair of types.
4353   % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4354   { 1 }
4355   {
4356     \l__zrefclever_tpairsep_tl
4357     \l__zrefclever_typeset_queue_curr_tl
4358   }
4359 }
4360 {
4361   % Last in list of types.
4362   % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4363   \l__zrefclever_tlastsep_tl
4364   \l__zrefclever_typeset_queue_curr_tl
4365 }
4366 % And nudge in case of multitype reference.
4367 \bool_lazy_all:nT
4368 {
4369   { \l__zrefclever_nudge_enabled_bool }
4370   { \l__zrefclever_nudge_multitype_bool }
4371   { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4372 }
4373 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4374 }
4375 {
4376   % There are further labels, set variables for next iteration.
4377   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4378     \l__zrefclever_typeset_queue_curr_tl
4379   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4380   \tl_clear:N \l__zrefclever_type_first_label_tl
4381   \tl_clear:N \l__zrefclever_type_first_label_type_tl
4382   \tl_clear:N \l__zrefclever_range_beg_label_tl
4383   \tl_clear:N \l__zrefclever_range_end_ref_tl
4384   \int_zero:N \l__zrefclever_label_count_int
4385   \int_zero:N \l__zrefclever_ref_count_int
4386   \int_incr:N \l__zrefclever_type_count_int
4387   \int_zero:N \l__zrefclever_range_count_int
4388   \int_zero:N \l__zrefclever_range_same_count_int
4389   \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4390   \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4391 }
4392 }

```

(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.

```

4393 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4394 {

```

```

4395 % Signal if next label may form a range with the current one (only
4396 % considered if compression is enabled in the first place).
4397 \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4398 \bool_set_false:N \l__zrefclever_next_is_same_bool
4399 \bool_if:NT \l__zrefclever_typeset_compress_bool
4400 {
4401   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4402   { }
4403   {
4404     \__zrefclever_labels_in_sequence:nn
4405     { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
4406   }
4407 }
4408
4409 % Process the current label to the current queue.
4410 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4411 {
4412   % Current label is the first of its type (also not the last, but it
4413   % doesn't matter here): just store the label.
4414   \tl_set:NV \l__zrefclever_type_first_label_tl
4415   \l__zrefclever_label_a_tl
4416   \tl_set:NV \l__zrefclever_type_first_label_type_tl
4417   \l__zrefclever_label_type_a_tl
4418   \int_incr:N \l__zrefclever_ref_count_int
4419
4420   % If the next label may be part of a range, signal it (we deal with it
4421   % as the "first", and must do it there, to handle hyperlinking), but
4422   % also step the range counters.
4423   % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4424   \bool_if:NT \l__zrefclever_next_maybe_range_bool
4425   {
4426     \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4427     \tl_set:NV \l__zrefclever_range_beg_label_tl
4428     \l__zrefclever_label_a_tl
4429     \tl_clear:N \l__zrefclever_range_end_ref_tl
4430     \int_incr:N \l__zrefclever_range_count_int
4431     \bool_if:NT \l__zrefclever_next_is_same_bool
4432     { \int_incr:N \l__zrefclever_range_same_count_int }
4433   }
4434 }
4435 {
4436   % Current label is neither the first (nor the last) of its type.
4437   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4438   {
4439     % Starting, or continuing a range.
4440     \int_compare:nNnTF
4441     { \l__zrefclever_range_count_int } = { 0 }
4442     {
4443       % There was no range going, we are starting one.
4444       \tl_set:NV \l__zrefclever_range_beg_label_tl
4445       \l__zrefclever_label_a_tl
4446       \tl_clear:N \l__zrefclever_range_end_ref_tl
4447       \int_incr:N \l__zrefclever_range_count_int
4448       \bool_if:NT \l__zrefclever_next_is_same_bool

```

```

4449         { \int_incr:N \l__zrefclever_range_same_count_int }
4450     }
4451     {
4452         % Second or more in the range, but not the last.
4453         \int_incr:N \l__zrefclever_range_count_int
4454         \bool_if:NT \l__zrefclever_next_is_same_bool
4455         { \int_incr:N \l__zrefclever_range_same_count_int }
4456     }
4457 }
4458 {
4459     % Next element is not in sequence: there was no range, or we are
4460     % closing one.
4461     \int_case:nnF { \l__zrefclever_range_count_int }
4462     {
4463         % There was no range going on.
4464         % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4465         { 0 }
4466         {
4467             \int_incr:N \l__zrefclever_ref_count_int
4468             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4469             {
4470                 \exp_not:V \l__zrefclever_listsep_tl
4471                 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4472                 \l__zrefclever_refbounds_mid_seq
4473             }
4474         }
4475         % Last is second in the range: if 'range_same_count' is also
4476         % '1', it's a repetition (drop it), otherwise, it's a "pair
4477         % within a list", treat as list.
4478         % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4479         % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4480         { 1 }
4481         {
4482             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4483             {
4484                 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4485                 \l__zrefclever_refbounds_first_seq
4486                 \bool_set_true:N
4487                 \l__zrefclever_type_first_refbounds_set_bool
4488             }
4489             {
4490                 \int_incr:N \l__zrefclever_ref_count_int
4491                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4492                 {
4493                     \exp_not:V \l__zrefclever_listsep_tl
4494                     \__zrefclever_get_ref:VN
4495                     \l__zrefclever_range_beg_label_tl
4496                     \l__zrefclever_refbounds_mid_seq
4497                 }
4498             }
4499             \int_compare:nNnF
4500             { \l__zrefclever_range_same_count_int } = { 1 }
4501             {
4502                 \int_incr:N \l__zrefclever_ref_count_int

```

```

4503         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4504         {
4505             \exp_not:V \l__zrefclever_listsep_tl
4506             \__zrefclever_get_ref:VN
4507             \l__zrefclever_label_a_tl
4508             \l__zrefclever_refbounds_mid_seq
4509         }
4510     }
4511 }
4512 }
4513 {
4514     % Last is third or more in the range: if 'range_count' and
4515     % 'range_same_count' are the same, its a repetition (drop it),
4516     % if they differ by '1', its a list, if they differ by more,
4517     % it is a real range.
4518     \int_case:nnF
4519     {
4520         \l__zrefclever_range_count_int -
4521         \l__zrefclever_range_same_count_int
4522     }
4523     {
4524         % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4525         { 0 }
4526         {
4527             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4528             {
4529                 \seq_set_eq:NN
4530                 \l__zrefclever_type_first_refbounds_seq
4531                 \l__zrefclever_refbounds_first_seq
4532                 \bool_set_true:N
4533                 \l__zrefclever_type_first_refbounds_set_bool
4534             }
4535             {
4536                 \int_incr:N \l__zrefclever_ref_count_int
4537                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4538                 {
4539                     \exp_not:V \l__zrefclever_listsep_tl
4540                     \__zrefclever_get_ref:VN
4541                     \l__zrefclever_range_beg_label_tl
4542                     \l__zrefclever_refbounds_mid_seq
4543                 }
4544             }
4545         }
4546         % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4547         { 1 }
4548         {
4549             \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4550             {
4551                 \seq_set_eq:NN
4552                 \l__zrefclever_type_first_refbounds_seq
4553                 \l__zrefclever_refbounds_first_seq
4554                 \bool_set_true:N
4555                 \l__zrefclever_type_first_refbounds_set_bool
4556             }

```

```

4557 {
4558   \int_incr:N \l__zrefclever_ref_count_int
4559   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4560   {
4561     \exp_not:V \l__zrefclever_listsep_tl
4562     \__zrefclever_get_ref:VN
4563     \l__zrefclever_range_beg_label_tl
4564     \l__zrefclever_refbounds_mid_seq
4565   }
4566 }
4567 \int_incr:N \l__zrefclever_ref_count_int
4568 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4569 {
4570   \exp_not:V \l__zrefclever_listsep_tl
4571   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4572   \l__zrefclever_refbounds_mid_seq
4573 }
4574 }
4575 }
4576 {
4577   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4578   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4579   {
4580     \seq_set_eq:NN
4581     \l__zrefclever_type_first_refbounds_seq
4582     \l__zrefclever_refbounds_first_rb_seq
4583     \bool_set_true:N
4584     \l__zrefclever_type_first_refbounds_set_bool
4585   }
4586   {
4587     \int_incr:N \l__zrefclever_ref_count_int
4588     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4589     {
4590       \exp_not:V \l__zrefclever_listsep_tl
4591       \__zrefclever_get_ref:VN
4592       \l__zrefclever_range_beg_label_tl
4593       \l__zrefclever_refbounds_mid_rb_seq
4594     }
4595   }
4596   % For the purposes of the serial comma, and thus for the
4597   % distinction of 'lastsep' and 'pairsep', a "range" counts
4598   % as one. Since 'range_beg' has already been counted
4599   % (here or with the first of type), we refrain from
4600   % incrementing 'ref_count_int'.
4601   \bool_lazy_and:nnTF
4602   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4603   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4604   {
4605     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4606     \l__zrefclever_range_beg_label_tl
4607     \l__zrefclever_label_a_tl
4608     \l__zrefclever_range_end_ref_tl
4609     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4610     {

```

```

4611             \exp_not:V \l__zrefclever_rangesep_tl
4612             \__zrefclever_get_ref_endrange:VVN
4613             \l__zrefclever_label_a_tl
4614             \l__zrefclever_range_end_ref_tl
4615             \l__zrefclever_refbounds_mid_re_seq
4616         }
4617     }
4618     {
4619         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4620         {
4621             \exp_not:V \l__zrefclever_rangesep_tl
4622             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4623             \l__zrefclever_refbounds_mid_re_seq
4624         }
4625     }
4626 }
4627 }
4628 % Reset counters.
4629 \int_zero:N \l__zrefclever_range_count_int
4630 \int_zero:N \l__zrefclever_range_same_count_int
4631 }
4632 }
4633 % Step label counter for next iteration.
4634 \int_incr:N \l__zrefclever_label_count_int
4635 }

```

(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 nic 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 \mathbf{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 \mathbf{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.

```

4636 \cs_new_protected:Npn \__zrefclever_ref_default:
4637 { \zref@default }
4638 \cs_new_protected:Npn \__zrefclever_name_default:
4639 { \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 ref-bounds, and hyperlinking. For use with all labels, except the first of its type, which is done by `__zrefclever_get_ref_first:`, and the last of a range, which is done by `__zrefclever_get_ref_endrange:nnN`.

```

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

```

```

4640 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4641 {
4642   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4643   {
4644     \bool_if:nTF
4645     {
4646       \l__zrefclever_hyperlink_bool &&
4647       ! \l__zrefclever_link_star_bool
4648     }
4649     {
4650       \exp_not:N \group_begin:
4651       \exp_not:N \l__zrefclever_reffont_tl
4652       \seq_item:Nn #2 { 1 }
4653       % It's two '@s', but escaped for DocStrip.
4654       \exp_not:N \hyper@@link
4655       { \__zrefclever_extract_url_unexp:n {#1} }
4656       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4657       {
4658         \seq_item:Nn #2 { 2 }
4659         \__zrefclever_extract_unexp:nvn {#1}
4660         { \l__zrefclever_ref_property_tl } { }
4661         \seq_item:Nn #2 { 3 }
4662       }
4663       \seq_item:Nn #2 { 4 }
4664       \exp_not:N \group_end:
4665     }
4666     {
4667       \exp_not:N \group_begin:
4668       \exp_not:N \l__zrefclever_reffont_tl
4669       \seq_item:Nn #2 { 1 }
4670       \seq_item:Nn #2 { 2 }
4671       \__zrefclever_extract_unexp:nvn {#1}
4672       { \l__zrefclever_ref_property_tl } { }
4673       \seq_item:Nn #2 { 3 }
4674       \seq_item:Nn #2 { 4 }
4675       \exp_not:N \group_end:
4676     }
4677   }

```

```

4678     { \_zrefclever_ref_default: }
4679   }
4680 \cs_generate_variant:Nn \_zrefclever_get_ref:nN { VN }

```

(End definition for _zrefclever_get_ref:nN.)

```

\_zrefclever_get_ref_endrange:nnN    \_zrefclever_get_ref_endrange:nnN {\label}\{reference}\{refbounds}
4681 \cs_new:Npn \_zrefclever_get_ref_endrange:nnN #1#2#3
4682 {
4683   \str_if_eq:nnTF {#2} { zc@missingproperty }
4684   { \_zrefclever_ref_default: }
4685   {
4686     \bool_if:nTF
4687     {
4688       \l_zrefclever_hyperlink_bool &&
4689       ! \l_zrefclever_link_star_bool
4690     }
4691     {
4692       \exp_not:N \group_begin:
4693       \exp_not:N \l_zrefclever_reffont_tl
4694       \seq_item:Nn #3 { 1 }
4695       % It's two 's's', but escaped for DocStrip.
4696       \exp_not:N \hyper@@link
4697       { \_zrefclever_extract_url_unexp:n {#1} }
4698       { \_zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4699       {
4700         \seq_item:Nn #3 { 2 }
4701         \exp_not:n {#2}
4702         \seq_item:Nn #3 { 3 }
4703       }
4704       \seq_item:Nn #3 { 4 }
4705       \exp_not:N \group_end:
4706     }
4707     {
4708       \exp_not:N \group_begin:
4709       \exp_not:N \l_zrefclever_reffont_tl
4710       \seq_item:Nn #3 { 1 }
4711       \seq_item:Nn #3 { 2 }
4712       \exp_not:n {#2}
4713       \seq_item:Nn #3 { 3 }
4714       \seq_item:Nn #3 { 4 }
4715       \exp_not:N \group_end:
4716     }
4717   }
4718 }
4719 \cs_generate_variant:Nn \_zrefclever_get_ref_endrange:nnN { VVN }

```

(End definition for _zrefclever_get_ref_endrange:nnN.)

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

is `\l__zrefclever_type_first_label_tl`, but it also expected to be called right after `__zrefclever_type_name_setup:` which sets `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool` which it uses.

```

4720 \cs_new:Npn \__zrefclever_get_ref_first:
4721 {
4722   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4723   { \__zrefclever_ref_default: }
4724   {
4725     \bool_if:NTF \l__zrefclever_name_in_link_bool
4726     {
4727       \zref@ifrefcontainsprop
4728       { \l__zrefclever_type_first_label_tl }
4729       { \l__zrefclever_ref_property_tl }
4730       {
4731         \exp_not:N \group_begin:
4732         % It's two '@s', but escaped for DocStrip.
4733         \exp_not:N \hyper@@link
4734         {
4735           \__zrefclever_extract_url_unexp:V
4736           \l__zrefclever_type_first_label_tl
4737         }
4738         {
4739           \__zrefclever_extract_unexp:Vnn
4740           \l__zrefclever_type_first_label_tl { anchor } { }
4741         }
4742         {
4743           \exp_not:N \group_begin:
4744           \exp_not:V \l__zrefclever_namefont_tl
4745           \exp_not:V \l__zrefclever_type_name_tl
4746           \exp_not:N \group_end:
4747           \exp_not:V \l__zrefclever_namesep_tl
4748           \exp_not:N \group_begin:
4749           \exp_not:V \l__zrefclever_reffont_tl
4750           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4751           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4752           \__zrefclever_extract_unexp:Vvn
4753           \l__zrefclever_type_first_label_tl
4754           { \l__zrefclever_ref_property_tl } { }
4755           \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4756           \exp_not:N \group_end:
4757         }
4758         \exp_not:V \l__zrefclever_reffont_tl
4759         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4760         \exp_not:N \group_end:
4761       }
4762       {
4763         \exp_not:N \group_begin:
4764         \exp_not:V \l__zrefclever_namefont_tl
4765         \exp_not:V \l__zrefclever_type_name_tl
4766         \exp_not:N \group_end:
4767         \exp_not:V \l__zrefclever_namesep_tl
4768         \__zrefclever_ref_default:
4769       }
4770     }

```

```

4771 {
4772   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4773   {
4774     \__zrefclever_name_default:
4775     \exp_not:V \l__zrefclever_namesep_tl
4776   }
4777   {
4778     \exp_not:N \group_begin:
4779     \exp_not:V \l__zrefclever_namefont_tl
4780     \exp_not:V \l__zrefclever_type_name_tl
4781     \exp_not:N \group_end:
4782     \tl_if_empty:NF \l__zrefclever_type_name_tl
4783     { \exp_not:V \l__zrefclever_namesep_tl }
4784   }
4785   \zref@ifrefcontainsprop
4786   { \l__zrefclever_type_first_label_tl }
4787   { \l__zrefclever_ref_property_tl }
4788   {
4789     \bool_if:nTF
4790     {
4791       \l__zrefclever_hyperlink_bool &&
4792       ! \l__zrefclever_link_star_bool
4793     }
4794     {
4795       \exp_not:N \group_begin:
4796       \exp_not:V \l__zrefclever_reffont_tl
4797       \seq_item:Nn
4798       \l__zrefclever_type_first_refbounds_seq { 1 }
4799       % It's two '@s', but escaped for DocStrip.
4800       \exp_not:N \hyper@@link
4801       {
4802         \__zrefclever_extract_url_unexp:V
4803         \l__zrefclever_type_first_label_tl
4804       }
4805       {
4806         \__zrefclever_extract_unexp:Vnn
4807         \l__zrefclever_type_first_label_tl { anchor } { }
4808       }
4809       {
4810         \seq_item:Nn
4811         \l__zrefclever_type_first_refbounds_seq { 2 }
4812         \__zrefclever_extract_unexp:Vnn
4813         \l__zrefclever_type_first_label_tl
4814         { \l__zrefclever_ref_property_tl } { }
4815         \seq_item:Nn
4816         \l__zrefclever_type_first_refbounds_seq { 3 }
4817       }
4818       \seq_item:Nn
4819       \l__zrefclever_type_first_refbounds_seq { 4 }
4820       \exp_not:N \group_end:
4821     }
4822     {
4823       \exp_not:N \group_begin:
4824       \exp_not:V \l__zrefclever_reffont_tl

```

```

4825         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4826         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4827         \__zrefclever_extract_unexp:Vvn
4828         \l__zrefclever_type_first_label_tl
4829         { \l__zrefclever_ref_property_tl } { }
4830         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4831         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4832         \exp_not:N \group_end:
4833     }
4834 }
4835 { \__zrefclever_ref_default: }
4836 }
4837 }
4838 }

