

The zref-clever package^{*}

Code documentation

Gustavo Barros[†]

2022-04-22

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	20
4.6	Languages	24
4.7	Language files	29
4.8	Options	42
5	Configuration	66
5.1	\zcsetup	66
5.2	\zcRefTypeSetup	67
5.3	\zcLanguageSetup	72
6	User interface	82
6.1	\zceref	82
6.2	\zcpageref	84
7	Sorting	84
8	Typesetting	91

^{*}This file describes v0.3.0, released 2022-04-22.

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

9	Compatibility	126
9.1	appendix	126
9.2	appendices	127
9.3	memoir	128
9.4	KOMA	131
9.5	amsmath	131
9.6	mathtools	134
9.7	breqn	135
9.8	listings	136
9.9	enumitem	137
9.10	subcaption	137
9.11	subfig	138
10	Language files	138
10.1	English	139
10.2	German	142
10.3	French	151
10.4	Portuguese	155
10.5	Spanish	160
10.6	Dutch	164
	Index	168

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} {2022-04-22} {0.3.0}
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 { ifdraft }

```

3 zref setup

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

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

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

```

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

```

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

```

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

```

```

29     }
30   }
31   \zref@addprop \ZREF@mainlist { thecounter }

```

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

```

32   \zref@newprop { zc@type }
33   {
34     \exp_args:NNe \prop_if_in:NnTF \l__zrefclever_counter_type_prop
35       \l__zrefclever_current_counter_tl
36     {
37       \exp_args:NNe \prop_item:Nn \l__zrefclever_counter_type_prop
38         { \l__zrefclever_current_counter_tl }
39     }
40     { \l__zrefclever_current_counter_tl }
41   }
42   \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`’).

```

43   \zref@newprop { zc@cntval } [0]
44   {
45     \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
46       { \int_use:c { c@ \l__zrefclever_current_counter_tl } }
47       {
48         \cs_if_exist:cT { c@ \@currentcounter }
49         { \int_use:c { c@ \@currentcounter } }
50       }
51   }
52   \zref@addprop \ZREF@mainlist { zc@cntval }
53   \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
54   \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>}

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

```

```

58     {
59       { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
60       \_zrefclever_get_enclosing_counters_value:e
61       { \_zrefclever_counter_reset_by:n {#1} }
62     }
63   }

```

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).

```

64 \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 }

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

```

(End definition for `_zrefclever_counter_reset_by:n`.)

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

```

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

```

```

93     \_zrefclever_get_enclosing_counters_value:e
94     \l\_zrefclever_current_counter_tl
95   }
96   \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

97   \tl_new:N \g__zrefclever_page_format_tl
98   \AddToHook { shipout / before }
99   {
100     \group_begin:
101     \int_set:Nn \c@page { 1 }
102     \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
103     \group_end:
104   }
105   \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
106   \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.

```

107   \prg_new_conditional:Npnn \_zrefclever_if_package_loaded:n #1 { T , F , TF }
108   { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
109   \prg_new_conditional:Npnn \_zrefclever_if_class_loaded:n #1 { T , F , TF }
110   { \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

```
111 \msg_new:nnn { zref-clever } { option-not-type-specific }
112 {
113   Option~'#1'~is-not-type-specific~\msg_line_context:..~
114   Set~it~in~'\iow_char:N\zcLanguageSetup'~before~first~'type'~
115   switch~or~as~package~option.
116 }
117 \msg_new:nnn { zref-clever } { option-only-type-specific }
118 {
119   No~type~specified~for~option~'#1'~\msg_line_context:..~
120   Set~it~after~'type'~switch.
121 }
122 \msg_new:nnn { zref-clever } { key-requires-value }
123 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:.. }
124 \msg_new:nnn { zref-clever } { language-declared }
125 { Language~'#1'~is~already~declared~\msg_line_context:..Nothing~to~do. }
126 \msg_new:nnn { zref-clever } { unknown-language-alias }
127 {
128   Language~'#1'~is~unknown~\msg_line_context:..Can't~alias~to~it..~
129   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
130   '\iow_char:N\zcDeclareLanguageAlias'.
131 }
132 \msg_new:nnn { zref-clever } { unknown-language-setup }
133 {
134   Language~'#1'~is~unknown~\msg_line_context:..Can't~set~it~up..~
135   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
136   '\iow_char:N\zcDeclareLanguageAlias'.
137 }
138 \msg_new:nnn { zref-clever } { unknown-language-opt }
139 {
140   Language~'#1'~is~unknown~\msg_line_context:..~
141   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
142   '\iow_char:N\zcDeclareLanguageAlias'.
143 }
144 \msg_new:nnn { zref-clever } { unknown-language-decl }
145 {
146   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:..~
147   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
148   '\iow_char:N\zcDeclareLanguageAlias'.
149 }
150 \msg_new:nnn { zref-clever } { language-no-decl-ref }
151 {
152   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
153   Nothing~to~do~with~option~'d=#2'.
154 }
155 \msg_new:nnn { zref-clever } { language-no-gender }
156 {
157   Language~'#1'~has~no~declared~gender~\msg_line_context:..~
158   Nothing~to~do~with~option~'#2=#3'.
159 }
160 \msg_new:nnn { zref-clever } { language-no-decl-setup }
161 {
162   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
```



```

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

```

```

217     '\iow_char:N\zcref'.
218   }
219   \msg_new:nnn { zref-clever } { missing-hyperref }
220     { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
221   \msg_new:nnn { zref-clever } { option-preamble-only }
222     { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
223   \msg_new:nnn { zref-clever } { unknown-compat-module }
224     {
225       Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
226       Nothing~to~do.
227     }
228   \msg_new:nnn { zref-clever } { refbounds-must-be-four }
229     {
230       The~value~of~option~'#1'~must~be~a~comma~separated~list~
231       of~four~items.~We~received~'#2'~items~\msg_line_context:.~
232       Option~not~set.
233     }
234   \msg_new:nnn { zref-clever } { missing-zref-check }
235     {
236       Option~'check'~requested~\msg_line_context:.~
237       But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
238     }
239   \msg_new:nnn { zref-clever } { zref-check-too-old }
240     {
241       Option~'check'~requested~\msg_line_context:.~
242       But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
243     }
244   \msg_new:nnn { zref-clever } { missing-type }
245     { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
246   \msg_new:nnn { zref-clever } { missing-property }
247     { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
248   \msg_new:nnn { zref-clever } { missing-name }
249     { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
250   \msg_new:nnn { zref-clever } { single-element-range }
251     { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
252   \msg_new:nnn { zref-clever } { compat-package }
253     { Loaded~support~for~'#1'~package. }
254   \msg_new:nnn { zref-clever } { compat-class }
255     { Loaded~support~for~'#1'~documentclass. }
256   \msg_new:nnn { zref-clever } { option-deprecated }
257     {
258       Option~'#1'~has~been~deprecated~\msg_line_context:.~\iow_newline:
259       Use~'#2'~instead.
260     }
261   \msg_new:nnn { zref-clever } { load-time-options }
262     {
263       'zref-clever'~does~not~accept~load~time~options.~
264       To~configure~package~options,~use~'\iow_char:N\zcsetup'.
265     }

```

4.3 Data extraction

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

```

    \_zrefclever_extract_default:Nnnn {\tl var}}
    {\label}} {\prop}} {\default}}

266 \cs_new_protected:Npn \_zrefclever_extract_default:Nnnn #1#2#3#4
267 {
268     \exp_args:NNo \exp_args:NNo \tl_set:Nn #1
269     { \zref@extractdefault {#2} {#3} {#4} }
270 }
271 \cs_generate_variant:Nn \_zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

(End definition for _zrefclever_extract_default:Nnnn.)

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

```

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

272 \cs_new:Npn \_zrefclever_extract_unexp:nnn #1#2#3
273 {
274     \exp_args:NNo \exp_args:No
275     \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
276 }
277 \cs_generate_variant:Nn \_zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End definition for _zrefclever_extract_unexp:nnn.)

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

```

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

278 \cs_new:Npn \_zrefclever_extract:nnn #1#2#3
279 { \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}
```

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

```
281 { 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}
```

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

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

```
284 \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}
```

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

```
286 {
```

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

```
288   {
```

```
     g\_zrefclever_opt_language_
```

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

```
291     _ #2 _ #3
```

```
292   }
```

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

```
294 }
```

```
295 \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}
```

```

296 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
297 {
298   \__zrefclever_language_if_declared:nTF {#1}
299   {
300     g__zrefclever_opt_lang_
301     \tl_use:c { \__zrefclever_language_varname:n {#1} }
302     _default_ #2 _ #3
303   }
304   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
305 }
306 \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}

307 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
308 {
309   \__zrefclever_language_if_declared:nTF {#1}
310   {
311     g__zrefclever_opt_lang_
312     \tl_use:c { \__zrefclever_language_varname:n {#1} }
313     _type_ #2 _ #3 _ #4
314   }
315   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
316 }
317 \cs_generate_variant:Nn
318 \__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}

319 \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
320 { c__zrefclever_opt_fallback_ #1 _ #2 }

```

(End definition for __zrefclever_opt_varname_fallback:nn.)

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

__zrefclever_opt_var_set_bool:n {\<option var>}

321 \cs_new:Npn __zrefclever_opt_var_set_bool:n #1

322 { \cs_to_str:N #1 _is_set_bool }

(End definition for __zrefclever_opt_var_set_bool:n.)

 __zrefclever_opt_tl_set:N {\<option tl>} {\<value>}

 __zrefclever_opt_tl_clear:N {\<option tl>}

 __zrefclever_opt_tl_gset:N {\<option tl>} {\<value>}

 __zrefclever_opt_tl_gclear:N {\<option tl>}

323 \cs_new_protected:Npn __zrefclever_opt_tl_set:Nn #1#2

324 {

 \tl_if_exist:NF #1

 { \tl_new:N #1 }

 \tl_set:Nn #1 {#2}

 \bool_if_exist:cF { __zrefclever_opt_var_set_bool:n {#1} }

 { \bool_new:c { __zrefclever_opt_var_set_bool:n {#1} } }

 \bool_set_true:c { __zrefclever_opt_var_set_bool:n {#1} }

331 }

332 \cs_generate_variant:Nn __zrefclever_opt_tl_set:Nn { cn }

333 \cs_new_protected:Npn __zrefclever_opt_tl_clear:N #1

334 {

 \tl_if_exist:NF #1

 { \tl_new:N #1 }

 \tl_clear:N #1

 \bool_if_exist:cF { __zrefclever_opt_var_set_bool:n {#1} }

 { \bool_new:c { __zrefclever_opt_var_set_bool:n {#1} } }

 \bool_set_true:c { __zrefclever_opt_var_set_bool:n {#1} }

341 }

342 \cs_generate_variant:Nn __zrefclever_opt_tl_clear:N { c }

343 \cs_new_protected:Npn __zrefclever_opt_tl_gset:Nn #1#2

344 {

 \tl_if_exist:NF #1

 { \tl_new:N #1 }

 \tl_gset:Nn #1 {#2}

348 }

349 \cs_generate_variant:Nn __zrefclever_opt_tl_gset:Nn { cn }

350 \cs_new_protected:Npn __zrefclever_opt_tl_gclear:N #1

351 {

 \tl_if_exist:NF #1

 { \tl_new:N #1 }

 \tl_gclear:N #1

355 }

356 \cs_generate_variant:Nn __zrefclever_opt_tl_gclear:N { c }

(End definition for __zrefclever_opt_tl_set:Nn and others.)