```

(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.

```

4839 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4840 {
4841     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4842     {
4843         \tl_clear:N \l__zrefclever_type_name_tl
4844         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4845     }
4846     {
4847         \tl_if_eq:NnTF
4848         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4849         {
4850             \tl_clear:N \l__zrefclever_type_name_tl
4851             \bool_set_true:N \l__zrefclever_type_name_missing_bool
4852         }
4853         {
4854             % Determine whether we should use capitalization, abbreviation,
4855             % and plural.
4856             \bool_lazy_or:nnTF
4857             { \l__zrefclever_cap_bool }
4858             {
4859                 \l__zrefclever_capfirst_bool &&
4860                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4861             }
4862             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4863             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }

```

```

4864 % If the queue is empty, we have a singular, otherwise, plural.
4865 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4866 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4867 { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4868 \bool_lazy_and:nnTF
4869 { \l__zrefclever_abbrev_bool }
4870 {
4871   ! \int_compare_p:nNn
4872     { \l__zrefclever_type_count_int } = { 0 } ||
4873   ! \l__zrefclever_noabbrev_first_bool
4874 }
4875 {
4876   \tl_set:NV \l__zrefclever_name_format_fallback_tl
4877     \l__zrefclever_name_format_tl
4878   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4879 }
4880 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4881
4882 % Handle number and gender nudges.
4883 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4884 {
4885   \bool_if:NTF \l__zrefclever_nudge_singular_bool
4886   {
4887     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4888     {
4889       \msg_warning:nnx { zref-clever }
4890       { nudge-plural-when-sg }
4891       { \l__zrefclever_type_first_label_type_tl }
4892     }
4893   }
4894   {
4895     \bool_lazy_all:nT
4896     {
4897       { \l__zrefclever_nudge_comptosing_bool }
4898       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4899       {
4900         \int_compare_p:nNn
4901           { \l__zrefclever_label_count_int } > { 0 }
4902       }
4903     }
4904     {
4905       \msg_warning:nnx { zref-clever }
4906       { nudge-comptosing }
4907       { \l__zrefclever_type_first_label_type_tl }
4908     }
4909   }
4910   \bool_lazy_and:nnT
4911   { \l__zrefclever_nudge_gender_bool }
4912   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4913   {
4914     \__zrefclever_get_rf_opt_seq:nxxN { gender }
4915     { \l__zrefclever_type_first_label_type_tl }
4916     { \l__zrefclever_ref_language_tl }
4917     \l__zrefclever_type_name_gender_seq

```

```

4918 \seq_if_in:NVF
4919 \l__zrefclever_type_name_gender_seq
4920 \l__zrefclever_ref_gender_tl
4921 {
4922   \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4923   {
4924     \msg_warning:nnxxx { zref-clever }
4925     { nudge-gender-not-declared-for-type }
4926     { \l__zrefclever_ref_gender_tl }
4927     { \l__zrefclever_type_first_label_type_tl }
4928     { \l__zrefclever_ref_language_tl }
4929   }
4930   {
4931     \msg_warning:nnxxxx { zref-clever }
4932     { nudge-gender-mismatch }
4933     { \l__zrefclever_type_first_label_type_tl }
4934     { \l__zrefclever_ref_gender_tl }
4935     {
4936       \seq_use:Nn
4937       \l__zrefclever_type_name_gender_seq { ,~ }
4938     }
4939     { \l__zrefclever_ref_language_tl }
4940   }
4941 }
4942 }
4943 }
4944
4945 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4946 {
4947   \__zrefclever_opt_tl_get:cNF
4948   {
4949     \__zrefclever_opt_varname_type:een
4950     { \l__zrefclever_type_first_label_type_tl }
4951     { \l__zrefclever_name_format_tl }
4952     { tl }
4953   }
4954   \l__zrefclever_type_name_tl
4955   {
4956     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
4957     {
4958       \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
4959       \tl_put_left:NV \l__zrefclever_name_format_tl
4960       \l__zrefclever_ref_decl_case_tl
4961     }
4962     \__zrefclever_opt_tl_get:cNF
4963     {
4964       \__zrefclever_opt_varname_lang_type:eeen
4965       { \l__zrefclever_ref_language_tl }
4966       { \l__zrefclever_type_first_label_type_tl }
4967       { \l__zrefclever_name_format_tl }
4968       { tl }
4969     }
4970     \l__zrefclever_type_name_tl
4971     {

```

```

4972         \tl_clear:N \l__zrefclever_type_name_tl
4973         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4974         \msg_warning:nnxx { zref-clever } { missing-name }
4975         { \l__zrefclever_name_format_tl }
4976         { \l__zrefclever_type_first_label_type_tl }
4977     }
4978 }
4979 }
4980 {
4981     \__zrefclever_opt_tl_get:cNF
4982     {
4983         \__zrefclever_opt_varname_type:een
4984         { \l__zrefclever_type_first_label_type_tl }
4985         { \l__zrefclever_name_format_tl }
4986         { tl }
4987     }
4988     \l__zrefclever_type_name_tl
4989     {
4990         \__zrefclever_opt_tl_get:cNF
4991         {
4992             \__zrefclever_opt_varname_type:een
4993             { \l__zrefclever_type_first_label_type_tl }
4994             { \l__zrefclever_name_format_fallback_tl }
4995             { tl }
4996         }
4997         \l__zrefclever_type_name_tl
4998         {
4999             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5000             {
5001                 \tl_put_left:Nn
5002                 \l__zrefclever_name_format_tl { - }
5003                 \tl_put_left:NV \l__zrefclever_name_format_tl
5004                 \l__zrefclever_ref_decl_case_tl
5005                 \tl_put_left:Nn
5006                 \l__zrefclever_name_format_fallback_tl { - }
5007                 \tl_put_left:NV
5008                 \l__zrefclever_name_format_fallback_tl
5009                 \l__zrefclever_ref_decl_case_tl
5010             }
5011             \__zrefclever_opt_tl_get:cNF
5012             {
5013                 \__zrefclever_opt_varname_lang_type:eeen
5014                 { \l__zrefclever_ref_language_tl }
5015                 { \l__zrefclever_type_first_label_type_tl }
5016                 { \l__zrefclever_name_format_tl }
5017                 { tl }
5018             }
5019             \l__zrefclever_type_name_tl
5020             {
5021                 \__zrefclever_opt_tl_get:cNF
5022                 {
5023                     \__zrefclever_opt_varname_lang_type:eeen
5024                     { \l__zrefclever_ref_language_tl }
5025                     { \l__zrefclever_type_first_label_type_tl }

```

```

5026         { \l__zrefclever_name_format_fallback_tl }
5027         { tl }
5028     }
5029     \l__zrefclever_type_name_tl
5030     {
5031         \tl_clear:N \l__zrefclever_type_name_tl
5032         \bool_set_true:N
5033         \l__zrefclever_type_name_missing_bool
5034         \msg_warning:nxxx { zref-clever }
5035         { missing-name }
5036         { \l__zrefclever_name_format_tl }
5037         { \l__zrefclever_type_first_label_type_tl }
5038     }
5039 }
5040 }
5041 }
5042 }
5043 }
5044 }
5045
5046 % Signal whether the type name is to be included in the hyperlink or not.
5047 \bool_lazy_any:nTF
5048 {
5049     { ! \l__zrefclever_hyperlink_bool }
5050     { \l__zrefclever_link_star_bool }
5051     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5052     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5053 }
5054 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5055 {
5056     \bool_lazy_any:nTF
5057     {
5058         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5059         {
5060             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5061             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5062         }
5063         {
5064             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5065             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5066             \l__zrefclever_typeset_last_bool &&
5067             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5068         }
5069     }
5070     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5071     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5072 }
5073 }

```

(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.
5074 \cs_new:Npn \_zrefclever_extract_url_unexp:n #1
5075 {
5076   \zref@ifpropundefined { urluse }
5077   { \_zrefclever_extract_unexp:nnn {#1} { url } { } }
5078   {
5079     \zref@ifrefcontainsprop {#1} { urluse }
5080     { \_zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5081     { \_zrefclever_extract_unexp:nnn {#1} { url } { } }
5082   }
5083 }
5084 \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 {\langle label\ a \rangle} {\langle label\ b \rangle}

5085 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
5086 {
5087   \exp_args:Nxx \tl_if_eq:nnT
5088   { \_zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5089   { \_zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5090   {
5091     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5092     {
5093       \exp_args:Nxx \tl_if_eq:nnT
5094       { \_zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5095       { \_zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5096       {
5097         \int_compare:nNnTF
5098         { \_zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5099         =
5100         { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5101         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5102         {
5103           \int_compare:nNnTF
5104           { \_zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5105           =
5106           { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5107           {
5108             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5109             \bool_set_true:N \l__zrefclever_next_is_same_bool
5110           }
5111         }
5112       }
5113     }

```



```

5114 {
5115   \exp_args:Nxx \tl_if_eq:nnT
5116   { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5117   { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5118   {
5119     \exp_args:Nxx \tl_if_eq:nnT
5120     { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5121     { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5122     {
5123       \int_compare:nNnTF
5124       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5125       =
5126       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5127       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5128       {
5129         \int_compare:nNnTF
5130         { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5131         =
5132         { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5133         {

```

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

```

5134   \exp_args:Nxx \tl_if_eq:nnT
5135   {
5136     \_zrefclever_extract_unexp:nvn {#1}
5137     { \l__zrefclever_ref_property_tl } { }
5138   }
5139   {
5140     \_zrefclever_extract_unexp:nvn {#2}
5141     { \l__zrefclever_ref_property_tl } { }
5142   }
5143   {
5144     \bool_set_true:N
5145     \l__zrefclever_next_maybe_range_bool
5146     \bool_set_true:N
5147     \l__zrefclever_next_is_same_bool
5148   }
5149 }
5150 }
5151 }
5152 }
5153 }
5154 }
5155 }

```

(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>}
5156 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5157 {
5158   % First attempt: general options.
5159   \__zrefclever_opt_tl_get:cNF
5160   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5161   #4
5162   {
5163     % If not found, try type specific options.
5164     \__zrefclever_opt_tl_get:cNF
5165     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5166     #4
5167     {
5168       % If not found, try type- and language-specific.
5169       \__zrefclever_opt_tl_get:cNF
5170       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5171       #4
5172       {
5173         % If not found, try language-specific default.
5174         \__zrefclever_opt_tl_get:cNF
5175         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5176         #4
5177         {
5178           % If not found, try fallback.
5179           \__zrefclever_opt_tl_get:cNF
5180           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5181           #4
5182           { \tl_clear:N #4 }
5183         }
5184       }
5185     }
5186   }
5187 }
5188 \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>}
5189 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5190 {
5191   % First attempt: general options.
5192   \__zrefclever_opt_seq_get:cNF
5193   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5194   #4
5195   {
5196     % If not found, try type specific options.
5197     \__zrefclever_opt_seq_get:cNF
5198     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5199     #4
5200     {
5201       % If not found, try type- and language-specific.
5202       \__zrefclever_opt_seq_get:cNF

```

```

5203         { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5204         #4
5205         {
5206             % If not found, try language-specific default.
5207             \_zrefclever_opt_seq_get:cNF
5208             { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5209             #4
5210             {
5211                 % If not found, try fallback.
5212                 \_zrefclever_opt_seq_get:cNF
5213                 { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5214                 #4
5215                 { \seq_clear:N #4 }
5216             }
5217         }
5218     }
5219 }
5220 }
5221 \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>}
5222 \cs_new_protected:Npn \_zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5223 {
5224     % First attempt: general options.
5225     \_zrefclever_opt_bool_get:cNF
5226     { \_zrefclever_opt_varname_general:nn {#1} { bool } }
5227     #5
5228     {
5229         % If not found, try type specific options.
5230         \_zrefclever_opt_bool_get:cNF
5231         { \_zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5232         #5
5233         {
5234             % If not found, try type- and language-specific.
5235             \_zrefclever_opt_bool_get:cNF
5236             { \_zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5237             #5
5238             {
5239                 % If not found, try language-specific default.
5240                 \_zrefclever_opt_bool_get:cNF
5241                 { \_zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5242                 #5
5243                 {
5244                     % If not found, try fallback.
5245                     \_zrefclever_opt_bool_get:cNF
5246                     { \_zrefclever_opt_varname_fallback:nn {#1} { bool } }
5247                     #5
5248                     { \use:c { bool_set_ #2 :N } #5 }
5249                 }
5250             }
5251         }

```

```

5252     }
5253   }
5254   \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nxxxN }

```

(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.

```

5255 \__zrefclever_compat_module:nn { appendix }
5256 {
5257   \AddToHook { cmd / appendix / before }
5258   {
5259     \__zrefclever_zcsetup:n
5260     {
5261       countertype =
5262       {
5263         chapter      = appendix ,
5264         section      = appendix ,
5265         subsection   = appendix ,
5266         subsubsection = appendix ,
5267         paragraph    = appendix ,
5268         subparagraph = appendix ,
5269       }
5270     }
5271   }
5272 }

```

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.

```

5273 \__zrefclever_compat_module:nn { appendices }
5274 {
5275   \__zrefclever_if_package_loaded:nT { appendix }
5276   {
5277     \newcounter { zc@appendix }
5278     \newcounter { zc@save@appendix }
5279     \setcounter { zc@appendix } { 0 }
5280     \setcounter { zc@save@appendix } { 0 }
5281     \cs_if_exist:cTF { chapter }
5282     {
5283       \__zrefclever_zcsetup:n
5284       { counterresetby = { chapter = zc@appendix } }
5285     }
5286     {
5287       \cs_if_exist:cT { section }
5288       {
5289         \__zrefclever_zcsetup:n
5290         { counterresetby = { section = zc@appendix } }
5291       }
5292     }
5293   }
5294   \AddToHook { env / appendices / begin }
5295   {
5296     \stepcounter { zc@save@appendix }
5297     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5298     \__zrefclever_zcsetup:n
5299     {
5300       countertype =
5301       {
5302         chapter      = appendix ,
5303         section      = appendix ,
5304         subsection   = appendix ,
5305         subsubsection = appendix ,
5306         paragraph    = appendix ,
5307         subparagraph = appendix ,
5308       }
5309     }
5310   }
5311   \AddToHook { env / appendices / end }
5312   { \setcounter { zc@appendix } { 0 } }
5313   \AddToHook { cmd / appendix / before }
5314   {
5315     \stepcounter { zc@save@appendix }
5316     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5317   }

```

```

5317 \AddToHook { env / subappendices / begin }
5318 {
5319     \__zrefclever_zcsetup:n
5320     {
5321         countertype =
5322         {
5323             section      = appendix ,
5324             subsection   = appendix ,
5325             subsubsection = appendix ,
5326             paragraph    = appendix ,
5327             subparagraph = appendix ,
5328         } ,
5329     }
5330 }
5331 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5332 }
5333 }

```

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.

```

5334 \__zrefclever_compat_module:n { memoir }
5335 {
5336   \__zrefclever_if_class_loaded:nT { memoir }
5337   {

```

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`.

```

5338   \__zrefclever_zcsetup:n
5339   {
5340     countertype =
5341     {
5342       subfigure = figure ,
5343       subtable  = table ,
5344       poemline  = line ,
5345     } ,
5346     counterresetby =
5347     {
5348       subfigure = figure ,
5349       subtable  = table ,
5350     } ,
5351   }

```

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

```

5352   \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5353   {
5354     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5355     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5356     {
5357       \__zrefclever_memoir_orig_label:n {##1}
5358       \zlabel{##1}
5359     }
5360     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n

```

```

5361     }
5362     \AddToHook { env / sidecaption / begin }
5363       { \_zrefclever_memoir_both_labels: }
5364     \AddToHook { env / sidecontcaption / begin }
5365       { \_zrefclever_memoir_both_labels: }
5366     \AddToHook{ cmd / bitwonumcaption / before }
5367       { \group_begin: \_zrefclever_memoir_both_labels: }
5368     \AddToHook{ cmd / bitwonumcaption / after }
5369       { \group_end: }
5370     \AddToHook{ cmd / bionenumcaption / before }
5371       { \group_begin: \_zrefclever_memoir_both_labels: }
5372     \AddToHook{ cmd / bionenumcaption / after }
5373       { \group_end: }
5374     \AddToHook{ cmd / bicaption / before }
5375       { \group_begin: \_zrefclever_memoir_both_labels: }
5376     \AddToHook{ cmd / bicaption / after }
5377       { \group_end: }

```

Support for subcaption reference.

```

5378     \zref@newprop { subcaption }
5379       { \cs_if_exist_use:c { @@thesub \@capttype } }
5380     \AddToHook{ cmd / @memsubcaption / before }
5381       { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5382     \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5383     \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5384     \AddToHook { env / minipage / begin }
5385       { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5386     \AddToHook { cmd / @makefntext / before }
5387       {
5388         \__zrefclever_zcsetup:x
5389           { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5390       }
5391     \AddToHook { cmd / @makesidefntext / before }
5392       { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5393     \__zrefclever_zcsetup:n
5394       {
5395         countertype =
5396           {
5397             sidefootnote = footnote ,
5398             pagenote = endnote ,
5399           } ,
5400       }
5401     \AddToHook { file / \jobname.ent / before }
5402       { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5403     \msg_info:nnn { zref-clever } { compat-class } { memoir }
5404   }
5405 }

```

9.4 KOMA

Support for KOMA-Script document classes.

```

5406 \__zrefclever_compat_module:nn { KOMA }

```



```

5407 {
5408   \cs_if_exist:NT \KOMAClassName
5409   {

```

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.

```

5410   \AddToHook { env / captionbeside / after }
5411   {
5412     \tl_if_exist:NT \@capttype
5413     { \tl_set_eq:NN \@currentcounter \@capttype }
5414   }
5415   \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
5416   \AddToHook { env / captionofbeside / end }
5417   { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
5418   \AddToHook { env / captionofbeside / after }
5419   {
5420     \tl_if_eq:NnF \@currentenv { document }
5421     {
5422       \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
5423       {
5424         \tl_set_eq:NN
5425         \@currentcounter \g__zrefclever_koma_capttype_tl
5426       }
5427     }
5428     \tl_gclear:N \g__zrefclever_koma_capttype_tl
5429   }
5430   \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
5431 }
5432 }