__zrefclever_opt_tl_unset:N Unset {\<option tl>}.

 __zrefclever_opt_tl_unset:N {\<option tl>}

357 \cs_new_protected:Npn __zrefclever_opt_tl_unset:N #1

358 {

 \tl_if_exist:NT #1

359

```

360     {
361       \tl_clear:N #1 % ?
362       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
363       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
364       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
365     }
366   }
367   \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End definition for __zrefclever_opt_tl_unset:N.)

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

```

\__zrefclever_opt_tl_if_set:N(TF) {<option tl>} {<true>} {<false>}
368 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
369 {
370   \tl_if_exist:NTF #1
371   {
372     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
373     {
374       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
375       { \prg_return_true: }
376       { \prg_return_false: }
377     }
378     { \prg_return_true: }
379   }
380   { \prg_return_false: }
381 }

```

(End definition for __zrefclever_opt_tl_if_set:NTF.)

```

\__zrefclever_opt_tl_gset_if_new:Nn \__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\__zrefclever_opt_tl_gclear_if_new:N \__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
382 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
383 {
384   \__zrefclever_opt_tl_if_set:NF #1
385   {
386     \tl_if_exist:NF #1
387     { \tl_new:N #1 }
388     \tl_gset:Nn #1 {#2}
389   }
390 }
391 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
392 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
393 {
394   \__zrefclever_opt_tl_if_set:NF #1
395   {
396     \tl_if_exist:NF #1
397     { \tl_new:N #1 }
398     \tl_gclear:N #1
399   }

```

```

400 }
401 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }

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

```

```

\__zrefclever_opt_tl_get:NNTF \__zrefclever_opt_tl_get:NN(TF) {\<option tl to get>} {\<tl var to set>}
{\<true>} {\<false>}}
402 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
403 {
404   \__zrefclever_opt_tl_if_set:NTF #1
405   {
406     \tl_set_eq:NN #2 #1
407     \prg_return_true:
408   }
409   { \prg_return_false: }
410 }
411 \prg_generate_conditional_variant:Nnn
412 \__zrefclever_opt_tl_get:NN { cN } { F }

(End definition for \__zrefclever_opt_tl_get:NNTF.)

```

```

\__zrefclever_opt_seq_set_clist_split:Nn \__zrefclever_opt_seq_set_clist_split:Nn {\<option seq>} {\<value>}
\__zrefclever_opt_seq_gset_clist_split:Nn \__zrefclever_opt_seq_gset_clist_split:Nn {\<option seq>} {\<value>}
\__zrefclever_opt_seq_set_eq:NN \__zrefclever_opt_seq_set_eq:NN {\<option seq>} {\<seq var>}
\__zrefclever_opt_seq_gset_eq:NN \__zrefclever_opt_seq_gset_eq:NN {\<option seq>} {\<seq var>}
413 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
414 { \seq_set_split:Nnn #1 { , } {#2} }
415 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
416 { \seq_gset_split:Nnn #1 { , } {#2} }
417 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
418 {
419   \seq_if_exist:NF #1
420   { \seq_new:N #1 }
421   \seq_set_eq:NN #1 #2
422   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
423   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
424   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
425 }
426 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
427 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
428 {
429   \seq_if_exist:NF #1
430   { \seq_new:N #1 }
431   \seq_gset_eq:NN #1 #2
432 }
433 \cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }

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

```

__zrefclever_opt_seq_unset:N Unset <option seq>.

```

\__zrefclever_opt_seq_unset:N {\<option seq>}

```



```

434 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
435 {
436   \seq_if_exist:NT #1
437   {
438     \seq_clear:N #1 % ?
439     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
440     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
441     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
442   }
443 }
444 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

(End definition for __zrefclever_opt_seq_unset:N.)

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

```

\__zrefclever_opt_seq_if_set:N(TF) {\<option seq>} {\<true>} {\<false>}
445 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
446 {
447   \seq_if_exist:NTF #1
448   {
449     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
450     {
451       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
452       { \prg_return_true: }
453       { \prg_return_false: }
454     }
455     { \prg_return_true: }
456   }
457   { \prg_return_false: }
458 }
459 \prg_generate_conditional_variant:Nnn
460 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_seq_if_set:NTF.)

```

\__zrefclever_opt_seq_get:NNTF \__zrefclever_opt_seq_get:NN(TF) {\<option seq to get>} {\<seq var to set>}
{\<true>} {\<false>}
461 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
462 {
463   \__zrefclever_opt_seq_if_set:NTF #1
464   {
465     \seq_set_eq:NN #2 #1
466     \prg_return_true:
467   }
468   { \prg_return_false: }
469 }
470 \prg_generate_conditional_variant:Nnn
471 \__zrefclever_opt_seq_get:NN { cN } { F }

```

(End definition for __zrefclever_opt_seq_get:NNTF.)

_zrefclever_opt_bool_unset:N Unset *<option bool>*.

```

\_zrefclever_opt_bool_unset:N {\<option bool>}

```

```

472 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
473 {
474   \bool_if_exist:NT #1
475   {
476     % \bool_set_false:N #1 % ?
477     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
478     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
479     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
480   }
481 }
482 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }

```

(End definition for __zrefclever_opt_bool_unset:N.)

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

```

\__zrefclever_opt_bool_if_set:N(TF) {\<option bool>} {\<true>} {\<false>}
483 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
484 {
485   \bool_if_exist:NTF #1
486   {
487     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
488     {
489       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
490       { \prg_return_true: }
491       { \prg_return_false: }
492     }
493     { \prg_return_true: }
494   }
495   { \prg_return_false: }
496 }
497 \prg_generate_conditional_variant:Nnn
498 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End definition for __zrefclever_opt_bool_if_set:NTF.)

```

\__zrefclever_opt_bool_set_true:N {\<option bool>}
\__zrefclever_opt_bool_set_false:N {\<option bool>}
\__zrefclever_opt_bool_gset_true:N {\<option bool>}
\__zrefclever_opt_bool_gset_false:N {\<option bool>}
499 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
500 {
501   \bool_if_exist:NF #1
502   { \bool_new:N #1 }
503   \bool_set_true:N #1
504   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
505   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
506   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
507 }
508 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
509 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
510 {
511   \bool_if_exist:NF #1
512   { \bool_new:N #1 }

```

```

513     \bool_set_false:N #1
514     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
515       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
516     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
517   }
518 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
519 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
520 {
521   \bool_if_exist:NF #1
522     { \bool_new:N #1 }
523   \bool_gset_true:N #1
524 }
525 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
526 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
527 {
528   \bool_if_exist:NF #1
529     { \bool_new:N #1 }
530   \bool_gset_false:N #1
531 }
532 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }

```

(End definition for __zrefclever_opt_bool_set_true:N and others.)

```

\__zrefclever_opt_bool_get:NNTF      \__zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
                                     {{true}} {{false}}
533 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
534 {
535   \__zrefclever_opt_bool_if_set:NTF #1
536   {
537     \bool_set_eq:NN #2 #1
538     \prg_return_true:
539   }
540   { \prg_return_false: }
541 }
542 \prg_generate_conditional_variant:Nnn
543   \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End definition for __zrefclever_opt_bool_get:NNTF.)

```

\__zrefclever_opt_bool_if:NNTF      \__zrefclever_opt_bool_if:N(TF) {<option bool>} {{true}} {{false}}
544 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
545 {
546   \__zrefclever_opt_bool_if_set:NTF #1
547   { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
548   { \prg_return_false: }
549 }
550 \prg_generate_conditional_variant:Nnn
551   \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

(End definition for __zrefclever_opt_bool_if:NTF.)

4.5 Reference format

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

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

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

<code>\l__zrefclever_setup_type_tl</code>	Store “current” type, language, and declension cases in different places for type-specific and language-specific options handling, notably in <code>__zrefclever_provide_langfile:n</code> , <code>\zcRefTypeSetup</code> , and <code>\zcLanguageSetup</code> , but also for language specific options retrieval.
<code>\l__zrefclever_setup_language_tl</code>	
<code>\l__zrefclever_lang_decl_case_tl</code>	
<code>\l__zrefclever_lang_declension_seq</code>	
<code>\l__zrefclever_lang_gender_seq</code>	

```

552 \tl_new:N \l__zrefclever_setup_type_tl
553 \tl_new:N \l__zrefclever_setup_language_tl
554 \tl_new:N \l__zrefclever_lang_decl_case_tl
555 \seq_new:N \l__zrefclever_lang_declension_seq
556 \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
efclever_rf_opts_tl_maybe_type_specific_seq
\g_zrefclever_rf_opts_seq_refbounds_seq
clever_rf_opts_bool_maybe_type_specific_seq
\g_zrefclever_rf_opts_tl_type_names_seq
\g_zrefclever_rf_opts_tl_typesetup_seq
\g_zrefclever_rf_opts_tl_reference_seq

```

Lists of reference format options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent. These variables are *constants*, but I don’t seem to be able to find a way to concatenate two constants into a third one without triggering L^AT_EX3 debug error “Inconsistent local/global assignment”. And repeating things in a new `\seq_const_from_clist:Nn` defeats the purpose of these variables.

```

557 \seq_new:N \g_zrefclever_rf_opts_tl_not_type_specific_seq
558 \seq_gset_from_clist:Nn
559 \g_zrefclever_rf_opts_tl_not_type_specific_seq
560 {
561     tpairsep ,
562     tlistsep ,
563     tlastsep ,
564     notesep ,
565 }
566 \seq_new:N \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
567 \seq_gset_from_clist:Nn
568 \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
569 {
570     namesep ,
571     pairsep ,
572     listsep ,
573     lastsep ,
574     rangesep ,
575     namefont ,
576     reffont ,
577 }
578 \seq_new:N \g_zrefclever_rf_opts_seq_refbounds_seq
579 \seq_gset_from_clist:Nn
580 \g_zrefclever_rf_opts_seq_refbounds_seq
581 {
582     refbounds-first ,
583     refbounds-first-sg ,
584     refbounds-first-pb ,
585     refbounds-first-rb ,
586     refbounds-mid ,
587     refbounds-mid-rb ,
588     refbounds-mid-re ,
589     refbounds-last ,
590     refbounds-last-pe ,
591     refbounds-last-re ,
592 }
593 \seq_new:N \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
594 \seq_gset_from_clist:Nn
595 \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
596 {
597     cap ,
598     abbrev ,
599     rangetopair ,
600 }

```

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:.

```

601 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
602 \seq_gset_from_clist:Nn
603 \g__zrefclever_rf_opts_tl_type_names_seq
604 {
605   Name-sg ,
606   name-sg ,
607   Name-pl ,
608   name-pl ,
609   Name-sg-ab ,
610   name-sg-ab ,
611   Name-pl-ab ,
612   name-pl-ab ,
613 }

```

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

```

614 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
615 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
616 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
617 \g__zrefclever_rf_opts_tl_type_names_seq
618 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
619 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
620 \g__zrefclever_rf_opts_tl_not_type_specific_seq
621 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

(End definition for \g__zrefclever_rf_opts_tl_not_type_specific_seq and others.)

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