```

9.5 amsmath

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

```

5433 \__zrefclever_compat_module:nn { amsmath }
5434 {
5435   \__zrefclever_if_package_loaded:nT { amsmath }
5436   {

```

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...).

```

5437     \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5438     {
5439         \__zrefclever_orig_ltxlabel:n {#1}
5440         \zref@wrapper@babel \zref@label {#1}
5441     }

```

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>.

```

5442     \__zrefclever_if_package_loaded:nTF { hyperref }
5443     {
5444         \AddToHook { package / nameref / after }
5445         {
5446             \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5447             \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5448         }
5449     }
5450     {
5451         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5452         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5453     }

```

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).

```

5454 \bool_new:N \l__zrefclever_amsmath_subequations_bool
5455 \AddToHook { env / subequations / begin }
5456 {
5457   \__zrefclever_zcsetup:x
5458   {
5459     counterresetby =
5460     {
5461       parentequation =
5462       \__zrefclever_counter_reset_by:n { equation } ,
5463       equation = parentequation ,
5464     } ,
5465     currentcounter = parentequation ,
5466     countertype = { parentequation = equation } ,
5467   }
5468   \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5469 }

```

`amsmath` does use `\refstepcounter` for the equation counter throughout and does set `\@currentcounter` for `\tags`. But we still have to manually reset `currentcounter` to default because, since we had to manually set `currentcounter` to `parentequation` in `subequations`, we also have to manually set it to `equation` in environments which may be used within it. The `xxalignat` environment is not included, because it is “starred” by default (i.e. unnumbered), and does not display or accepts labels or tags anyway. The `-ed` (`gathered`, `aligned`, and `alignedat`) and `cases` environments “must appear within an enclosing math environment”. Same logic applies to other environments defined or redefined by the package, like `array`, `matrix` and variations. Finally, `split` too can only be used as part of another environment. We also arrange, at this point, for the provision of the `subeq` property, for the convenience of referring to them directly or to build terse ranges with the `endrange` option.

```

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

```

```

5492             \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5493             { \zref@localaddprop \ZREF@mainlist { subeq } }
5494         }
5495     }

```

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

```

5496     \zcRefTypeSetup { equation }
5497     { reffont = \upshape }
5498     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5499 }
5500 }

```

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.

```

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

```

```

5530         }
5531         \@esphack
5532     }
5533     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5534 }
5535 }
5536 }

```

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”.

```

5537 \__zrefclever_compat_module:nn { breqn }
5538 {
5539     \__zrefclever_if_package_loaded:nT { breqn }
5540 }

```

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>).

```

5541 \bool_new:N \l__zrefclever_breqn_dgroup_bool
5542 \AddToHook { env / dgroup / begin }
5543 {
5544     \__zrefclever_zcsetup:x
5545     {
5546         counterresetby =
5547         {
5548             parentequation =
5549             \__zrefclever_counter_reset_by:n { equation } ,
5550             equation = parentequation ,
5551         } ,
5552         currentcounter = parentequation ,
5553         countertype = { parentequation = equation } ,
5554     }
5555     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5556 }
5557 \zref@ifpropundefined { subeq }
5558 { \zref@newprop { subeq } { \alph { equation } } }
5559 { }
5560 \clist_map_inline:nn
5561 {
5562     dmath ,
5563     dseries ,
5564     darray ,
5565 }
5566 {

```

```

5567         \AddToHook { env / #1 / begin }
5568         {
5569             \__zrefclever_zcsetup:n { currentcounter = equation }
5570             \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5571             { \zref@localaddprop \ZREF@mainlist { subeq } }
5572         }
5573     }
5574     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5575 }
5576 }

```

9.8 listings

```

5577 \__zrefclever_compat_module:nn { listings }
5578 {
5579     \__zrefclever_if_package_loaded:NT { listings }
5580     {
5581         \__zrefclever_zcsetup:n
5582         {
5583             countertype =
5584             {
5585                 lstlisting = listing ,
5586                 lstnumber = line ,
5587             } ,
5588             counterresetby = { lstnumber = lstlisting } ,
5589         }
5590     }
5591 }

```

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.

```

5590 \lst@AddToHook { PreInit }
5591 { \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.

```

5592 \lst@AddToHook { Init }
5593 { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5594 \msg_info:nnn { zref-clever } { compat-package } { listings }
5595 }
5596 }

```

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

```

5597 \__zrefclever_compat_module:nn { enumitem }
5598 {
5599   \__zrefclever_if_package_loaded:nT { enumitem }
5600   {
5601     \int_set:Nn \l_tmpa_int { 5 }
5602     \bool_while_do:nn
5603     {
5604       \cs_if_exist_p:c
5605       { c@ enum \int_to_roman:n { \l_tmpa_int } }
5606     }
5607     {
5608       \__zrefclever_zcsetup:x
5609       {
5610         counterresetby =
5611         {
5612           enum \int_to_roman:n { \l_tmpa_int } =
5613           enum \int_to_roman:n { \l_tmpa_int - 1 }
5614         } ,
5615         countertype =
5616         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5617       }
5618       \int_incr:N \l_tmpa_int
5619     }
5620     \int_compare:nNnT { \l_tmpa_int } > { 5 }
5621     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5622   }
5623 }

```

9.10 subcaption

```

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

```

```

5636         {
5637             subfigure = figure ,
5638             subtable = table ,
5639         } ,
5640     }

```

Support for subref reference.

```

5641     \zref@newprop { subref }
5642     { \cs_if_exist_use:c { thesub \@captype } }
5643     \tl_put_right:Nn \caption@subtypetool
5644     { \zref@localaddprop \ZREF@mainlist { subref } }
5645 }
5646 }

```

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.

```

5647 \__zrefclever_compat_module:nn { subfig }
5648 {
5649     \__zrefclever_if_package_loaded:nT { subfig }
5650     {
5651         \__zrefclever_zcsetup:n
5652         {
5653             countertype =
5654             {
5655                 subfigure = figure ,
5656                 subtable = table ,
5657             } ,
5658             counterresetby =
5659             {
5660                 subfigure = figure ,
5661                 subtable = table ,
5662             } ,
5663         }
5664     }
5665 }
5666 </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.

```

5667 <*package>
5668 \zcDeclareLanguage { english }
5669 \zcDeclareLanguageAlias { american } { english }

```



```

5670 \zcDeclareLanguageAlias { australian } { english }
5671 \zcDeclareLanguageAlias { british    } { english }
5672 \zcDeclareLanguageAlias { canadian   } { english }
5673 \zcDeclareLanguageAlias { newzealand } { english }
5674 \zcDeclareLanguageAlias { UKenglish  } { english }
5675 \zcDeclareLanguageAlias { USenglish  } { english }
5676 </package>
5677 <*lang-english>
5678 namesep   = {\nobreakspace} ,
5679 pairsep   = {\~and\nobreakspace} ,
5680 listsep    = {\~,~} ,
5681 lastsep    = {\~and\nobreakspace} ,
5682 tpairsep   = {\~and\nobreakspace} ,
5683 tlistsep   = {\~,~} ,
5684 tlastsep   = {\~,~and\nobreakspace} ,
5685 notesep    = {\~} ,
5686 rangesep   = {\~to\nobreakspace} ,
5687
5688 type = book ,
5689   Name-sg = Book ,
5690   name-sg = book ,
5691   Name-pl = Books ,
5692   name-pl = books ,
5693
5694 type = part ,
5695   Name-sg = Part ,
5696   name-sg = part ,
5697   Name-pl = Parts ,
5698   name-pl = parts ,
5699
5700 type = chapter ,
5701   Name-sg = Chapter ,
5702   name-sg = chapter ,
5703   Name-pl = Chapters ,
5704   name-pl = chapters ,
5705
5706 type = section ,
5707   Name-sg = Section ,
5708   name-sg = section ,
5709   Name-pl = Sections ,
5710   name-pl = sections ,
5711
5712 type = paragraph ,
5713   Name-sg = Paragraph ,
5714   name-sg = paragraph ,
5715   Name-pl = Paragraphs ,
5716   name-pl = paragraphs ,
5717   Name-sg-ab = Par. ,
5718   name-sg-ab = par. ,
5719   Name-pl-ab = Par. ,
5720   name-pl-ab = par. ,
5721
5722 type = appendix ,

```

```

5723 Name-sg = Appendix ,
5724 name-sg = appendix ,
5725 Name-pl = Appendices ,
5726 name-pl = appendices ,
5727
5728 type = page ,
5729 Name-sg = Page ,
5730 name-sg = page ,
5731 Name-pl = Pages ,
5732 name-pl = pages ,
5733 rangesep = {\textendash} ,
5734 rangetopair = false ,
5735
5736 type = line ,
5737 Name-sg = Line ,
5738 name-sg = line ,
5739 Name-pl = Lines ,
5740 name-pl = lines ,
5741
5742 type = figure ,
5743 Name-sg = Figure ,
5744 name-sg = figure ,
5745 Name-pl = Figures ,
5746 name-pl = figures ,
5747 Name-sg-ab = Fig. ,
5748 name-sg-ab = fig. ,
5749 Name-pl-ab = Figs. ,
5750 name-pl-ab = figs. ,
5751
5752 type = table ,
5753 Name-sg = Table ,
5754 name-sg = table ,
5755 Name-pl = Tables ,
5756 name-pl = tables ,
5757
5758 type = item ,
5759 Name-sg = Item ,
5760 name-sg = item ,
5761 Name-pl = Items ,
5762 name-pl = items ,
5763
5764 type = footnote ,
5765 Name-sg = Footnote ,
5766 name-sg = footnote ,
5767 Name-pl = Footnotes ,
5768 name-pl = footnotes ,
5769
5770 type = endnote ,
5771 Name-sg = Note ,
5772 name-sg = note ,
5773 Name-pl = Notes ,
5774 name-pl = notes ,
5775
5776 type = note ,

```

```

5777 Name-sg = Note ,
5778 name-sg = note ,
5779 Name-pl = Notes ,
5780 name-pl = notes ,
5781
5782 type = equation ,
5783 Name-sg = Equation ,
5784 name-sg = equation ,
5785 Name-pl = Equations ,
5786 name-pl = equations ,
5787 Name-sg-ab = Eq. ,
5788 name-sg-ab = eq. ,
5789 Name-pl-ab = Eqs. ,
5790 name-pl-ab = eqs. ,
5791 refbounds-first-sg = {,(,),} ,
5792 refbounds = {(,,)} ,
5793
5794 type = theorem ,
5795 Name-sg = Theorem ,
5796 name-sg = theorem ,
5797 Name-pl = Theorems ,
5798 name-pl = theorems ,
5799
5800 type = lemma ,
5801 Name-sg = Lemma ,
5802 name-sg = lemma ,
5803 Name-pl = Lemmas ,
5804 name-pl = lemmas ,
5805
5806 type = corollary ,
5807 Name-sg = Corollary ,
5808 name-sg = corollary ,
5809 Name-pl = Corollaries ,
5810 name-pl = corollaries ,
5811
5812 type = proposition ,
5813 Name-sg = Proposition ,
5814 name-sg = proposition ,
5815 Name-pl = Propositions ,
5816 name-pl = propositions ,
5817
5818 type = definition ,
5819 Name-sg = Definition ,
5820 name-sg = definition ,
5821 Name-pl = Definitions ,
5822 name-pl = definitions ,
5823
5824 type = proof ,
5825 Name-sg = Proof ,
5826 name-sg = proof ,
5827 Name-pl = Proofs ,
5828 name-pl = proofs ,
5829
5830 type = result ,

```

```

5831   Name-sg = Result ,
5832   name-sg = result ,
5833   Name-pl = Results ,
5834   name-pl = results ,
5835
5836   type = remark ,
5837   Name-sg = Remark ,
5838   name-sg = remark ,
5839   Name-pl = Remarks ,
5840   name-pl = remarks ,
5841
5842   type = example ,
5843   Name-sg = Example ,
5844   name-sg = example ,
5845   Name-pl = Examples ,
5846   name-pl = examples ,
5847
5848   type = algorithm ,
5849   Name-sg = Algorithm ,
5850   name-sg = algorithm ,
5851   Name-pl = Algorithms ,
5852   name-pl = algorithms ,
5853
5854   type = listing ,
5855   Name-sg = Listing ,
5856   name-sg = listing ,
5857   Name-pl = Listings ,
5858   name-pl = listings ,
5859
5860   type = exercise ,
5861   Name-sg = Exercise ,
5862   name-sg = exercise ,
5863   Name-pl = Exercises ,
5864   name-pl = exercises ,
5865
5866   type = solution ,
5867   Name-sg = Solution ,
5868   name-sg = solution ,
5869   Name-pl = Solutions ,
5870   name-pl = solutions ,
5871 </lang-english>

```

10.2 German

German language file has been initially provided by the author.

```

5872 <*package>
5873 \zcDeclareLanguage
5874   [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5875   { german }
5876 \zcDeclareLanguageAlias { austrian      } { german }
5877 \zcDeclareLanguageAlias { germanb       } { german }
5878 \zcDeclareLanguageAlias { ngerman       } { german }
5879 \zcDeclareLanguageAlias { naustrian      } { german }

```

```

5880 \zcDeclareLanguageAlias { nswissgerman } { german }
5881 \zcDeclareLanguageAlias { swissgerman } { german }
5882 \end{package}
5883 \begin{lang-german}

5884 namesep = {\nobreakspace} ,
5885 pairsep = {\simund\nobreakspace} ,
5886 listsep = {,~} ,
5887 lastsep = {\simund\nobreakspace} ,
5888 tpairsep = {\simund\nobreakspace} ,
5889 tlistsep = {,~} ,
5890 tlastsep = {\simund\nobreakspace} ,
5891 notesep = {\sim} ,
5892 rangesep = {\simbis\nobreakspace} ,
5893
5894 type = book ,
5895 gender = n ,
5896 case = N ,
5897     Name-sg = Buch ,
5898     Name-pl = Bücher ,
5899 case = A ,
5900     Name-sg = Buch ,
5901     Name-pl = Bücher ,
5902 case = D ,
5903     Name-sg = Buch ,
5904     Name-pl = Büchern ,
5905 case = G ,
5906     Name-sg = Buches ,
5907     Name-pl = Bücher ,
5908
5909 type = part ,
5910 gender = m ,
5911 case = N ,
5912     Name-sg = Teil ,
5913     Name-pl = Teile ,
5914 case = A ,
5915     Name-sg = Teil ,
5916     Name-pl = Teile ,
5917 case = D ,
5918     Name-sg = Teil ,
5919     Name-pl = Teilen ,
5920 case = G ,
5921     Name-sg = Teiles ,
5922     Name-pl = Teile ,
5923
5924 type = chapter ,
5925 gender = n ,
5926 case = N ,
5927     Name-sg = Kapitel ,
5928     Name-pl = Kapitel ,
5929 case = A ,
5930     Name-sg = Kapitel ,
5931     Name-pl = Kapitel ,
5932 case = D ,

```

```

5933     Name-sg = Kapitel ,
5934     Name-pl = Kapiteln ,
5935     case = G ,
5936     Name-sg = Kapitels ,
5937     Name-pl = Kapitel ,
5938
5939 type = section ,
5940     gender = m ,
5941     case = N ,
5942     Name-sg = Abschnitt ,
5943     Name-pl = Abschnitte ,
5944     case = A ,
5945     Name-sg = Abschnitt ,
5946     Name-pl = Abschnitte ,
5947     case = D ,
5948     Name-sg = Abschnitt ,
5949     Name-pl = Abschnitten ,
5950     case = G ,
5951     Name-sg = Abschnitts ,
5952     Name-pl = Abschnitte ,
5953
5954 type = paragraph ,
5955     gender = m ,
5956     case = N ,
5957     Name-sg = Absatz ,
5958     Name-pl = Absätze ,
5959     case = A ,
5960     Name-sg = Absatz ,
5961     Name-pl = Absätze ,
5962     case = D ,
5963     Name-sg = Absatz ,
5964     Name-pl = Absätzen ,
5965     case = G ,
5966     Name-sg = Absatzes ,
5967     Name-pl = Absätze ,
5968
5969 type = appendix ,
5970     gender = m ,
5971     case = N ,
5972     Name-sg = Anhang ,
5973     Name-pl = Anhänge ,
5974     case = A ,
5975     Name-sg = Anhang ,
5976     Name-pl = Anhänge ,
5977     case = D ,
5978     Name-sg = Anhang ,
5979     Name-pl = Anhängen ,
5980     case = G ,
5981     Name-sg = Anhangs ,
5982     Name-pl = Anhänge ,
5983
5984 type = page ,
5985     gender = f ,
5986     case = N ,

```

```

5987     Name-sg = Seite ,
5988     Name-pl = Seiten ,
5989     case = A ,
5990     Name-sg = Seite ,
5991     Name-pl = Seiten ,
5992     case = D ,
5993     Name-sg = Seite ,
5994     Name-pl = Seiten ,
5995     case = G ,
5996     Name-sg = Seite ,
5997     Name-pl = Seiten ,
5998     rangesep = {\textendash} ,
5999     rangetopair = false ,
6000
6001 type = line ,
6002     gender = f ,
6003     case = N ,
6004     Name-sg = Zeile ,
6005     Name-pl = Zeilen ,
6006     case = A ,
6007     Name-sg = Zeile ,
6008     Name-pl = Zeilen ,
6009     case = D ,
6010     Name-sg = Zeile ,
6011     Name-pl = Zeilen ,
6012     case = G ,
6013     Name-sg = Zeile ,
6014     Name-pl = Zeilen ,
6015
6016 type = figure ,
6017     gender = f ,
6018     case = N ,
6019     Name-sg = Abbildung ,
6020     Name-pl = Abbildungen ,
6021     Name-sg-ab = Abb. ,
6022     Name-pl-ab = Abb. ,
6023     case = A ,
6024     Name-sg = Abbildung ,
6025     Name-pl = Abbildungen ,
6026     Name-sg-ab = Abb. ,
6027     Name-pl-ab = Abb. ,
6028     case = D ,
6029     Name-sg = Abbildung ,
6030     Name-pl = Abbildungen ,
6031     Name-sg-ab = Abb. ,
6032     Name-pl-ab = Abb. ,
6033     case = G ,
6034     Name-sg = Abbildung ,
6035     Name-pl = Abbildungen ,
6036     Name-sg-ab = Abb. ,
6037     Name-pl-ab = Abb. ,
6038
6039 type = table ,
6040     gender = f ,