```

622 \clist_map_inline:nn
623 {
624   reference ,
625   typesetup ,
626   langsetup ,
627   langfile ,
628 }
629 {
630   \keys_define:nn { zref-clever/ #1 }
631   {
632     +refbounds-first .meta:n =
633     {
634       refbounds-first = {##1} ,
635       refbounds-first-sg = {##1} ,
636       refbounds-first-pb = {##1} ,
637       refbounds-first-rb = {##1} ,
638     } ,
639     +refbounds-mid .meta:n =
640     {
641       refbounds-mid = {##1} ,
642       refbounds-mid-rb = {##1} ,
643       refbounds-mid-re = {##1} ,
644     } ,
645     +refbounds-last .meta:n =
646     {
647       refbounds-last = {##1} ,

```

```

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

```

```

702     }
703 }

```

4.6 Languages

`\l__zrefclever_current_language_tl` is an internal alias for babel's `\language` or polyglossia's `\mainbabelname` and, if none of them is loaded, we set it to `english`. `\l__zrefclever_main_language_tl` is an internal alias for babel's `\bbl@main@language` or for polyglossia's `\mainbabelname`, as the case may be. Note that for polyglossia we get babel's language names, so that we only need to handle those internally. `\l__zrefclever_ref_language_tl` is the internal variable which stores the language in which the reference is to be made.

```

704 \tl_new:N \l__zrefclever_ref_language_tl
705 \tl_new:N \l__zrefclever_current_language_tl
706 \tl_new:N \l__zrefclever_main_language_tl

```

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

```

707 \tl_new:N \l_zrefclever_ref_language_tl
708 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```

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

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

```

    \_zrefclever_language_varname:n {<language>}

709 \cs_new:Npn \_zrefclever_language_varname:n #1
710 { 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`.

```

711 \cs_set_eq:NN \zrefclever_language_varname:n
712 \_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>}

713 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
714 {
715     \tl_if_exist:cTF { \_zrefclever_language_varname:n {#1} }
716     { \prg_return_true: }
717     { \prg_return_false: }
718 }
719 \prg_generate_conditional_variant:Nnn
720 \_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`.

```

721 \prg_set_eq_conditional:Nn \zrefclever_language_if_declared:n
722 \__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`. $\langle language \rangle$ 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”. [$\langle options \rangle$] receive a `k=v` set of options, with three valid options. The first, `declension`, takes the noun declension cases prefixes for $\langle language \rangle$ as a comma separated list, whose first element is taken to be the default case. The second, `gender`, receives the genders for $\langle language \rangle$ as comma separated list. The third, `allcaps`, is a boolean, and indicates that for $\langle language \rangle$ all nouns must be capitalized for grammatical reasons, in which case, the `cap` option is disregarded for $\langle language \rangle$. If $\langle language \rangle$ is already known, just warn. This implies a particular restriction regarding [$\langle options \rangle$], 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 [ $\langle options \rangle$ ] { $\langle language \rangle$ }

723 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
724 {
725   \group_begin:
726   \tl_if_empty:nF {#2}
727   {
728     \__zrefclever_language_if_declared:nTF {#2}
729     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
730     {
731       \tl_new:c { \__zrefclever_language_varname:n {#2} }
732       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
733       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
734       \keys_set:nn { zref-clever/declarelang } {#1}
735     }
736   }
737   \group_end:
738 }
739 \@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 { $\langle language alias \rangle$ } { $\langle aliased language \rangle$ }

740 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
741 {
742   \tl_if_empty:nF {#1}
743   {

```

```

744     \_zrefclever_language_if_declared:nTF {#2}
745     {
746         \tl_new:c { \_zrefclever_language_varname:n {#1} }
747         \tl_gset:cx { \_zrefclever_language_varname:n {#1} }
748         { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
749     }
750     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
751 }
752 }
753 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

754 \keys_define:nn { zref-clever/declarelang }
755 {
756     declension .code:n =
757     {
758         \seq_new:c
759         {
760             \_zrefclever_opt_varname_language:enn
761             { \l__zrefclever_setup_language_tl } { declension } { seq }
762         }
763         \seq_gset_from_clist:cn
764         {
765             \_zrefclever_opt_varname_language:enn
766             { \l__zrefclever_setup_language_tl } { declension } { seq }
767         }
768         {#1}
769     } ,
770     declension .value_required:n = true ,
771     gender .code:n =
772     {
773         \seq_new:c
774         {
775             \_zrefclever_opt_varname_language:enn
776             { \l__zrefclever_setup_language_tl } { gender } { seq }
777         }
778         \seq_gset_from_clist:cn
779         {
780             \_zrefclever_opt_varname_language:enn
781             { \l__zrefclever_setup_language_tl } { gender } { seq }
782         }
783         {#1}
784     } ,
785     gender .value_required:n = true ,
786     allcaps .choices:nn =
787     { true , false }
788     {
789         \bool_new:c
790         {
791             \_zrefclever_opt_varname_language:enn
792             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
793         }
794         \use:c { bool_gset_ \l_keys_choice_tl :c }
795         {

```

```

796         \_zrefclever_opt_varname_language:enn
797         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
798     }
799 },
800 allcaps .default:n = true ,
801 }

```

`_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.

```

802 \cs_new_protected:Npn \_zrefclever_process_language_settings:
803 {
804     \_zrefclever_language_if_declared:xTF
805     { \l__zrefclever_ref_language_tl }
806     {

```

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.

```

807     \_zrefclever_opt_seq_get:cNF
808     {
809         \_zrefclever_opt_varname_language:enn
810         { \l__zrefclever_ref_language_tl } { declension } { seq }
811     }
812     \l__zrefclever_lang_declension_seq
813     { \seq_clear:N \l__zrefclever_lang_declension_seq }
814     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
815     {
816         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
817         {
818             \msg_warning:nxxx { zref-clever }
819             { language-no-decl-ref }
820             { \l__zrefclever_ref_language_tl }
821             { \l__zrefclever_ref_decl_case_tl }
822             \tl_clear:N \l__zrefclever_ref_decl_case_tl
823         }
824     }
825     {
826         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
827         {
828             \seq_get_left:NN \l__zrefclever_lang_declension_seq
829             \l__zrefclever_ref_decl_case_tl
830         }
831         {
832             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

833         \l__zrefclever_ref_decl_case_tl
834     {
835         \msg_warning:nxxx { zref-clever }
836         { unknown-decl-case }
837         { \l__zrefclever_ref_decl_case_tl }
838         { \l__zrefclever_ref_language_tl }
839         \seq_get_left:NN \l__zrefclever_lang_declension_seq
840         \l__zrefclever_ref_decl_case_tl
841     }
842 }
843 }

```

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.

```

844     \__zrefclever_opt_seq_get:cNF
845     {
846         \__zrefclever_opt_varname_language:enn
847         { \l__zrefclever_ref_language_tl } { gender } { seq }
848     }
849     \l__zrefclever_lang_gender_seq
850     { \seq_clear:N \l__zrefclever_lang_gender_seq }
851     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
852     {
853         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
854         {
855             \msg_warning:nxxxx { zref-clever }
856             { language-no-gender }
857             { \l__zrefclever_ref_language_tl }
858             { g }
859             { \l__zrefclever_ref_gender_tl }
860             \tl_clear:N \l__zrefclever_ref_gender_tl
861         }
862     }
863     {
864         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
865         {
866             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
867             \l__zrefclever_ref_gender_tl
868             {
869                 \msg_warning:nxxx { zref-clever }
870                 { gender-not-declared }
871                 { \l__zrefclever_ref_language_tl }
872                 { \l__zrefclever_ref_gender_tl }
873                 \tl_clear:N \l__zrefclever_ref_gender_tl
874             }
875         }
876     }

```

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

```

877     \__zrefclever_opt_bool_if:cT
878     {
879         \__zrefclever_opt_varname_language:enn
880         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

881     }
882     { \keys_set:nn { zref-clever/reference } { cap = true } }
883   }
884   {

```

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

```

885     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
886     {
887       \msg_warning:nxxx { zref-clever } { unknown-language-decl }
888       { \l__zrefclever_ref_decl_case_tl }
889       { \l__zrefclever_ref_language_tl }
890       \tl_clear:N \l__zrefclever_ref_decl_case_tl
891     }
892     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
893     {
894       \msg_warning:nxxxx { zref-clever }
895       { language-no-gender }
896       { \l__zrefclever_ref_language_tl }
897       { g }
898       { \l__zrefclever_ref_gender_tl }
899       \tl_clear:N \l__zrefclever_ref_gender_tl
900     }
901   }
902 }

```

(End definition for `__zrefclever_process_language_settings:.`)

4.7 Language files

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

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

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

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

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

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

```
903 \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}}

904 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
905 {
906   \group_begin:
907   \@bsphack
908   \__zrefclever_language_if_declared:nT {#1}
909   {
910     \seq_if_in:NxF
911     \g__zrefclever_loaded_langfiles_seq
912     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
913     {
914       \exp_args:Nx \file_get:nnNTF
915       {
916         zref-clever-
917         \tl_use:c { \__zrefclever_language_varname:n {#1} }
918         .lang
919       }
920       { \ExplSyntaxOn }
921       \l_tmpa_tl
922       {
923         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
924         \tl_clear:N \l__zrefclever_setup_type_tl
925         \__zrefclever_opt_seq_get:cNF
926         {
927           \__zrefclever_opt_varname_language:nnn
928           {#1} { declension } { seq }
929         }
930         \l__zrefclever_lang_declension_seq
931         { \seq_clear:N \l__zrefclever_lang_declension_seq }
932         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
```

```

933         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
934       {
935         \seq_get_left:NN \l__zrefclever_lang_declension_seq
936         \l__zrefclever_lang_decl_case_tl
937       }
938     \__zrefclever_opt_seq_get:cNF
939     {
940       \__zrefclever_opt_varname_language:nnn
941       {#1} { gender } { seq }
942     }
943     \l__zrefclever_lang_gender_seq
944     { \seq_clear:N \l__zrefclever_lang_gender_seq }
945     \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
946     \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
947     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
948     \msg_info:nnx { zref-clever } { langfile-loaded }
949     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
950   }
951   {

```

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.

```

952         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
953         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
954       }
955     }
956   }
957   \@esphack
958   \group_end:
959 }
960 \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.

```

961 \keys_define:nn { zref-clever/langfile }
962 {
963   type .code:n =
964   {
965     \tl_if_empty:nTF {#1}
966     { \tl_clear:N \l__zrefclever_setup_type_tl }
967     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
968   } ,
969
970   case .code:n =
971   {
972     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
973     {
974       \msg_info:nnxx { zref-clever } { language-no-decl-setup }
975       { \l__zrefclever_setup_language_tl } {#1}

```