```

```

6041 case = N ,
6042     Name-sg = Tabelle ,
6043     Name-pl = Tabellen ,
6044 case = A ,
6045     Name-sg = Tabelle ,
6046     Name-pl = Tabellen ,
6047 case = D ,
6048     Name-sg = Tabelle ,
6049     Name-pl = Tabellen ,
6050 case = G ,
6051     Name-sg = Tabelle ,
6052     Name-pl = Tabellen ,
6053
6054 type = item ,
6055     gender = m ,
6056     case = N ,
6057         Name-sg = Punkt ,
6058         Name-pl = Punkte ,
6059     case = A ,
6060         Name-sg = Punkt ,
6061         Name-pl = Punkte ,
6062     case = D ,
6063         Name-sg = Punkt ,
6064         Name-pl = Punkten ,
6065     case = G ,
6066         Name-sg = Punktes ,
6067         Name-pl = Punkte ,
6068
6069 type = footnote ,
6070     gender = f ,
6071     case = N ,
6072         Name-sg = Fußnote ,
6073         Name-pl = Fußnoten ,
6074     case = A ,
6075         Name-sg = Fußnote ,
6076         Name-pl = Fußnoten ,
6077     case = D ,
6078         Name-sg = Fußnote ,
6079         Name-pl = Fußnoten ,
6080     case = G ,
6081         Name-sg = Fußnote ,
6082         Name-pl = Fußnoten ,
6083
6084 type = endnote ,
6085     gender = f ,
6086     case = N ,
6087         Name-sg = Endnote ,
6088         Name-pl = Endnoten ,
6089     case = A ,
6090         Name-sg = Endnote ,
6091         Name-pl = Endnoten ,
6092     case = D ,
6093         Name-sg = Endnote ,
6094         Name-pl = Endnoten ,

```



```

6095     case = G ,
6096         Name-sg = Endnote ,
6097         Name-pl = Endnoten ,
6098
6099 type = note ,
6100     gender = f ,
6101     case = N ,
6102         Name-sg = Anmerkung ,
6103         Name-pl = Anmerkungen ,
6104     case = A ,
6105         Name-sg = Anmerkung ,
6106         Name-pl = Anmerkungen ,
6107     case = D ,
6108         Name-sg = Anmerkung ,
6109         Name-pl = Anmerkungen ,
6110     case = G ,
6111         Name-sg = Anmerkung ,
6112         Name-pl = Anmerkungen ,
6113
6114 type = equation ,
6115     gender = f ,
6116     case = N ,
6117         Name-sg = Gleichung ,
6118         Name-pl = Gleichungen ,
6119     case = A ,
6120         Name-sg = Gleichung ,
6121         Name-pl = Gleichungen ,
6122     case = D ,
6123         Name-sg = Gleichung ,
6124         Name-pl = Gleichungen ,
6125     case = G ,
6126         Name-sg = Gleichung ,
6127         Name-pl = Gleichungen ,
6128     refbounds-first-sg = {,(,),} ,
6129     refbounds = {(,,)} ,
6130
6131 type = theorem ,
6132     gender = n ,
6133     case = N ,
6134         Name-sg = Theorem ,
6135         Name-pl = Theoreme ,
6136     case = A ,
6137         Name-sg = Theorem ,
6138         Name-pl = Theoreme ,
6139     case = D ,
6140         Name-sg = Theorem ,
6141         Name-pl = Theoremen ,
6142     case = G ,
6143         Name-sg = Theorems ,
6144         Name-pl = Theoreme ,
6145
6146 type = lemma ,
6147     gender = n ,
6148     case = N ,

```

```

6149     Name-sg = Lemma ,
6150     Name-pl = Lemmata ,
6151     case = A ,
6152     Name-sg = Lemma ,
6153     Name-pl = Lemmata ,
6154     case = D ,
6155     Name-sg = Lemma ,
6156     Name-pl = Lemmata ,
6157     case = G ,
6158     Name-sg = Lemmas ,
6159     Name-pl = Lemmata ,
6160
6161 type = corollary ,
6162     gender = n ,
6163     case = N ,
6164     Name-sg = Korollar ,
6165     Name-pl = Korollare ,
6166     case = A ,
6167     Name-sg = Korollar ,
6168     Name-pl = Korollare ,
6169     case = D ,
6170     Name-sg = Korollar ,
6171     Name-pl = Korollaren ,
6172     case = G ,
6173     Name-sg = Korollars ,
6174     Name-pl = Korollare ,
6175
6176 type = proposition ,
6177     gender = m ,
6178     case = N ,
6179     Name-sg = Satz ,
6180     Name-pl = Sätze ,
6181     case = A ,
6182     Name-sg = Satz ,
6183     Name-pl = Sätze ,
6184     case = D ,
6185     Name-sg = Satz ,
6186     Name-pl = Sätzen ,
6187     case = G ,
6188     Name-sg = Satzes ,
6189     Name-pl = Sätze ,
6190
6191 type = definition ,
6192     gender = f ,
6193     case = N ,
6194     Name-sg = Definition ,
6195     Name-pl = Definitionen ,
6196     case = A ,
6197     Name-sg = Definition ,
6198     Name-pl = Definitionen ,
6199     case = D ,
6200     Name-sg = Definition ,
6201     Name-pl = Definitionen ,
6202     case = G ,

```

```

6203     Name-sg = Definition ,
6204     Name-pl = Definitionen ,
6205
6206 type = proof ,
6207     gender = m ,
6208     case = N ,
6209     Name-sg = Beweis ,
6210     Name-pl = Beweise ,
6211     case = A ,
6212     Name-sg = Beweis ,
6213     Name-pl = Beweise ,
6214     case = D ,
6215     Name-sg = Beweis ,
6216     Name-pl = Beweisen ,
6217     case = G ,
6218     Name-sg = Beweises ,
6219     Name-pl = Beweise ,
6220
6221 type = result ,
6222     gender = n ,
6223     case = N ,
6224     Name-sg = Ergebnis ,
6225     Name-pl = Ergebnisse ,
6226     case = A ,
6227     Name-sg = Ergebnis ,
6228     Name-pl = Ergebnisse ,
6229     case = D ,
6230     Name-sg = Ergebnis ,
6231     Name-pl = Ergebnissen ,
6232     case = G ,
6233     Name-sg = Ergebnisses ,
6234     Name-pl = Ergebnisse ,
6235
6236 type = remark ,
6237     gender = f ,
6238     case = N ,
6239     Name-sg = Bemerkung ,
6240     Name-pl = Bemerkungen ,
6241     case = A ,
6242     Name-sg = Bemerkung ,
6243     Name-pl = Bemerkungen ,
6244     case = D ,
6245     Name-sg = Bemerkung ,
6246     Name-pl = Bemerkungen ,
6247     case = G ,
6248     Name-sg = Bemerkung ,
6249     Name-pl = Bemerkungen ,
6250
6251 type = example ,
6252     gender = n ,
6253     case = N ,
6254     Name-sg = Beispiel ,
6255     Name-pl = Beispiele ,
6256     case = A ,

```

```

6257     Name-sg = Beispiel ,
6258     Name-pl = Beispiele ,
6259     case = D ,
6260     Name-sg = Beispiel ,
6261     Name-pl = Beispielen ,
6262     case = G ,
6263     Name-sg = Beispiels ,
6264     Name-pl = Beispiele ,
6265
6266 type = algorithm ,
6267     gender = m ,
6268     case = N ,
6269     Name-sg = Algorithmus ,
6270     Name-pl = Algorithmen ,
6271     case = A ,
6272     Name-sg = Algorithmus ,
6273     Name-pl = Algorithmen ,
6274     case = D ,
6275     Name-sg = Algorithmus ,
6276     Name-pl = Algorithmen ,
6277     case = G ,
6278     Name-sg = Algorithmus ,
6279     Name-pl = Algorithmen ,
6280
6281 type = listing ,
6282     gender = n ,
6283     case = N ,
6284     Name-sg = Listing ,
6285     Name-pl = Listings ,
6286     case = A ,
6287     Name-sg = Listing ,
6288     Name-pl = Listings ,
6289     case = D ,
6290     Name-sg = Listing ,
6291     Name-pl = Listings ,
6292     case = G ,
6293     Name-sg = Listings ,
6294     Name-pl = Listings ,
6295
6296 type = exercise ,
6297     gender = f ,
6298     case = N ,
6299     Name-sg = Übungsaufgabe ,
6300     Name-pl = Übungsaufgaben ,
6301     case = A ,
6302     Name-sg = Übungsaufgabe ,
6303     Name-pl = Übungsaufgaben ,
6304     case = D ,
6305     Name-sg = Übungsaufgabe ,
6306     Name-pl = Übungsaufgaben ,
6307     case = G ,
6308     Name-sg = Übungsaufgabe ,
6309     Name-pl = Übungsaufgaben ,
6310

```

```

6311 type = solution ,
6312   gender = f ,
6313   case = N ,
6314     Name-sg = Lösung ,
6315     Name-pl = Lösungen ,
6316   case = A ,
6317     Name-sg = Lösung ,
6318     Name-pl = Lösungen ,
6319   case = D ,
6320     Name-sg = Lösung ,
6321     Name-pl = Lösungen ,
6322   case = G ,
6323     Name-sg = Lösung ,
6324     Name-pl = Lösungen ,
6325 </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.

```

6326 <*package>
6327 \zcDeclareLanguage [ gender = { f , m } ] { french }
6328 \zcDeclareLanguageAlias { acadian } { french }
6329 \zcDeclareLanguageAlias { canadien } { french }
6330 \zcDeclareLanguageAlias { francais } { french }
6331 \zcDeclareLanguageAlias { frenchb } { french }
6332 </package>
6333 <*lang-french>
6334 namesep = {\nobreakspace} ,
6335 pairsep = {\~et\nobreakspace} ,
6336 listsep = { , ~ } ,
6337 lastsep = {\~et\nobreakspace} ,
6338 tpairsep = {\~et\nobreakspace} ,
6339 tlistsep = { , ~ } ,
6340 tlastsep = {\~et\nobreakspace} ,
6341 notesep = { ~ } ,
6342 rangesep = {\~à\nobreakspace} ,
6343
6344 type = book ,
6345   gender = m ,
6346   Name-sg = Livre ,
6347   name-sg = livre ,
6348   Name-pl = Livres ,
6349   name-pl = livres ,
6350
6351 type = part ,
6352   gender = f ,
6353   Name-sg = Partie ,
6354   name-sg = partie ,

```

```

6355     Name-pl = Parties ,
6356     name-pl = parties ,
6357
6358     type = chapter ,
6359     gender = m ,
6360     Name-sg = Chapitre ,
6361     name-sg = chapitre ,
6362     Name-pl = Chapitres ,
6363     name-pl = chapitres ,
6364
6365     type = section ,
6366     gender = f ,
6367     Name-sg = Section ,
6368     name-sg = section ,
6369     Name-pl = Sections ,
6370     name-pl = sections ,
6371
6372     type = paragraph ,
6373     gender = m ,
6374     Name-sg = Paragraphe ,
6375     name-sg = paragraphe ,
6376     Name-pl = Paragraphes ,
6377     name-pl = paragraphes ,
6378
6379     type = appendix ,
6380     gender = f ,
6381     Name-sg = Annexe ,
6382     name-sg = annexe ,
6383     Name-pl = Annexes ,
6384     name-pl = annexes ,
6385
6386     type = page ,
6387     gender = f ,
6388     Name-sg = Page ,
6389     name-sg = page ,
6390     Name-pl = Pages ,
6391     name-pl = pages ,
6392     rangesep = {-} ,
6393     rangetopair = false ,
6394
6395     type = line ,
6396     gender = f ,
6397     Name-sg = Ligne ,
6398     name-sg = ligne ,
6399     Name-pl = Lignes ,
6400     name-pl = lignes ,
6401
6402     type = figure ,
6403     gender = f ,
6404     Name-sg = Figure ,
6405     name-sg = figure ,
6406     Name-pl = Figures ,
6407     name-pl = figures ,
6408

```

```

6409 type = table ,
6410     gender = f ,
6411     Name-sg = Table ,
6412     name-sg = table ,
6413     Name-pl = Tables ,
6414     name-pl = tables ,
6415
6416 type = item ,
6417     gender = m ,
6418     Name-sg = Point ,
6419     name-sg = point ,
6420     Name-pl = Points ,
6421     name-pl = points ,
6422
6423 type = footnote ,
6424     gender = f ,
6425     Name-sg = Note ,
6426     name-sg = note ,
6427     Name-pl = Notes ,
6428     name-pl = notes ,
6429
6430 type = endnote ,
6431     gender = f ,
6432     Name-sg = Note ,
6433     name-sg = note ,
6434     Name-pl = Notes ,
6435     name-pl = notes ,
6436
6437 type = note ,
6438     gender = f ,
6439     Name-sg = Note ,
6440     name-sg = note ,
6441     Name-pl = Notes ,
6442     name-pl = notes ,
6443
6444 type = equation ,
6445     gender = f ,
6446     Name-sg = Équation ,
6447     name-sg = équation ,
6448     Name-pl = Équations ,
6449     name-pl = équations ,
6450     refbounds-first-sg = {,(,)}, ,
6451     refbounds = {(,,)} ,
6452
6453 type = theorem ,
6454     gender = m ,
6455     Name-sg = Théorème ,
6456     name-sg = théorème ,
6457     Name-pl = Théorèmes ,
6458     name-pl = théorèmes ,
6459
6460 type = lemma ,
6461     gender = m ,
6462     Name-sg = Lemme ,

```

```

6463     name-sg = lemme ,
6464     Name-pl = Lemmes ,
6465     name-pl = lemmes ,
6466
6467 type = corollary ,
6468     gender = m ,
6469     Name-sg = Corollaire ,
6470     name-sg = corollaire ,
6471     Name-pl = Corollaires ,
6472     name-pl = corollaires ,
6473
6474 type = proposition ,
6475     gender = f ,
6476     Name-sg = Proposition ,
6477     name-sg = proposition ,
6478     Name-pl = Propositions ,
6479     name-pl = propositions ,
6480
6481 type = definition ,
6482     gender = f ,
6483     Name-sg = Définition ,
6484     name-sg = définition ,
6485     Name-pl = Définitions ,
6486     name-pl = définitions ,
6487
6488 type = proof ,
6489     gender = f ,
6490     Name-sg = Démonstration ,
6491     name-sg = démonstration ,
6492     Name-pl = Démonstrations ,
6493     name-pl = démonstrations ,
6494
6495 type = result ,
6496     gender = m ,
6497     Name-sg = Résultat ,
6498     name-sg = résultat ,
6499     Name-pl = Résultats ,
6500     name-pl = résultats ,
6501
6502 type = remark ,
6503     gender = f ,
6504     Name-sg = Remarque ,
6505     name-sg = remarque ,
6506     Name-pl = Remarques ,
6507     name-pl = remarques ,
6508
6509 type = example ,
6510     gender = m ,
6511     Name-sg = Exemple ,
6512     name-sg = exemple ,
6513     Name-pl = Exemples ,
6514     name-pl = exemples ,
6515
6516 type = algorithm ,

```



```

6517   gender = m ,
6518   Name-sg = Algorithme ,
6519   name-sg = algorithme ,
6520   Name-pl = Algorithmes ,
6521   name-pl = algorithmes ,
6522
6523   type = listing ,
6524   gender = m ,
6525   Name-sg = Listing ,
6526   name-sg = listing ,
6527   Name-pl = Listings ,
6528   name-pl = listings ,
6529
6530   type = exercise ,
6531   gender = m ,
6532   Name-sg = Exercice ,
6533   name-sg = exercice ,
6534   Name-pl = Exercices ,
6535   name-pl = exercices ,
6536
6537   type = solution ,
6538   gender = f ,
6539   Name-sg = Solution ,
6540   name-sg = solution ,
6541   Name-pl = Solutions ,
6542   name-pl = solutions ,
6543 </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.

```

6544 <*package>
6545 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6546 \zcDeclareLanguageAlias { brazilian } { portuguese }
6547 \zcDeclareLanguageAlias { brazil } { portuguese }
6548 \zcDeclareLanguageAlias { portuges } { portuguese }
6549 </package>
6550 <*lang-portuguese>
6551 namesep = {\nobreakspace} ,
6552 pairsep = {\sim\nobreakspace} ,
6553 listsep = { , \sim } ,
6554 lastsep = {\sim\nobreakspace} ,
6555 tpairsep = {\sim\nobreakspace} ,
6556 tlistsep = { , \sim } ,
6557 tlastsep = {\sim\nobreakspace} ,
6558 notesep = { \sim } ,
6559 rangesep = { \sim a \nobreakspace } ,
6560
6561 type = book ,
6562 gender = m ,

```

```

6563 Name-sg = Livro ,
6564 name-sg = livro ,
6565 Name-pl = Livros ,
6566 name-pl = livros ,
6567
6568 type = part ,
6569 gender = f ,
6570 Name-sg = Parte ,
6571 name-sg = parte ,
6572 Name-pl = Partes ,
6573 name-pl = partes ,
6574
6575 type = chapter ,
6576 gender = m ,
6577 Name-sg = Capítulo ,
6578 name-sg = capítulo ,
6579 Name-pl = Capítulos ,
6580 name-pl = capítulos ,
6581
6582 type = section ,
6583 gender = f ,
6584 Name-sg = Seção ,
6585 name-sg = seção ,
6586 Name-pl = Seções ,
6587 name-pl = seções ,
6588
6589 type = paragraph ,
6590 gender = m ,
6591 Name-sg = Parágrafo ,
6592 name-sg = parágrafo ,
6593 Name-pl = Parágrafos ,
6594 name-pl = parágrafos ,
6595 Name-sg-ab = Par. ,
6596 name-sg-ab = par. ,
6597 Name-pl-ab = Par. ,
6598 name-pl-ab = par. ,
6599
6600 type = appendix ,
6601 gender = m ,
6602 Name-sg = Apêndice ,
6603 name-sg = apêndice ,
6604 Name-pl = Apêndices ,
6605 name-pl = apêndices ,
6606
6607 type = page ,
6608 gender = f ,
6609 Name-sg = Página ,
6610 name-sg = página ,
6611 Name-pl = Páginas ,
6612 name-pl = páginas ,
6613 rangesep = {\textendash} ,
6614 rangetopair = false ,
6615
6616 type = line ,