```

976     }
977     {
978         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
979         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
980         {
981             \msg_info:nxxx { zref-clever } { unknown-decl-case }
982             {#1} { \l__zrefclever_setup_language_tl }
983             \seq_get_left:NN \l__zrefclever_lang_declension_seq
984             \l__zrefclever_lang_decl_case_tl
985         }
986     }
987 },
988 case .value_required:n = true ,
989
990 gender .value_required:n = true ,
991 gender .code:n =
992 {
993     \seq_if_empty:NnTF \l__zrefclever_lang_gender_seq
994     {
995         \msg_info:nxxxx { zref-clever } { language-no-gender }
996         { \l__zrefclever_setup_language_tl } { gender } {#1}
997     }
998     {
999         \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
1000         {
1001             \msg_info:nnn { zref-clever }
1002             { option-only-type-specific } { gender }
1003         }
1004         {
1005             \seq_clear:N \l_tmpa_seq
1006             \clist_map_inline:nn {#1}
1007             {
1008                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1009                 { \seq_put_right:Nn \l_tmpa_seq {##1} }
1010                 {
1011                     \msg_info:nxxx { zref-clever }
1012                     { gender-not-declared }
1013                     { \l__zrefclever_setup_language_tl } {##1}
1014                 }
1015             }
1016             \__zrefclever_opt_seq_if_set:cF
1017             {
1018                 \__zrefclever_opt_varname_lang_type:eenn
1019                 { \l__zrefclever_setup_language_tl }
1020                 { \l__zrefclever_setup_type_tl }
1021                 { gender }
1022                 { seq }
1023             }
1024             {
1025                 \seq_new:c
1026                 {
1027                     \__zrefclever_opt_varname_lang_type:eenn
1028                     { \l__zrefclever_setup_language_tl }
1029                     { \l__zrefclever_setup_type_tl }

```



```

1030         { gender }
1031         { seq }
1032     }
1033     \seq_gset_eq:cN
1034     {
1035         \__zrefclever_opt_varname_lang_type:eenn
1036         { \l__zrefclever_setup_language_tl }
1037         { \l__zrefclever_setup_type_tl }
1038         { gender }
1039         { seq }
1040     }
1041     \l_tmpa_seq
1042 }
1043 }
1044 }
1045 },
1046 }
1047 \seq_map_inline:Nn
1048 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1049 {
1050     \keys_define:nn { zref-clever/langfile }
1051     {
1052         #1 .value_required:n = true ,
1053         #1 .code:n =
1054         {
1055             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1056             {
1057                 \__zrefclever_opt_tl_gset_if_new:cn
1058                 {
1059                     \__zrefclever_opt_varname_lang_default:enn
1060                     { \l__zrefclever_setup_language_tl }
1061                     {#1} { tl }
1062                 }
1063                 {##1}
1064             }
1065             {
1066                 \msg_info:nnn { zref-clever }
1067                 { option-not-type-specific } {#1}
1068             }
1069         },
1070     }
1071 }
1072 \seq_map_inline:Nn
1073 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1074 {
1075     \keys_define:nn { zref-clever/langfile }
1076     {
1077         #1 .value_required:n = true ,
1078         #1 .code:n =
1079         {
1080             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1081             {
1082                 \__zrefclever_opt_tl_gset_if_new:cn
1083                 {

```

```

1084         \_zrefclever_opt_varname_lang_default:enn
1085         { \l_zrefclever_setup_language_tl }
1086         {#1} { tl }
1087     }
1088     {##1}
1089 }
1090 {
1091     \_zrefclever_opt_tl_gset_if_new:cn
1092     {
1093         \_zrefclever_opt_varname_lang_type:enn
1094         { \l_zrefclever_setup_language_tl }
1095         { \l_zrefclever_setup_type_tl }
1096         {#1} { tl }
1097     }
1098     {##1}
1099 }
1100 },
1101 }
1102 }
1103 \keys_define:nn { zref-clever/langfile }
1104 {
1105     endrange .value_required:n = true ,
1106     endrange .code:n =
1107     {
1108         \str_case:nnF {#1}
1109         {
1110             { ref }
1111             {
1112                 \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1113                 {
1114                     \_zrefclever_opt_tl_gclear_if_new:c
1115                     {
1116                         \_zrefclever_opt_varname_lang_default:enn
1117                         { \l_zrefclever_setup_language_tl }
1118                         { endrangefunc } { tl }
1119                     }
1120                     \_zrefclever_opt_tl_gclear_if_new:c
1121                     {
1122                         \_zrefclever_opt_varname_lang_default:enn
1123                         { \l_zrefclever_setup_language_tl }
1124                         { endrangeprop } { tl }
1125                     }
1126                 }
1127             }
1128             \_zrefclever_opt_tl_gclear_if_new:c
1129             {
1130                 \_zrefclever_opt_varname_lang_type:enn
1131                 { \l_zrefclever_setup_language_tl }
1132                 { \l_zrefclever_setup_type_tl }
1133                 { endrangefunc } { tl }
1134             }
1135             \_zrefclever_opt_tl_gclear_if_new:c
1136             {
1137                 \_zrefclever_opt_varname_lang_type:enn

```

```

1138         { \l__zrefclever_setup_language_tl }
1139         { \l__zrefclever_setup_type_tl }
1140         { endrangeprop } { tl }
1141     }
1142 }
1143 }
1144
1145 { stripprefix }
1146 {
1147     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1148     {
1149         \__zrefclever_opt_tl_gset_if_new:cn
1150         {
1151             \__zrefclever_opt_varname_lang_default:enn
1152             { \l__zrefclever_setup_language_tl }
1153             { endrangefunc } { tl }
1154         }
1155         { __zrefclever_get_endrange_stripprefix }
1156         \__zrefclever_opt_tl_gclear_if_new:c
1157         {
1158             \__zrefclever_opt_varname_lang_default:enn
1159             { \l__zrefclever_setup_language_tl }
1160             { endrangeprop } { tl }
1161         }
1162     }
1163     {
1164         \__zrefclever_opt_tl_gset_if_new:cn
1165         {
1166             \__zrefclever_opt_varname_lang_type:eenn
1167             { \l__zrefclever_setup_language_tl }
1168             { \l__zrefclever_setup_type_tl }
1169             { endrangefunc } { tl }
1170         }
1171         { __zrefclever_get_endrange_stripprefix }
1172         \__zrefclever_opt_tl_gclear_if_new:c
1173         {
1174             \__zrefclever_opt_varname_lang_type:eenn
1175             { \l__zrefclever_setup_language_tl }
1176             { \l__zrefclever_setup_type_tl }
1177             { endrangeprop } { tl }
1178         }
1179     }
1180 }
1181
1182 { pagecomp }
1183 {
1184     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1185     {
1186         \__zrefclever_opt_tl_gset_if_new:cn
1187         {
1188             \__zrefclever_opt_varname_lang_default:enn
1189             { \l__zrefclever_setup_language_tl }
1190             { endrangefunc } { tl }
1191         }

```