```

```

6617     gender = f ,
6618     Name-sg = Linha ,
6619     name-sg = linha ,
6620     Name-pl = Linhas ,
6621     name-pl = linhas ,
6622
6623     type = figure ,
6624     gender = f ,
6625     Name-sg = Figura ,
6626     name-sg = figura ,
6627     Name-pl = Figuras ,
6628     name-pl = figuras ,
6629     Name-sg-ab = Fig. ,
6630     name-sg-ab = fig. ,
6631     Name-pl-ab = Figs. ,
6632     name-pl-ab = figs. ,
6633
6634     type = table ,
6635     gender = f ,
6636     Name-sg = Tabela ,
6637     name-sg = tabela ,
6638     Name-pl = Tabelas ,
6639     name-pl = tabelas ,
6640
6641     type = item ,
6642     gender = m ,
6643     Name-sg = Item ,
6644     name-sg = item ,
6645     Name-pl = Itens ,
6646     name-pl = itens ,
6647
6648     type = footnote ,
6649     gender = f ,
6650     Name-sg = Nota ,
6651     name-sg = nota ,
6652     Name-pl = Notas ,
6653     name-pl = notas ,
6654
6655     type = endnote ,
6656     gender = f ,
6657     Name-sg = Nota ,
6658     name-sg = nota ,
6659     Name-pl = Notas ,
6660     name-pl = notas ,
6661
6662     type = note ,
6663     gender = f ,
6664     Name-sg = Nota ,
6665     name-sg = nota ,
6666     Name-pl = Notas ,
6667     name-pl = notas ,
6668
6669     type = equation ,
6670     gender = f ,

```

```

6671 Name-sg = Equação ,
6672 name-sg = equação ,
6673 Name-pl = Equações ,
6674 name-pl = equações ,
6675 Name-sg-ab = Eq. ,
6676 name-sg-ab = eq. ,
6677 Name-pl-ab = Eqs. ,
6678 name-pl-ab = eqs. ,
6679 refbounds-first-sg = {(,),(,)},
6680 refbounds = {(,,)} ,
6681
6682 type = theorem ,
6683 gender = m ,
6684 Name-sg = Teorema ,
6685 name-sg = teorema ,
6686 Name-pl = Teoremas ,
6687 name-pl = teoremas ,
6688
6689 type = lemma ,
6690 gender = m ,
6691 Name-sg = Lema ,
6692 name-sg = lema ,
6693 Name-pl = Lemas ,
6694 name-pl = lemas ,
6695
6696 type = corollary ,
6697 gender = m ,
6698 Name-sg = Corolário ,
6699 name-sg = corolário ,
6700 Name-pl = Corolários ,
6701 name-pl = corolários ,
6702
6703 type = proposition ,
6704 gender = f ,
6705 Name-sg = Proposição ,
6706 name-sg = proposição ,
6707 Name-pl = Proposições ,
6708 name-pl = proposições ,
6709
6710 type = definition ,
6711 gender = f ,
6712 Name-sg = Definição ,
6713 name-sg = definição ,
6714 Name-pl = Definições ,
6715 name-pl = definições ,
6716
6717 type = proof ,
6718 gender = f ,
6719 Name-sg = Demonstração ,
6720 name-sg = demonstração ,
6721 Name-pl = Demonstrações ,
6722 name-pl = demonstrações ,
6723
6724 type = result ,

```

```

6725     gender = m ,
6726     Name-sg = Resultado ,
6727     name-sg = resultado ,
6728     Name-pl = Resultados ,
6729     name-pl = resultados ,
6730
6731     type = remark ,
6732     gender = f ,
6733     Name-sg = Observação ,
6734     name-sg = observação ,
6735     Name-pl = Observações ,
6736     name-pl = observações ,
6737
6738     type = example ,
6739     gender = m ,
6740     Name-sg = Exemplo ,
6741     name-sg = exemplo ,
6742     Name-pl = Exemplos ,
6743     name-pl = exemplos ,
6744
6745     type = algorithm ,
6746     gender = m ,
6747     Name-sg = Algoritmo ,
6748     name-sg = algoritmo ,
6749     Name-pl = Algoritmos ,
6750     name-pl = algoritmos ,
6751
6752     type = listing ,
6753     gender = f ,
6754     Name-sg = Listagem ,
6755     name-sg = listagem ,
6756     Name-pl = Listagens ,
6757     name-pl = listagens ,
6758
6759     type = exercise ,
6760     gender = m ,
6761     Name-sg = Exercício ,
6762     name-sg = exercício ,
6763     Name-pl = Exercícios ,
6764     name-pl = exercícios ,
6765
6766     type = solution ,
6767     gender = f ,
6768     Name-sg = Solução ,
6769     name-sg = solução ,
6770     Name-pl = Soluções ,
6771     name-pl = soluções ,
6772 </lang-portuguese>

```

10.5 Spanish

Spanish language file has been initially provided by the author.

```

6773 <*package>

```

```

6774 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6775 \</package>
6776 \<*lang-spanish>
6777 namesep = {\nobreakspace} ,
6778 pairsep = {\~y\nobreakspace} ,
6779 listsep = { , ~ } ,
6780 lastsep = {\~y\nobreakspace} ,
6781 tpairsep = {\~y\nobreakspace} ,
6782 tlistsep = { , ~ } ,
6783 tlastsep = {\~y\nobreakspace} ,
6784 notesep = { ~ } ,
6785 rangesep = {\~a\nobreakspace} ,
6786
6787 type = book ,
6788   gender = m ,
6789   Name-sg = Libro ,
6790   name-sg = libro ,
6791   Name-pl = Libros ,
6792   name-pl = libros ,
6793
6794 type = part ,
6795   gender = f ,
6796   Name-sg = Parte ,
6797   name-sg = parte ,
6798   Name-pl = Partes ,
6799   name-pl = partes ,
6800
6801 type = chapter ,
6802   gender = m ,
6803   Name-sg = Capítulo ,
6804   name-sg = capítulo ,
6805   Name-pl = Capítulos ,
6806   name-pl = capítulos ,
6807
6808 type = section ,
6809   gender = f ,
6810   Name-sg = Sección ,
6811   name-sg = sección ,
6812   Name-pl = Secciones ,
6813   name-pl = secciones ,
6814
6815 type = paragraph ,
6816   gender = m ,
6817   Name-sg = Párrafo ,
6818   name-sg = párrafo ,
6819   Name-pl = Párrafos ,
6820   name-pl = párrafos ,
6821
6822 type = appendix ,
6823   gender = m ,
6824   Name-sg = Apéndice ,
6825   name-sg = apéndice ,
6826   Name-pl = Apéndices ,

```

```

6827     name-pl = apéndices ,
6828
6829 type = page ,
6830     gender = f ,
6831     Name-sg = Página ,
6832     name-sg = página ,
6833     Name-pl = Páginas ,
6834     name-pl = páginas ,
6835     rangeseq = {\textendash} ,
6836     rangetopair = false ,
6837
6838 type = line ,
6839     gender = f ,
6840     Name-sg = Línea ,
6841     name-sg = línea ,
6842     Name-pl = Líneas ,
6843     name-pl = líneas ,
6844
6845 type = figure ,
6846     gender = f ,
6847     Name-sg = Figura ,
6848     name-sg = figura ,
6849     Name-pl = Figuras ,
6850     name-pl = figuras ,
6851
6852 type = table ,
6853     gender = m ,
6854     Name-sg = Cuadro ,
6855     name-sg = cuadro ,
6856     Name-pl = Cuadros ,
6857     name-pl = cuadros ,
6858
6859 type = item ,
6860     gender = m ,
6861     Name-sg = Punto ,
6862     name-sg = punto ,
6863     Name-pl = Puntos ,
6864     name-pl = puntos ,
6865
6866 type = footnote ,
6867     gender = f ,
6868     Name-sg = Nota ,
6869     name-sg = nota ,
6870     Name-pl = Notas ,
6871     name-pl = notas ,
6872
6873 type = endnote ,
6874     gender = f ,
6875     Name-sg = Nota ,
6876     name-sg = nota ,
6877     Name-pl = Notas ,
6878     name-pl = notas ,
6879
6880 type = note ,

```

```

6881   gender = f ,
6882   Name-sg = Nota ,
6883   name-sg = nota ,
6884   Name-pl = Notas ,
6885   name-pl = notas ,
6886
6887   type = equation ,
6888   gender = f ,
6889   Name-sg = Ecuación ,
6890   name-sg = ecuación ,
6891   Name-pl = Ecuaciones ,
6892   name-pl = ecuaciones ,
6893   refbounds-first-sg = {(,),(,)},
6894   refbounds = {(,,)} ,
6895
6896   type = theorem ,
6897   gender = m ,
6898   Name-sg = Teorema ,
6899   name-sg = teorema ,
6900   Name-pl = Teoremas ,
6901   name-pl = teoremas ,
6902
6903   type = lemma ,
6904   gender = m ,
6905   Name-sg = Lema ,
6906   name-sg = lema ,
6907   Name-pl = Lemas ,
6908   name-pl = lemas ,
6909
6910   type = corollary ,
6911   gender = m ,
6912   Name-sg = Corolario ,
6913   name-sg = corolario ,
6914   Name-pl = Corolarios ,
6915   name-pl = corolarios ,
6916
6917   type = proposition ,
6918   gender = f ,
6919   Name-sg = Proposición ,
6920   name-sg = proposición ,
6921   Name-pl = Proposiciones ,
6922   name-pl = proposiciones ,
6923
6924   type = definition ,
6925   gender = f ,
6926   Name-sg = Definición ,
6927   name-sg = definición ,
6928   Name-pl = Definiciones ,
6929   name-pl = definiciones ,
6930
6931   type = proof ,
6932   gender = f ,
6933   Name-sg = Demostración ,
6934   name-sg = demostración ,

```



```

6935     Name-pl = Demostraciones ,
6936     name-pl = demostraciones ,
6937
6938     type = result ,
6939     gender = m ,
6940     Name-sg = Resultado ,
6941     name-sg = resultado ,
6942     Name-pl = Resultados ,
6943     name-pl = resultados ,
6944
6945     type = remark ,
6946     gender = f ,
6947     Name-sg = Observación ,
6948     name-sg = observación ,
6949     Name-pl = Observaciones ,
6950     name-pl = observaciones ,
6951
6952     type = example ,
6953     gender = m ,
6954     Name-sg = Ejemplo ,
6955     name-sg = ejemplo ,
6956     Name-pl = Ejemplos ,
6957     name-pl = ejemplos ,
6958
6959     type = algorithm ,
6960     gender = m ,
6961     Name-sg = Algoritmo ,
6962     name-sg = algoritmo ,
6963     Name-pl = Algoritmos ,
6964     name-pl = algoritmos ,
6965
6966     type = listing ,
6967     gender = m ,
6968     Name-sg = Listado ,
6969     name-sg = listado ,
6970     Name-pl = Listados ,
6971     name-pl = listados ,
6972
6973     type = exercise ,
6974     gender = m ,
6975     Name-sg = Ejercicio ,
6976     name-sg = ejercicio ,
6977     Name-pl = Ejercicios ,
6978     name-pl = ejercicios ,
6979
6980     type = solution ,
6981     gender = f ,
6982     Name-sg = Solución ,
6983     name-sg = solución ,
6984     Name-pl = Soluciones ,
6985     name-pl = soluciones ,
6986 </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.

```
6987 <*package>
6988 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6989 </package>

6990 <*lang-dutch>

6991 namesep    = {\nobreakspace} ,
6992 pairsep    = {\sim\nobreakspace} ,
6993 listsep     = {,~} ,
6994 lastsep     = {\sim\nobreakspace} ,
6995 tpairsep    = {\sim\nobreakspace} ,
6996 tlistsep    = {,~} ,
6997 tlastsep    = {,~\sim\nobreakspace} ,
6998 notesep     = {\sim} ,
6999 rangesep    = {\sim/m\nobreakspace} ,
7000
7001 type = book ,
7002   gender = n ,
7003   Name-sg = Boek ,
7004   name-sg = boek ,
7005   Name-pl = Boeken ,
7006   name-pl = boeken ,
7007
7008 type = part ,
7009   gender = n ,
7010   Name-sg = Deel ,
7011   name-sg = deel ,
7012   Name-pl = Delen ,
7013   name-pl = delen ,
7014
7015 type = chapter ,
7016   gender = n ,
7017   Name-sg = Hoofdstuk ,
7018   name-sg = hoofdstuk ,
7019   Name-pl = Hoofdstukken ,
7020   name-pl = hoofdstukken ,
7021
7022 type = section ,
7023   gender = m ,
7024   Name-sg = Paragraaf ,
7025   name-sg = paragraaf ,
7026   Name-pl = Paragrafen ,
7027   name-pl = paragrafen ,
7028
7029 type = paragraph ,
7030   gender = f ,
7031   Name-sg = Alinea ,
7032   name-sg = alinea ,
7033   Name-pl = Alinea's ,
7034   name-pl = alinea's ,
7035
```

```

7036 type = appendix ,
7037     gender = { m , n } ,
7038     Name-sg = Appendix ,
7039     name-sg = appendix ,
7040     Name-pl = Appendices ,
7041     name-pl = appendices ,
7042
7043 type = page ,
7044     gender = { f , m } ,
7045     Name-sg = Pagina ,
7046     name-sg = pagina ,
7047     Name-pl = Pagina's ,
7048     name-pl = pagina's ,
7049     rangesep = {\textendash} ,
7050     rangetopair = false ,
7051
7052 type = line ,
7053     gender = m ,
7054     Name-sg = Regel ,
7055     name-sg = regel ,
7056     Name-pl = Regels ,
7057     name-pl = regels ,
7058
7059 type = figure ,
7060     gender = { n , f , m } ,
7061     Name-sg = Figuur ,
7062     name-sg = figuur ,
7063     Name-pl = Figuren ,
7064     name-pl = figuren ,
7065
7066 type = table ,
7067     gender = { f , m } ,
7068     Name-sg = Tabel ,
7069     name-sg = tabel ,
7070     Name-pl = Tabellen ,
7071     name-pl = tabellen ,
7072
7073 type = item ,
7074     gender = n ,
7075     Name-sg = Punt ,
7076     name-sg = punt ,
7077     Name-pl = Punten ,
7078     name-pl = punten ,
7079
7080 type = footnote ,
7081     gender = { f , m } ,
7082     Name-sg = Voetnoot ,
7083     name-sg = voetnoot ,
7084     Name-pl = Voetnoten ,
7085     name-pl = voetnoten ,
7086
7087 type = endnote ,
7088     gender = { f , m } ,
7089     Name-sg = Eindnoot ,

```

```

7090 name-sg = eindnoot ,
7091 Name-pl = Eindnoten ,
7092 name-pl = eindnoten ,
7093
7094 type = note ,
7095 gender = f ,
7096 Name-sg = Opmerking ,
7097 name-sg = opmerking ,
7098 Name-pl = Opmerkingen ,
7099 name-pl = opmerkingen ,
7100
7101 type = equation ,
7102 gender = f ,
7103 Name-sg = Vergelijking ,
7104 name-sg = vergelijking ,
7105 Name-pl = Vergelijkingen ,
7106 name-pl = vergelijkingen ,
7107 Name-sg-ab = Vgl. ,
7108 name-sg-ab = vgl. ,
7109 Name-pl-ab = Vgl.'s ,
7110 name-pl-ab = vgl.'s ,
7111 refbounds-first-sg = {(,),(,)},
7112 refbounds = {(,,)},
7113
7114 type = theorem ,
7115 gender = f ,
7116 Name-sg = Stelling ,
7117 name-sg = stelling ,
7118 Name-pl = Stellingen ,
7119 name-pl = stellingen ,
7120

```

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”.

```

7121 type = lemma ,
7122 gender = n ,
7123 Name-sg = Lemma ,
7124 name-sg = lemma ,
7125 Name-pl = Lemma's ,
7126 name-pl = lemma's ,
7127
7128 type = corollary ,
7129 gender = n ,
7130 Name-sg = Gevolg ,
7131 name-sg = gevolg ,
7132 Name-pl = Gevolgen ,
7133 name-pl = gevolgen ,
7134
7135 type = proposition ,
7136 gender = f ,
7137 Name-sg = Propositie ,
7138 name-sg = propositie ,
7139 Name-pl = Proposities ,

```

```

7140     name-pl = proposities ,
7141
7142     type = definition ,
7143     gender = f ,
7144     Name-sg = Definitie ,
7145     name-sg = definitie ,
7146     Name-pl = Definities ,
7147     name-pl = definities ,
7148
7149     type = proof ,
7150     gender = n ,
7151     Name-sg = Bewijs ,
7152     name-sg = bewijs ,
7153     Name-pl = Bewijzen ,
7154     name-pl = bewijzen ,
7155
7156     type = result ,
7157     gender = n ,
7158     Name-sg = Resultaat ,
7159     name-sg = resultaat ,
7160     Name-pl = Resultaten ,
7161     name-pl = resultaten ,
7162
7163     type = remark ,
7164     gender = f ,
7165     Name-sg = Opmerking ,
7166     name-sg = opmerking ,
7167     Name-pl = Opmerkingen ,
7168     name-pl = opmerkingen ,
7169
7170     type = example ,
7171     gender = n ,
7172     Name-sg = Voorbeeld ,
7173     name-sg = voorbeeld ,
7174     Name-pl = Voorbeelden ,
7175     name-pl = voorbeelden ,
7176
7177     type = algorithm ,
7178     gender = { n , f , m } ,
7179     Name-sg = Algoritme ,
7180     name-sg = algoritme ,
7181     Name-pl = Algoritmes ,
7182     name-pl = algoritmes ,
7183

```

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

```

7184     type = listing ,
7185     gender = m ,
7186     Name-sg = Listing ,
7187     name-sg = listing ,
7188     Name-pl = Listings ,
7189     name-pl = listings ,
7190

```

```

7191 type = exercise ,
7192     gender = { f , m } ,
7193     Name-sg = Opgave ,
7194     name-sg = opgave ,
7195     Name-pl = Opgaven ,
7196     name-pl = opgaven ,
7197
7198 type = solution ,
7199     gender = f ,
7200     Name-sg = Oplossing ,
7201     name-sg = oplossing ,
7202     Name-pl = Oplossingen ,
7203     name-pl = oplossingen ,
7204 </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		4437, 4448, 4454, 4482, 4527, 4549,
\A	1702		4578, 4725, 4883, 4885, 5492, 5570
\AddToHook	100, 1848, 1895, 1919, 1950, 1952, 1990, 2063, 2079, 2092, 2116, 2247, 2260, 2268, 5257, 5293, 5310, 5312, 5317, 5362, 5364, 5366, 5368, 5370, 5372, 5374, 5376, 5380, 5384, 5386, 5391, 5401, 5410, 5416, 5418, 5444, 5455, 5489, 5542, 5567	\bool_if:nTF	68, 3490, 3500, 3524, 3541, 3556, 3621, 3629, 4270, 4644, 4686, 4772, 4789
\alph	5470, 5558	\bool_if_exist:Ntf	381
\appendix	2, 124, 127	\bool_lazy_all:nTF	4367, 4895
\appendixname	124	\bool_lazy_and:nnTF ...	326, 3347, 3368, 4151, 4222, 4601, 4868, 4910
\AtEndOfPackage	2258	\bool_lazy_any:nTF	5047, 5056
	B	\bool_lazy_or:nnTF	1190, 1226, 1778, 2340, 2601, 3202, 3238, 3351, 4856
\babelname	1905	\bool_new:N	1392, 1393, 1420, 1444, 1796, 1803, 1810, 1823, 1824, 1998, 1999, 2000, 2001, 2002, 2109, 2110, 2227, 3384, 3399, 3661, 3662, 3673, 3674, 3681, 3683, 3684, 3697, 3698, 3699, 3711, 3712, 5454, 5501, 5541
\babelprovide	25, 50	\bool_set:Nn	3344
\bicaption	126	\bool_set_eq:NN	388
\bionenumcaption	126	\bool_set_false:N	
\bitwonumcaption	126		1405, 1409, 1581, 1679, 1724, 1766, 1831, 1840, 1841, 1858, 1865, 2010, 2014, 2021, 2029, 2030, 2031, 2132, 2370, 2638, 3389, 3482, 3728, 3729, 3746, 3785, 3796, 4196, 4389, 4390, 4397, 4398, 5054, 5071
bool commands:		\bool_set_true:N	1399, 1400, 1404, 1410, 1603, 1625, 1688, 1690, 1728, 1730, 1739, 1741, 1770, 1772, 1785, 1787, 1830, 1835, 1836, 2008, 2015, 2020, 2037, 2039, 2041,
\bool_case_true:	2		
\bool_gset_false:N			
.....	1308, 1325, 3294, 3302		
\bool_gset_true:N			
.....	1270, 1287, 2234, 3273, 3281		
\bool_if:Ntf			
.....	398, 1578, 1629, 1852, 1856, 2270, 3376, 3779, 3920, 4047, 4083, 4117, 4184, 4197, 4209, 4260, 4277, 4287, 4292, 4339, 4343, 4399, 4424, 4431.		

2044, 2045, 2046, 2120, 2125, 2365, 2629, 3496, 3506, 3510, 3532, 3547, 3562, 3586, 3754, 3780, 3786, 3790, 3797, 3943, 3954, 3965, 4015, 4051, 4087, 4121, 4138, 4219, 4253, 4426, 4486, 4532, 4554, 4583, 4844, 4851, 4973, 5032, 5070, 5101, 5108, 5109, 5127, 5144, 5146, 5468, 5508, 5555	
\bool_until_do:Nn	1680, 1731, 1773, 3522, 3747
\bool_while_do:nn	5602
\l_tmpa_bool	1581, 1603, 1625, 1629, 1679, 1680, 1688, 1690, 1724, 1728, 1730, 1731, 1739, 1741, 1766, 1770, 1772, 1773, 1785, 1787
C	
\caption	129
clist commands:	
\clist_map_inline:nn	468, 806, 2032, 2175, 2236, 2749, 5471, 5560
\contsubbottom	127
\contsubcaption	127
\counterwithin	5
\crefstripprefix	41
cs commands:	
\cs_generate_variant:Nn	65, 268, 274, 281, 292, 303, 314, 322, 323, 337, 357, 358, 378, 379, 758, 1662, 1698, 1749, 1795, 2396, 4680, 4719, 5084, 5188, 5221, 5254
\cs_gset_eq:NN	356, 377
\cs_if_exist:NTF	25, 28, 46, 49, 58, 78, 5281, 5287, 5408
\cs_if_exist_p:N	4153, 4224, 4603, 5604
\cs_if_exist_use:N	5379, 5642
\cs_new:Npn	56, 66, 76, 87, 269, 275, 277, 279, 282, 293, 304, 316, 523, 4640, 4681, 4720, 5074
\cs_new_eq:NN	5446, 5451
\cs_new_protected:Npn ..	263, 318, 320, 332, 349, 351, 353, 355, 374, 376, 600, 702, 1341, 1360, 1564, 1663, 1708, 1750, 2266, 2394, 3339, 3403, 3445, 3456, 3469, 3599, 3651, 3713, 3927, 4393, 4636, 4638, 4839, 5085, 5156, 5189, 5222, 5352, 5509
\cs_new_protected:Npx	99
\cs_set:Npn	5355
\cs_set_eq:NN	103, 354, 375, 525, 5354, 5360, 5447, 5452
\cs_set_nopar:Npn	5437
D	
\d	1702
E	
\endinput	12
\endsidecaption	126
exp commands:	
\exp_args:NNe	35, 38
\exp_args:NNNo	265
\exp_args:NNNV	1643, 1692, 1743, 1789
\exp_args:NNo	265, 271
\exp_args:No	271
\exp_args:Nx	712, 5514, 5522
\exp_args:Nxx	1582, 1594, 1606, 1616, 1682, 1733, 1775, 5087, 5093, 5115, 5119, 5134
\exp_not:N	111, 4294, 4297, 4308, 4311, 4314, 4650, 4654, 4664, 4667, 4675, 4692, 4696, 4705, 4708, 4715, 4731, 4733, 4743, 4746, 4748, 4756, 4760, 4763, 4766, 4778, 4781, 4795, 4800, 4820, 4823, 4832
\exp_not:n ..	272, 3959, 3982, 3990, 4008, 4021, 4025, 4060, 4069, 4091, 4099, 4106, 4130, 4144, 4161, 4171, 4213, 4236, 4246, 4295, 4307, 4312, 4313, 4470, 4493, 4505, 4539, 4561, 4570, 4590, 4611, 4621, 4651, 4668, 4693, 4701, 4709, 4712, 4744, 4745, 4747, 4749, 4758, 4764, 4765, 4767, 4775, 4779, 4780, 4783, 4796, 4824
\ExplSyntaxOn	25, 718
F	
file commands:	
\file_get:nnNTF	712
\fmtversion	3
\footnote	2, 127, 128
G	
group commands:	
\group_begin:	102, 539, 704, 1580, 1667, 1712, 1754, 2661, 3341, 3355, 3387, 4294, 4311, 4650, 4667, 4692, 4708, 4731, 4743, 4748, 4763, 4778, 4795, 4823, 5367, 5371, 5375
\group_end:	105, 551, 756, 1644, 1693, 1744, 1790, 2689, 3358, 3381, 3391, 4308, 4314, 4664, 4675, 4705, 4715, 4746, 4756, 4760, 4766, 4781, 4820, 4832, 5369, 5373, 5377
I	
\IfBooleanT	3388
\IfClassLoadedTF	112
\ifdraft	2013
\IfFormatAtLeastTF	3, 4
\ifoptionfinal	2019

\nobreakspace	1354, 5678, 5679, 5681, 5682, 5684, 5686, 5884, 5885, 5887, 5888, 5890, 5892, 6334, 6335, 6337, 6338, 6340, 6342, 6551, 6552, 6554, 6555, 6557, 6559, 6777, 6778, 6780, 6781, 6783, 6785, 6991, 6992, 6994, 6995, 6997, 6999
\NumCheckSetup	41
\NumsCheckSetup	41
P	
\PackageError	7
\pagenote	127, 128
\pagenumbering	7
\pageref	82
\PagesCheckSetup	41
\paragraph	57
prg commands:	
\prg_generate_conditional_- variant:Nnn	347, 361, 372, 382, 393, 401, 533, 1706
\prg_new_conditional:Npnn	109, 111, 324, 359, 380, 395, 527
\prg_new_protected_conditional:Npnn	338, 363, 384, 1699
\prg_return_false:	110, 112, 330, 345, 360, 370, 381, 391, 398, 399, 531, 1704
\prg_return_true:	110, 112, 329, 343, 360, 368, 381, 389, 398, 530, 1703
\prg_set_eq_conditional:NNn	535
\ProcessKeysOptions	2391
prop commands:	
\prop_if_in:NnTF	35
\prop_if_in_p:Nn	69
\prop_item:Nn	38, 70
\prop_new:N	2141, 2196
\prop_put:Nnn	1364
\prop_remove:Nn	1363
\providecommand	3
\ProvidesExplPackage	14
\ProvidesFile	25
R	
\refstepcounter	3, 126, 129–131, 133, 134
regex commands:	
\regex_match:nnTF	1702
\relax	15
\renewlist	134, 135
\RequirePackage	16, 17, 18, 19, 20, 1853, 2076, 2089, 2113
S	
scan commands:	
\scan_stop:	15, 354, 356, 375, 377
\scantokens	124
seq commands:	
\seq_clear:N	611, 648, 729, 742, 805, 1441, 2337, 2598, 2672, 2685, 2748, 3405, 5215
\seq_const_from_clist:Nn	408, 416, 427, 441, 448
\seq_count:N	1194, 1208, 1230, 1245, 2342, 2351, 2603, 2615, 3206, 3220, 3242, 3256
\seq_gclear:N	1187, 1223, 3199, 3235
\seq_gconcat:NNN	461, 465
\seq_get_left:NN	626, 637, 733, 781, 2676, 2714, 3757
\seq_gput_right:Nn	744, 750, 2240
\seq_gremove_all:Nn	2272
\seq_gset_eq:NN	825, 1197, 1233, 2759, 3209, 3245
\seq_gset_from_clist:Nn	571, 581
\seq_gset_split:Nnn	352
\seq_if_empty:NnTF	612, 649, 730, 770, 791, 2673, 2703, 2724, 3751, 4922
\seq_if_exist:NnTF	360
\seq_if_in:NnTF	630, 664, 708, 776, 808, 2177, 2238, 2271, 2709, 2751, 3449, 4918
\seq_item:Nn	4652, 4658, 4661, 4663, 4669, 4670, 4673, 4674, 4694, 4700, 4702, 4704, 4710, 4711, 4713, 4714, 4750, 4751, 4755, 4759, 4797, 4810, 4815, 4818, 4825, 4826, 4830, 4831
\seq_map_break:n	90, 3642, 3645
\seq_map_function:NN	3408
\seq_map_indexed_inline:Nn	40, 3603
\seq_map_inline:Nn	840, 868, 1124, 1169, 1253, 2262, 2275, 2323, 2357, 2403, 2415, 2581, 2621, 2773, 2807, 3127, 3179, 3263, 3639, 5512
\seq_map_tokens:Nn	72
\seq_new:N	406, 407, 460, 464, 701, 1429, 2170, 2228, 3383, 3402, 3660, 3677, 3700, 3701, 3702, 3703, 3704, 3705, 3706, 3707, 3708, 3709, 3710
\seq_pop_left:NN	3749
\seq_put_right:Nn	809, 2179, 2752, 3452
\seq_reverse:N	1435
\seq_set_eq:NN	367, 2344, 2605, 3715, 3941, 3952, 3963, 4013, 4049, 4085, 4119, 4135, 4217, 4251, 4262, 4484, 4529, 4551, 4580
\seq_set_from_clist:Nn	1434, 3343
\seq_set_split:Nnn	350
\seq_sort:Nn	85, 3411
\seq_use:Nn	4936

<code>\g_tmpa_seq</code>	1187, 1189, 1194, 1203, 1208, 1223, 1225, 1230, 1240, 1245, 3199, 3201, 3206, 3215, 3220, 3235, 3237, 3242, 3251, 3256
<code>\l_tmpa_seq</code>	805, 809, 833, 2337, 2339, 2342, 2346, 2351, 2598, 2600, 2603, 2610, 2615, 2748, 2752, 2767
<code>\setcounter</code>	5279, 5280, 5296, 5311, 5315
<code>\sidefootnote</code>	127, 128
sort commands:	
<code>\sort_return_same:</code>	85, 89, 3418, 3423, 3497, 3517, 3538, 3553, 3567, 3592, 3627, 3642, 3658
<code>\sort_return_swapped:</code>	85, 89, 3431, 3507, 3516, 3537, 3552, 3568, 3591, 3635, 3645, 3657
<code>\stepcounter</code>	133, 5295, 5314
str commands:	
<code>\str_case:nn</code>	40
<code>\str_case:nnTF</code>	909, 1458, 1921, 1958, 2034, 2446, 2863
<code>\str_compare:nNnTF</code>	3513
<code>\str_if_eq:nnTF</code>	89, 4683
<code>\str_if_eq_p:nn</code>	5052, 5058, 5060, 5064
<code>\str_new:N</code>	1868
<code>\str_set:Nn</code>	1873, 1875, 1877, 1879
<code>\string</code>	5519, 5527
<code>\subbottom</code>	127
<code>\subcaption</code>	127
<code>\subcaptionref</code>	126
<code>\subref</code>	136
<code>\subsubsection</code>	57
<code>\subsubsubsection</code>	57
<code>\subtop</code>	127
T	
<code>\tag</code>	121, 131, 133
\TeX and \LaTeX 2 ϵ commands:	
<code>\@sidecaption</code>	126
<code>\@Alph</code>	124
<code>\@addtoreset</code>	5
<code>\@auxout</code>	5518, 5526
<code>\@bsphack</code>	705, 5511
<code>\@capytype</code>	129, 5379, 5412, 5413, 5417, 5642
<code>\@chapapp</code>	124
<code>\@currentcounter</code>	2, 3, 5, 58, 127, 129–131, 134, 28, 29, 49, 50, 2225, 5413, 5425
<code>\@currentlabel</code>	3, 121, 127, 129, 134
<code>\@currenvir</code>	5420
<code>\@elt</code>	5
<code>\@esphack</code>	755, 5531
<code>\@ifl@t@r</code>	3
<code>\@mem@scap@afterhook</code>	126
<code>\@memsubcaption</code>	127
<code>\@onlypreamble</code>	553, 566, 2691
<code>\@raw@opt@{package}.sty</code>	54
<code>\bbl@loaded</code>	50
<code>\bbl@main@language</code>	50, 1900
<code>\c@lstnumber</code>	134
<code>\c@page</code>	7, 103
<code>\caption@subtypehook</code>	5643
<code>\hyper@link</code>	110, 4297, 4654, 4696, 4733, 4800
<code>\lst@AddToHook</code>	5590, 5592
<code>\lst@Init</code>	134
<code>\lst@label</code>	5591
<code>\lst@MakeCaption</code>	134
<code>\ltx@gobble</code>	130
<code>\ltx@label</code>	130, 5446, 5447, 5451, 5452
<code>\m@mscaplabel</code>	126
<code>\MT@newlabel</code>	5519, 5527
<code>\protected@write</code>	5518, 5526
<code>\zref@addprop</code>	22, 32, 43, 53, 55, 97, 108
<code>\zref@default</code>	110, 111, 4637, 4639
<code>\zref@extractdefault</code>	10, 11, 119, 266, 272, 276
<code>\zref@ifpropundefined</code>	38, 1077, 1378, 1540, 2528, 3053, 5076, 5557
<code>\zref@ifrefcontainsprop</code>	38, 40, 1568, 1576, 1635, 1653, 1665, 1710, 1752, 3814, 4642, 4727, 4785, 5079
<code>\zref@ifrefundefined</code>	3413, 3415, 3427, 3782, 3784, 3789, 3806, 4188, 4199, 4401, 4722, 4841
<code>\zref@label</code>	129, 5440
<code>\zref@localaddprop</code>	5381, 5493, 5571, 5644
<code>\ZREF@mainlist</code>	22, 32, 43, 53, 55, 97, 108, 5381, 5493, 5571, 5644
<code>\zref@newprop</code>	5, 7, 21, 23, 33, 44, 54, 92, 107, 5378, 5470, 5558, 5641
<code>\zref@refused</code>	3804
<code>\zref@wrapper@babel</code>	81, 129, 3338, 5440
<code>\textendash</code>	1358, 5733, 5998, 6613, 6835, 7049
<code>\textup</code>	132
<code>\thechapter</code>	124
<code>\thelstnumber</code>	134
<code>\thepage</code>	7, 104
<code>\thesection</code>	124
tl commands:	
<code>\c_novaluel_tl</code>	319, 321, 485, 492, 499, 505, 511, 518, 520, 788, 845, 873, 1129, 1174, 2280, 2328, 2420, 2586, 2721, 2778, 3184

<code>\tl_clear:N</code>	<code>\tl_new:N</code>
.... 620, 658, 671, 688, 697, 722,	98, 403, 404,
731, 764, 1462, 1467, 1482, 1497,	405, 545, 1366, 1892, 1893, 1894,
1512, 2401, 2450, 2455, 2470, 2485,	1984, 2003, 2071, 2103, 2220, 3393,
2500, 2664, 2674, 2697, 3717, 3718,	3394, 3395, 3396, 3397, 3398, 3666,
3719, 3720, 3721, 3722, 3753, 4379,	3667, 3668, 3669, 3670, 3671, 3672,
4380, 4381, 4382, 4383, 4429, 4446,	3675, 3676, 3680, 3682, 3685, 3686,
4843, 4850, 4880, 4972, 5031, 5182	3687, 3688, 3689, 3690, 3691, 3692,
<code>\tl_const:Nn</code>	3693, 3694, 3695, 3696, 5382, 5415
1343	<code>\tl_put_left:Nn</code> . 4273, 4280, 4324,
<code>\tl_gclear:N</code>	4958, 4959, 5001, 5003, 5005, 5007
... 2869, 2875, 2883, 2890, 2911,	<code>\tl_put_right:Nn</code> . 3957, 3980, 3988,
2927, 2948, 2964, 2985, 3001, 5428	4006, 4019, 4058, 4067, 4089, 4097,
<code>\tl_gset:Nn</code>	4104, 4128, 4142, 4159, 4169, 4468,
104,	4491, 4503, 4537, 4559, 4568, 4588,
335, 546, 560, 2792, 2826, 2846,	4609, 4619, 4866, 4867, 4878, 5643
2904, 2919, 2941, 2956, 2978, 2993,	<code>\tl_reverse:N</code>
3061, 3068, 3077, 3085, 3154, 3164	3473, 3476
<code>\tl_gset_eq:NN</code>	<code>\tl_set:Nn</code>
321, 5417	... 265, 547, 721, 765, 777, 1375,
<code>\tl_head:N</code>	1382, 1384, 1476, 1491, 1506, 1546,
.. 3551, 3564, 3576, 3578, 3588, 3590	1552, 1573, 1641, 1645, 1658, 1669,
<code>\tl_head:n</code> 1683, 1734, 1776, 1780	1674, 1685, 1686, 1694, 1696, 1714,
<code>\tl_if_empty:NTF</code>	1719, 1736, 1737, 1745, 1747, 1756,
..... 80, 614, 624, 651, 662,	1761, 1782, 1783, 1791, 1793, 1899,
683, 690, 797, 848, 876, 913, 952,	1900, 1905, 1906, 1909, 1910, 1914,
991, 1030, 1083, 1132, 1138, 1177,	1925, 1931, 1936, 1962, 1968, 1973,
1261, 1299, 1566, 1687, 1738, 2730,	2289, 2399, 2432, 2464, 2479, 2494,
2781, 2815, 2867, 2902, 2939, 2976,	2534, 2541, 2665, 2698, 2710, 3580,
3013, 3059, 3135, 3152, 3187, 3271,	3582, 3763, 3764, 3937, 3939, 4211,
3292, 3313, 3359, 4186, 4782, 4865,	4234, 4244, 4290, 4414, 4416, 4427,
4887, 4945, 4956, 4999, 5422, 5591	4444, 4862, 4863, 4876, 5383, 5385
<code>\tl_if_empty:nTF</code>	<code>\tl_set_eq:NN</code> 319, 342, 4377, 5413, 5424
..... 540, 556, 763, 1071, 1362,	<code>\tl_show:N</code>
1371, 1534, 2233, 2522, 2696, 3047	4340
<code>\tl_if_empty_p:N</code> 4152, 4223,	<code>\tl_tail:N</code>
4602, 4898, 4912, 5051, 5061, 5065	3581, 3583
<code>\tl_if_empty_p:n</code> 1191, 1227,	<code>\tl_tail:n</code>
2341, 2602, 3203, 3239, 3492, 3493,	.. 1685, 1686, 1736, 1737, 1782, 1783
3502, 3503, 3528, 3529, 3544, 3559	<code>\tl_use:N</code>
<code>\tl_if_eq:NNTF</code> 3463, 3486, 3793	287,
<code>\tl_if_eq:NnTF</code>	298, 309, 561, 710, 715, 745, 747, 751
1592, 3406, 3438, 3613, 3616, 3641,	<code>\l_tmpa_tl</code>
3644, 3761, 3809, 4847, 5091, 5420	719, 743, 1669,
<code>\tl_if_eq:nnTF</code> .. 1582, 1594, 1606,	1683, 1685, 1714, 1725, 1734, 1736,
1616, 1682, 1733, 1775, 3605, 5087,	1756, 1767, 1776, 1782, 3364, 3365
5093, 5115, 5119, 5134, 5514, 5522	<code>\l_tmpb_tl</code> . 1631, 1638, 1641, 1645,
<code>\tl_if_exist:NTF</code>	1674, 1683, 1686, 1687, 1694, 1719,
529, 5412	1727, 1734, 1737, 1738, 1745, 1761,
<code>\tl_if_exist_p:N</code>	1769, 1776, 1779, 1780, 1783, 1791
327	
<code>\tl_if_novalue:nTF</code>	
803,	
850, 878, 890, 1140, 1153, 1179,	
1214, 2283, 2331, 2423, 2589, 2736,	
2783, 2817, 2835, 3141, 3189, 3225	
<code>\tl_if_novalue_p:n</code>	
328	
<code>\tl_map_break:n</code>	
90	
<code>\tl_map_tokens:Nn</code>	
82	

	U
<code>\upshape</code>	5497
use commands:	
<code>\use:N</code> 26, 29, 592, 4155, 4230, 4605, 5248	
<code>\UseHook</code>	1668, 1713, 1755

	V
<code>\value</code>	5296, 5315
<code>\verbfootnote</code>	127, 128

Z

<code>\Z</code>	1702	<code>\l__zrefclever_current_counter_-</code>	
<code>\zcDeclareLanguage</code>	12, 20,	<code>tl</code>	3, 5, 58, 21, 25,
<code>\zcDeclareLanguageAlias</code>	537, 5668, 5873, 6327, 6545, 6774, 6988		26, 36, 39, 41, 46, 47, 95, 2220, 2223
<code>\zcLanguageSetup</code> 17, 24, 25, 63, 68, 69, 2659		<code>\l__zrefclever_current_language_-</code>	
<code>\zcpageref</code>	82, 3385	<code>tl</code>	50,
<code>\zcRefTypeSetup</code>	17, 63, 2397, 5496		1893, 1899, 1905, 1909, 1926, 1963
<code>\zcsetup</code>	50, 54, 60, 62, 2392	<code>\l__zrefclever_endrangefunc_tl</code> ..	
<code>\zlabel</code> 126, 129, 131, 133, 134, 5358, 5591		...	3685, 3860, 4152, 4153, 4155,
<code>zrefcheck</code> commands:			4223, 4224, 4230, 4602, 4603, 4605
<code>\zrefcheck_zcRef_beg_label:</code> ..	3350	<code>\l__zrefclever_endrangeprop_tl</code> ..	
<code>\zrefcheck_zcRef_end_label_-</code>		43, 1566, 1576, 3685, 3864
<code>maybe:</code>	3372	<code>__zrefclever_extract:nnn</code>	
<code>\zrefcheck_zcRef_run_checks_on_-</code>		11, 275, 1671, 1676,
<code>labels:n</code>	3373		1716, 1721, 1758, 1763, 3534, 3536,
<code>zrefclever</code> commands:			3549, 3566, 3654, 3656, 5098, 5100,
<code>\zrefclever_language_if_declared:nTF</code>		<code>__zrefclever_extract_default:Nnnn</code>	
.....	535	10, 263, 1570, 1631,
<code>\zrefclever_language_varname:n</code> .	525		1638, 1648, 1655, 3447, 3458, 3460,
<code>\l_zrefclever_ref_language_tl</code> .	1914		3471, 3474, 3477, 3479, 3767, 3770
<code>zrefclever</code> internal commands:		<code>__zrefclever_extract_unexp:nnn</code> .	
<code>\l__zrefclever_abbrev_bool</code>	3685, 3872, 4869	..	11, 120, 269, 1584, 1588, 1596,
<code>\l__zrefclever_amsmath_subequations_-</code>			1600, 1608, 1612, 1618, 1622, 4303,
<code>bool</code>	5454, 5468, 5492		4656, 4659, 4671, 4698, 4739, 4752,
<code>\l__zrefclever_breqn_dgroup_bool</code>			4806, 4812, 4827, 5077, 5080, 5081,
.....	5541, 5555, 5570		5088, 5089, 5094, 5095, 5116, 5117,
<code>\l__zrefclever_cap_bool</code>	3685, 3868, 4857		5120, 5121, 5136, 5140, 5515, 5523
<code>\l__zrefclever_capfirst_bool</code> ..		<code>__zrefclever_extract_url_-</code>	
.....	1803, 1806, 4859	<code>unexp:n</code>	
<code>__zrefclever_compat_module:nn</code>	4299, 4655, 4697, 4735, 4802, 5074
.....	59,	<code>__zrefclever_get_enclosing_-</code>	
<code>__zrefclever_counter_reset_by:n</code>		<code>counters_value:n</code> .	5, 6, 56, 61, 94
..	6, 57, 58, 58, 60, 62, 66, 5462, 5549	<code>__zrefclever_get_endrange_-</code>	
<code>__zrefclever_counter_reset_by_-</code>		<code>pagecomp:nnN</code>	1708, 1749
<code>aux:nn</code>	73, 76	<code>__zrefclever_get_endrange_-</code>	
<code>__zrefclever_counter_reset_by_-</code>		<code>pagecomptwo:nnN</code>	1750, 1795
<code>auxi:nnn</code>	83, 87	<code>__zrefclever_get_endrange_-</code>	
<code>\l__zrefclever_counter_resetby_-</code>		<code>property:nnN</code>	40, 1564, 1662
<code>prop</code>	5, 58, 69, 70, 2196, 2208	<code>__zrefclever_get_endrange_-</code>	
<code>\l__zrefclever_counter_resettters_-</code>		<code>stripprefix:nnN</code>	1663, 1698
<code>seq</code> .	5, 57, 58, 72, 2170, 2177, 2180	<code>__zrefclever_get_ref:nN</code>	
<code>\l__zrefclever_counter_type_prop</code>		110, 111, 3960, 3983,
.....	4, 56, 35, 38, 2141, 2153		3991, 4009, 4022, 4026, 4061, 4070,
			4092, 4100, 4107, 4131, 4145, 4172,
			4214, 4247, 4282, 4471, 4494, 4506,
			4540, 4562, 4571, 4591, 4622, 4640
		<code>__zrefclever_get_ref_endrange:nnN</code>	
		40,
			41, 111, 112, 4162, 4237, 4612, 4681
		<code>__zrefclever_get_ref_first:</code> ..	
		110, 111, 115, 4274, 4325, 4720
		<code>__zrefclever_get_rf_opt_bool:nN</code>	123
		<code>__zrefclever_get_rf_opt_-</code>	
		<code>bool:nnnnN</code>	16, 3865, 3869, 3873, 5222

```

\__zrefclever_get_rf_opt_-
  seq:nnnN ..... 16, 122,
  3877, 3881, 3885, 3889, 3893, 3897,
  3901, 3905, 3909, 3913, 4914, 5189
\__zrefclever_get_rf_opt_tl:nnnN
  ..... 16, 18, 41, 122, 3361, 3732,
  3736, 3740, 3829, 3833, 3837, 3841,
  3845, 3849, 3853, 3857, 3861, 5156
\l_zrefclever_hyperlink_bool ...
  1823, 1830, 1835, 1840, 1852, 1858,
  1865, 3389, 4646, 4688, 4791, 5049
\l_zrefclever_hyperref_warn_-
  bool ... 1824, 1831, 1836, 1841, 1856
\__zrefclever_if_class_loaded:n 109
\__zrefclever_if_class_loaded:nTF
  ..... 5336
\__zrefclever_if_package_-
  loaded:n ..... 109
\__zrefclever_if_package_-
  loaded:nTF ..... 1850,
  1897, 1903, 2118, 5275, 5435, 5442,
  5504, 5539, 5579, 5599, 5626, 5649
\__zrefclever_is_integer_rgx:n 1699
\__zrefclever_is_integer_rgx:nTF
  ..... 1725, 1727, 1767, 1769
\g__zrefclever_koma_captionofbeside_-
  capttype_tl ..... 5415
\g__zrefclever_koma_capttype_tl ..
  ..... 5417, 5422, 5425, 5428
\l_zrefclever_label_a_tl .....
  . 90, 3666, 3750, 3769, 3782, 3804,
  3806, 3812, 3815, 3821, 3938, 3960,
  3983, 3991, 4026, 4092, 4107, 4157,
  4163, 4172, 4204, 4214, 4232, 4238,
  4247, 4401, 4405, 4415, 4428, 4445,
  4471, 4507, 4571, 4607, 4613, 4622
\l_zrefclever_label_b_tl .....
  ..... 90, 3666,
  3753, 3758, 3772, 3784, 3789, 4405
\l_zrefclever_label_count_int ..
  ..... 90, 3663, 3723,
  3827, 3930, 4384, 4410, 4634, 4901
\l_zrefclever_label_enclval_a_-
  tl ..... 3393, 3471, 3473, 3528,
  3544, 3564, 3576, 3580, 3581, 3588
\l_zrefclever_label_enclval_b_-
  tl ..... 3393, 3474, 3476, 3529,
  3551, 3559, 3578, 3582, 3583, 3590
\l_zrefclever_label_extdoc_a_tl
  .. 3393, 3477, 3487, 3492, 3502, 3515
\l_zrefclever_label_extdoc_b_tl
  .. 3393, 3479, 3488, 3493, 3503, 3514
\l_zrefclever_label_type_a_tl ..
  ..... 3362, 3393, 3448, 3450,
  3453, 3459, 3464, 3613, 3641, 3733,
  3737, 3741, 3763, 3768, 3794, 3809,
  3830, 3834, 3838, 3842, 3846, 3850,
  3854, 3858, 3862, 3866, 3870, 3874,
  3878, 3882, 3886, 3890, 3894, 3898,
  3902, 3906, 3910, 3914, 3940, 4417
\l_zrefclever_label_type_b_tl ..
  ..... 3393, 3461,
  3465, 3616, 3644, 3764, 3771, 3795
\__zrefclever_label_type_put_-
  new_right:n .... 83, 85, 3409, 3445
\l_zrefclever_label_types_seq ..
  .... 85, 3402, 3405, 3449, 3452, 3639
\__zrefclever_labels_in_sequence:nn
  ..... 43, 91, 120, 4202, 4404, 5085
\l_zrefclever_lang_decl_case_tl
  403, 731, 734, 777, 782, 1138, 1160,
  2674, 2677, 2710, 2715, 3152, 3169
\l_zrefclever_lang_declension_-
  seq ..... 403,
  610, 611, 612, 626, 630, 637, 728,
  729, 730, 733, 770, 776, 781, 2671,
  2672, 2673, 2676, 2703, 2709, 2714
\l_zrefclever_lang_gender_seq ..
  .... 403, 647, 648, 649, 664, 741,
  742, 791, 808, 2684, 2685, 2724, 2751
\__zrefclever_language_if_-
  declared:n ..... 20, 536
\__zrefclever_language_if_-
  declared:n(TF) ..... 20
\__zrefclever_language_if_-
  declared:nTF 284, 295, 306, 527,
  542, 558, 602, 706, 1937, 1974, 2662
\__zrefclever_language_varname:n
  ..... 19, 20,
  287, 298, 309, 523, 526, 529, 545,
  546, 560, 561, 710, 715, 745, 747, 751
\l_zrefclever_last_of_type_bool
  ..... 90, 3660, 3780,
  3785, 3786, 3790, 3796, 3797, 3920
\l_zrefclever_lastsep_tl . 3685,
  3844, 3990, 4025, 4069, 4106, 4144
\l_zrefclever_link_star_bool ...
  .. 3344, 3383, 4647, 4689, 4792, 5050
\l_zrefclever_listsep_tl .....
  ... 3685, 3840, 4021, 4099, 4470,
  4493, 4505, 4539, 4561, 4570, 4590
\g__zrefclever_loaded_langfiles_-
  seq ..... 701, 709, 744, 750
\__zrefclever_ltxlabel:n .....
  ..... 130, 5437, 5447, 5452
\l_zrefclever_main_language_tl .
  50, 1894, 1900, 1906, 1910, 1932, 1969

```

_zrefclever_mathtools_showonlyrefs:n	_l_zrefclever_nudge_gender_bool
..... 3378, 5509 2002, 2031, 2041, 2046, 4911
_l_zrefclever_mathtools_-	_l_zrefclever_nudge_multitype_-
showonlyrefs_bool 3376, 5501, 5508	bool ... 1999, 2029, 2037, 2044, 4370
_zrefclever_memoir_both_-	_l_zrefclever_nudge_singular_-
labels: .. 5352, 5363, 5365, 5367, 5371, 5375	bool 2001, 2057, 4885
_l_zrefclever_memoir_footnote_-	_zrefclever_opt_bool_get:NN(TF)
type_tl 5382, 5383, 5385, 5389 16
_zrefclever_memoir_label_and_-	_zrefclever_opt_bool_get:NNTF .
zlabel:n 5355, 5360	... 384, 5225, 5230, 5235, 5240, 5245
_zrefclever_memoir_orig_-	_zrefclever_opt_bool_gunset:N .
label:n 5354, 5357 15, 374, 3315, 3323
_zrefclever_name_default:	_zrefclever_opt_bool_if:N 15
..... 4636, 4774	_zrefclever_opt_bool_if:N(TF) . 16
_l_zrefclever_name_format_-	_zrefclever_opt_bool_if:NTF ...
fallback_tl 3672, 4876, 395, 675
4880, 4945, 4994, 5006, 5008, 5026	_zrefclever_opt_bool_if_set:N . 15
_l_zrefclever_name_format_tl ...	_zrefclever_opt_bool_if_-
... 3672, 4862, 4863, 4866, 4867,	set:N(TF) 15
4877, 4878, 4951, 4958, 4959, 4967,	_zrefclever_opt_bool_if_-
4975, 4985, 5002, 5003, 5016, 5036	set:NTF 380,
_l_zrefclever_name_in_link_bool	386, 397, 1263, 1279, 1301, 1317
..... 113,	_zrefclever_opt_bool_unset:N ..
115, 3672, 4292, 4725, 5054, 5070, 5071 15, 374, 2375, 2647
_l_zrefclever_namefont_tl 3685,	_zrefclever_opt_seq_get:NN(TF) 15
3852, 4295, 4312, 4744, 4764, 4779	_zrefclever_opt_seq_get:NNTF ..
_l_zrefclever_nameinlink_str 363, 605, 642, 723, 736, 2666,
..... 1868, 1873, 1875,	2679, 5192, 5197, 5202, 5207, 5212
1877, 1879, 5052, 5058, 5060, 5064	_zrefclever_opt_seq_gset_-
_l_zrefclever_namesep_tl 3685, 3832, 4747, 4767, 4775, 4783	clist_split:Nn 14, 349, 1188, 1224, 3200, 3236
_l_zrefclever_next_is_same_bool	_zrefclever_opt_seq_gunset:N ..
..... 91, 120, 3678, 14, 353, 2738, 3191, 3227
4398, 4431, 4448, 4454, 5109, 5147	_zrefclever_opt_seq_if_set:N .. 14
_l_zrefclever_next_maybe_range_-	_zrefclever_opt_seq_if_-
bool 90, 120, 3678, 4196, 4209, 4397,	set:N(TF) 15
4424, 4437, 5101, 5108, 5127, 5145	_zrefclever_opt_seq_if_set:NTF
_l_zrefclever_noabbrev_first_- 359, 365, 816, 1181, 1216
bool 1810, 1813, 4873	_zrefclever_opt_seq_set_clist_-
_g_zrefclever_nocompat_bool ...	split:Nn 14, 349, 2338, 2599
..... 2227, 2234, 2270	_zrefclever_opt_seq_unset:N ...
_l_zrefclever_nocompat_bool ... 59 14, 353, 2333, 2591
_g_zrefclever_nocompat_modules_-	_zrefclever_opt_tl_cset_-
seq 2228, 2238, 2241, 2262, 2271, 2272	fallback:nn 1341, 1348
_l_zrefclever_nocompat_modules_-	_zrefclever_opt_tl_get:NN(TF) . 14
seq 59	_zrefclever_opt_tl_get:NNTF ...
_l_zrefclever_nudge_comptosing_-	338, 4947, 4962, 4981, 4990, 5011,
bool ... 2000, 2030, 2039, 2045, 4897	5021, 5159, 5164, 5169, 5174, 5179
_l_zrefclever_nudge_enabled_-	_zrefclever_opt_tl_gset_if_-
bool 1998, 2008, 2010,	new:Nn 14,
2014, 2015, 2020, 2021, 4369, 4883	332, 852, 880, 892, 915, 922, 931,
	939, 954, 961, 970, 978, 993, 1000,
	1009, 1017, 1032, 1039, 1048, 1056,
	1085, 1092, 1101, 1109, 1142, 1155

```

\__zrefclever_opt_tl_gunset:N ...
..... 13, 318, 2785, 2819,
2837, 3015, 3021, 3029, 3036, 3143
\__zrefclever_opt_tl_if_set:N ... 13
\__zrefclever_opt_tl_if_set:N(TF)
..... 14
\__zrefclever_opt_tl_if_set:NTF .
..... 324, 334, 340
\__zrefclever_opt_tl_unset:N 13,
318, 1521, 1526, 2285, 2425, 2509, 2514
\__zrefclever_opt_varname_-
  fallback:nn .....
..... 13, 316, 1344, 5180, 5213, 5246
\__zrefclever_opt_varname_-
  general:nn ..... 12,
277, 1464, 1469, 1478, 1484, 1493,
1499, 1508, 1514, 1523, 1528, 1548,
1554, 2286, 2290, 2334, 2345,
2366, 2371, 2376, 5160, 5193, 5226
\__zrefclever_opt_varname_lang_-
  default:nnn ..... 12,
293, 854, 882, 917, 924, 956, 963,
995, 1002, 1034, 1041, 1087, 1094,
1183, 1199, 1265, 1272, 1303, 1310,
2787, 2794, 2821, 2828, 2871, 2877,
2906, 2913, 2943, 2950, 2980, 2987,
3017, 3023, 3063, 3070, 3193, 3211,
3275, 3296, 3317, 5175, 5208, 5241
\__zrefclever_opt_varname_lang_-
  type:nnnn ... 13, 304, 818, 827,
894, 933, 941, 972, 980, 1011, 1019,
1050, 1058, 1103, 1111, 1144, 1157,
1218, 1235, 1281, 1289, 1319, 1327,
2740, 2761, 2839, 2848, 2885, 2892,
2921, 2929, 2958, 2966, 2995, 3003,
3031, 3038, 3079, 3087, 3145, 3156,
3166, 3229, 3247, 3283, 3304, 3325,
4964, 5013, 5023, 5170, 5203, 5236
\__zrefclever_opt_varname_-
  language:nnn . 12, 282, 573, 583,
594, 607, 644, 677, 725, 738, 2668, 2681
\__zrefclever_opt_varname_-
  type:nnn ..... 12, 279, 2427,
2434, 2452, 2457, 2466, 2472, 2481,
2487, 2496, 2502, 2511, 2516, 2536,
2543, 2593, 2607, 2631, 2640, 2649,
4949, 4983, 4992, 5165, 5198, 5231
\__zrefclever_orig_ltxlabel:n ...
..... 5439, 5446, 5451
\__zrefclever_page_format_aux: .
..... 99, 103
\g__zrefclever_page_format_tl ...
..... 7, 98, 104, 107
\l__zrefclever_pairsep_tl .....
..... 3685, 3836, 3959,
3982, 4008, 4060, 4091, 4130, 4213
\__zrefclever_process_language_-
  settings: ..... 53, 54, 600, 3346
\__zrefclever_prop_put_non_-
  empty:Nnn ..... 38, 1360, 2152, 2207
\__zrefclever_provide_langfile:n
..... 17, 25, 26, 81, 702, 1943, 3345
\l__zrefclever_range_beg_is_-
  first_bool .....
... 3678, 3728, 4047, 4083, 4117,
4389, 4426, 4482, 4527, 4549, 4578
\l__zrefclever_range_beg_label_-
  tl ..... 90,
3678, 3721, 4010, 4023, 4062, 4071,
4101, 4132, 4146, 4156, 4382, 4427,
4444, 4495, 4541, 4563, 4592, 4606
\l__zrefclever_range_count_int ..
..... 90,
3678, 3726, 3972, 4036, 4387, 4430,
4441, 4447, 4453, 4461, 4520, 4629
\l__zrefclever_range_end_ref_tl .
... 3678, 3722, 4158, 4164, 4233,
4239, 4383, 4429, 4446, 4608, 4614
\l__zrefclever_range_same_count_-
  int ..... 90,
3678, 3727, 3950, 4001, 4037, 4388,
4432, 4449, 4455, 4500, 4521, 4630
\l__zrefclever_rangeseq_tl .....
..... 3685, 3848,
4161, 4171, 4236, 4246, 4611, 4621
\l__zrefclever_rangetopair_bool .
..... 3685, 3876, 4197
\l__zrefclever_ref_count_int ...
..... 3663, 3725,
3978, 4056, 4126, 4385, 4418, 4467,
4490, 4502, 4536, 4558, 4567, 4587
\l__zrefclever_ref_decl_case_tl .
..... 22, 614, 619, 620, 624, 627,
631, 635, 638, 683, 686, 688, 1984,
1994, 4956, 4960, 4999, 5004, 5009
\__zrefclever_ref_default: .....
.. 4636, 4678, 4684, 4723, 4768, 4835
\l__zrefclever_ref_gender_tl ...
..... 23, 651, 657,
658, 662, 665, 670, 671, 690, 696,
697, 2003, 2067, 4912, 4920, 4926, 4934
\l__zrefclever_ref_language_tl ..
..... 22, 50, 51, 603, 608, 618,
636, 645, 655, 669, 678, 687, 694,
1892, 1914, 1925, 1931, 1936, 1944,
1962, 1968, 1973, 3345, 3363, 3734,
3738, 3742, 3831, 3835, 3839, 3843,
3847, 3851, 3855, 3859, 3863, 3867,

```


3871, 3875, 3879, 3883, 3887, 3891,
 3895, 3899, 3903, 3907, 3911, 3915,
 4916, 4928, 4939, 4965, 5014, 5024
 \l_zrefclever_ref_property_tl ..
 38, 43, 1366,
 1375, 1382, 1384, 1568, 1592, 1636,
 1653, 1665, 1672, 1677, 1710, 1717,
 1722, 1752, 1759, 1764, 3438, 3761,
 3816, 3820, 4642, 4729, 4787, 5091
 \l_zrefclever_ref_propserity_tl 3406
 \l_zrefclever_ref_typeset_font_-
 tl 2071, 2073, 3356
 \l_zrefclever_refbounds_first_-
 pb_seq 3700,
 3888, 3964, 4014, 4086, 4137, 4218
 \l_zrefclever_refbounds_first_-
 rb_seq . 3700, 3892, 4120, 4252, 4582
 \l_zrefclever_refbounds_first_-
 seq 3700, 3880, 4263, 4485, 4531, 4553
 \l_zrefclever_refbounds_first_-
 sg_seq . 3700, 3884, 3942, 3953, 4050
 \l_zrefclever_refbounds_last_-
 pe_seq 3700, 3912,
 3961, 3984, 4011, 4063, 4093, 4215
 \l_zrefclever_refbounds_last_-
 re_seq
 .. 3700, 3916, 4165, 4173, 4240, 4248
 \l_zrefclever_refbounds_last_-
 seq 3700, 3908, 3992, 4027, 4072, 4108
 \l_zrefclever_refbounds_mid_rb_-
 seq ... 3700, 3900, 4133, 4147, 4593
 \l_zrefclever_refbounds_mid_re_-
 seq 3700, 3904, 4615, 4623
 \l_zrefclever_refbounds_mid_seq
 3700, 3896, 4024, 4102,
 4472, 4496, 4508, 4542, 4564, 4572
 \l_zrefclever_reffont_tl
 3685, 3856, 4651, 4668,
 4693, 4709, 4749, 4758, 4796, 4824
 \c_zrefclever_rf_opts_bool_-
 maybe_type_specific_seq
 48, 408, 1254, 2358, 2622, 3264
 \c_zrefclever_rf_opts_seq_-
 refbounds_seq
 408, 1170, 2324, 2582, 3180
 \c_zrefclever_rf_opts_tl_font_-
 seq 408
 \c_zrefclever_rf_opts_tl_maybe_-
 type_specific_seq . 408, 869, 2808
 \c_zrefclever_rf_opts_tl_not_-
 type_specific_seq
 408, 841, 2404, 2774
 \c_zrefclever_rf_opts_tl_-
 reference_seq 408, 2276
 \c_zrefclever_rf_opts_tl_type_-
 names_seq 408, 1125, 3128
 \c_zrefclever_rf_opts_tl_-
 typesetup_seq 408, 2416
 \l_zrefclever_setup_language_tl
 . 403, 547, 574, 584, 595, 721, 773,
 780, 794, 813, 819, 828, 855, 883,
 895, 918, 925, 934, 942, 957, 964,
 973, 981, 996, 1003, 1012, 1020,
 1035, 1042, 1051, 1059, 1088, 1095,
 1104, 1112, 1145, 1158, 1184, 1200,
 1219, 1236, 1266, 1273, 1282, 1290,
 1304, 1311, 1320, 1328, 2665, 2706,
 2713, 2727, 2741, 2756, 2762, 2788,
 2795, 2822, 2829, 2840, 2849, 2872,
 2878, 2886, 2893, 2907, 2914, 2922,
 2930, 2944, 2951, 2959, 2967, 2981,
 2988, 2996, 3004, 3018, 3024, 3032,
 3039, 3064, 3071, 3080, 3088, 3146,
 3157, 3167, 3194, 3212, 3230, 3248,
 3276, 3284, 3297, 3305, 3318, 3326
 \l_zrefclever_setup_type_tl 403,
 722, 764, 765, 797, 820, 829, 848,
 876, 896, 913, 935, 943, 952, 974,
 982, 991, 1013, 1021, 1030, 1052,
 1060, 1083, 1105, 1113, 1132, 1146,
 1159, 1177, 1220, 1237, 1261, 1283,
 1291, 1299, 1321, 1329, 2399, 2401,
 2428, 2435, 2453, 2458, 2467, 2473,
 2482, 2488, 2497, 2503, 2512, 2517,
 2537, 2544, 2594, 2608, 2632, 2641,
 2650, 2664, 2697, 2698, 2730, 2742,
 2763, 2781, 2815, 2841, 2850, 2867,
 2887, 2894, 2902, 2923, 2931, 2939,
 2960, 2968, 2976, 2997, 3005, 3013,
 3033, 3040, 3059, 3081, 3089, 3135,
 3147, 3158, 3168, 3187, 3231, 3249,
 3271, 3285, 3292, 3306, 3313, 3327
 \l_zrefclever_sort_decided_bool
 3399, 3482, 3496, 3506,
 3510, 3522, 3532, 3547, 3562, 3586
 _zrefclever_sort_default:nn ...
 85, 3440, 3456
 _zrefclever_sort_default_-
 different_types:nn
 40, 83, 88, 3467, 3599
 _zrefclever_sort_default_same_-
 type:nn 83, 85, 3466, 3469
 _zrefclever_sort_labels:
 83, 85, 89, 3354, 3403
 _zrefclever_sort_page:nn
 89, 3439, 3651
 \l_zrefclever_sort_prior_a_int .
 3400,

3601, 3607, 3608, 3614, 3624, 3632
 \l_zrefclever_sort_prior_b_int .
 3400,
 3602, 3609, 3610, 3617, 3625, 3633
 \l_zrefclever_tlastsep_tl
 3685, 3743, 4363
 \l_zrefclever_tlistsep_tl
 3685, 3739, 4334
 \l_zrefclever_tpairsep_tl
 3685, 3735, 4356
 \l_zrefclever_type_count_int ...
 . 90, 115, 3663, 3724, 4331, 4333,
 4346, 4371, 4386, 4860, 4872, 5067
 \l_zrefclever_type_first_label_-
 tl 90,
 113, 3666, 3719, 3937, 4188, 4199,
 4203, 4231, 4282, 4300, 4304, 4380,
 4414, 4722, 4728, 4736, 4740, 4753,
 4786, 4803, 4807, 4813, 4828, 4841
 \l_zrefclever_type_first_label_-
 type_tl ... 90, 115, 3666, 3720,
 3939, 4192, 4381, 4416, 4848, 4891,
 4907, 4915, 4927, 4933, 4950, 4966,
 4976, 4984, 4993, 5015, 5025, 5037
 \l_zrefclever_type_first_-
 refbounds_seq
 ... 3700, 3941, 3952, 3963, 4013,
 4049, 4085, 4119, 4136, 4217, 4251,
 4262, 4283, 4484, 4530, 4552, 4581,
 4750, 4751, 4755, 4759, 4798, 4811,
 4816, 4819, 4825, 4826, 4830, 4831
 \l_zrefclever_type_first_-
 refbounds_set_bool 3700,
 3729, 3943, 3954, 3965, 4016, 4052,
 4088, 4122, 4139, 4219, 4253,
 4260, 4390, 4487, 4533, 4555, 4584
 \l_zrefclever_type_name_gender_-
 seq ... 3672, 4917, 4919, 4922, 4937
 \l_zrefclever_type_name_-
 missing_bool
 ... 3672, 4772, 4844, 4851, 4973, 5033
 _zrefclever_type_name_setup: ..
 16, 18, 113, 4269, 4839
 \l_zrefclever_type_name_tl
 113, 115,
 3672, 4307, 4313, 4745, 4765, 4780,
 4782, 4843, 4850, 4954, 4970, 4972,
 4988, 4997, 5019, 5029, 5031, 5051
 \l_zrefclever_typeset_compress_-
 bool 1444, 1447, 4399
 \l_zrefclever_typeset_labels_-
 seq 90, 3660, 3715, 3749, 3751, 3757
 \l_zrefclever_typeset_last_bool
 90, 3660,
 3746, 3747, 3754, 3779, 4343, 5066
 \l_zrefclever_typeset_name_bool
 .. 1393, 1400, 1405, 1410, 4271, 4287
 \l_zrefclever_typeset_queue_-
 curr_tl 90,
 92, 110, 115, 3666, 3718, 3957,
 3980, 3988, 4006, 4019, 4058,
 4067, 4089, 4097, 4104, 4128, 4142,
 4159, 4169, 4186, 4211, 4234, 4244,
 4273, 4280, 4290, 4324, 4340, 4351,
 4357, 4364, 4378, 4379, 4468, 4491,
 4503, 4537, 4559, 4568, 4588, 4609,
 4619, 4865, 4887, 4898, 5061, 5065
 \l_zrefclever_typeset_queue_-
 prev_tl . 90, 3666, 3717, 4335, 4377
 \l_zrefclever_typeset_range_-
 bool ... 1578, 1796, 1799, 3353, 4184
 \l_zrefclever_typeset_ref_bool .
 .. 1392, 1399, 1404, 1409, 4271, 4277
 _zrefclever_typeset_refs:
 90-92, 3357, 3713
 _zrefclever_typeset_refs_last_-
 of_type: 97, 110, 112, 115, 3922, 3927
 _zrefclever_typeset_refs_not_-
 last_of_type:
 91, 97, 110, 120, 3924, 4393
 \l_zrefclever_typeset_sort_bool
 1420, 1423, 3352
 \l_zrefclever_typesort_seq
 . 40, 88, 1429, 1434, 1435, 1441, 3603
 \l_zrefclever_verbose_testing_-
 bool 3712, 4339
 _zrefclever_zcref:nnn
 22, 52, 3338, 3339
 _zrefclever_zcref:nnnn 81, 84, 3339
 \l_zrefclever_zcref_labels_seq .
 84, 85, 3343,
 3374, 3379, 3383, 3408, 3411, 3716
 \l_zrefclever_zcref_note_tl ...
 2103, 2106, 3359, 3366
 \l_zrefclever_zcref_with_check_-
 bool 2110, 2125, 3349, 3370
 _zrefclever_zcsetup:n
 62, 2393, 2394, 5259,
 5283, 5289, 5297, 5319, 5338, 5388,
 5392, 5393, 5402, 5457, 5491, 5544,
 5569, 5581, 5593, 5608, 5628, 5651
 \l_zrefclever_zrefcheck_-
 available_bool
 2109, 2120, 2132, 3348, 3369