```

1192         { __zrefclever_get_endrange_pagecomp }
1193     \__zrefclever_opt_tl_gclear_if_new:c
1194     {
1195         \__zrefclever_opt_varname_lang_default:enn
1196         { \l__zrefclever_setup_language_tl }
1197         { endrangeprop } { t1 }
1198     }
1199 }
1200 {
1201     \__zrefclever_opt_tl_gset_if_new:cn
1202     {
1203         \__zrefclever_opt_varname_lang_type:een
1204         { \l__zrefclever_setup_language_tl }
1205         { \l__zrefclever_setup_type_tl }
1206         { endrangefunc } { t1 }
1207     }
1208     { __zrefclever_get_endrange_pagecomp }
1209     \__zrefclever_opt_tl_gclear_if_new:c
1210     {
1211         \__zrefclever_opt_varname_lang_type:een
1212         { \l__zrefclever_setup_language_tl }
1213         { \l__zrefclever_setup_type_tl }
1214         { endrangeprop } { t1 }
1215     }
1216 }
1217 }
1218
1219 { pagecomp2 }
1220 {
1221     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1222     {
1223         \__zrefclever_opt_tl_gset_if_new:cn
1224         {
1225             \__zrefclever_opt_varname_lang_default:enn
1226             { \l__zrefclever_setup_language_tl }
1227             { endrangefunc } { t1 }
1228         }
1229         { __zrefclever_get_endrange_pagecomptwo }
1230         \__zrefclever_opt_tl_gclear_if_new:c
1231         {
1232             \__zrefclever_opt_varname_lang_default:enn
1233             { \l__zrefclever_setup_language_tl }
1234             { endrangeprop } { t1 }
1235         }
1236     }
1237     {
1238         \__zrefclever_opt_tl_gset_if_new:cn
1239         {
1240             \__zrefclever_opt_varname_lang_type:een
1241             { \l__zrefclever_setup_language_tl }
1242             { \l__zrefclever_setup_type_tl }
1243             { endrangefunc } { t1 }
1244         }
1245         { __zrefclever_get_endrange_pagecomptwo }

```

```

1246         \_zrefclever_opt_tl_gclear_if_new:c
1247         {
1248             \_zrefclever_opt_varname_lang_type:enn
1249             { \l_zrefclever_setup_language_tl }
1250             { \l_zrefclever_setup_type_tl }
1251             { endrangeprop } { tl }
1252         }
1253     }
1254 }
1255 }
1256 {
1257     \tl_if_empty:nTF {#1}
1258     {
1259         \msg_info:nnn { zref-clever }
1260         { endrange-property-undefined } {#1}
1261     }
1262     {
1263         \zref@ifpropundefined {#1}
1264         {
1265             \msg_info:nnn { zref-clever }
1266             { endrange-property-undefined } {#1}
1267         }
1268         {
1269             \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1270             {
1271                 \_zrefclever_opt_tl_gset_if_new:cn
1272                 {
1273                     \_zrefclever_opt_varname_lang_default:enn
1274                     { \l_zrefclever_setup_language_tl }
1275                     { endrangefunc } { tl }
1276                 }
1277                 { __zrefclever_get_endrange_property }
1278                 \_zrefclever_opt_tl_gset_if_new:cn
1279                 {
1280                     \_zrefclever_opt_varname_lang_default:enn
1281                     { \l_zrefclever_setup_language_tl }
1282                     { endrangeprop } { tl }
1283                 }
1284                 {#1}
1285             }
1286             {
1287                 \_zrefclever_opt_tl_gset_if_new:cn
1288                 {
1289                     \_zrefclever_opt_varname_lang_type:enn
1290                     { \l_zrefclever_setup_language_tl }
1291                     { \l_zrefclever_setup_type_tl }
1292                     { endrangefunc } { tl }
1293                 }
1294                 { __zrefclever_get_endrange_property }
1295                 \_zrefclever_opt_tl_gset_if_new:cn
1296                 {
1297                     \_zrefclever_opt_varname_lang_type:enn
1298                     { \l_zrefclever_setup_language_tl }
1299                     { \l_zrefclever_setup_type_tl }

```

```

1300             { endrangeprop } { t1 }
1301         }
1302         {#1}
1303     }
1304 }
1305 }
1306 }
1307 },
1308 }
1309 \seq_map_inline:Nn
1310 \g__zrefclever_rf_opts_tl_type_names_seq
1311 {
1312     \keys_define:nn { zref-clever/langfile }
1313     {
1314         #1 .value_required:n = true ,
1315         #1 .code:n =
1316         {
1317             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1318             {
1319                 \msg_info:nnn { zref-clever }
1320                 { option-only-type-specific } {#1}
1321             }
1322             {
1323                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1324                 {
1325                     \__zrefclever_opt_tl_gset_if_new:cn
1326                     {
1327                         \__zrefclever_opt_varname_lang_type:een
1328                         { \l__zrefclever_setup_language_tl }
1329                         { \l__zrefclever_setup_type_tl }
1330                         {#1} { t1 }
1331                     }
1332                     {##1}
1333                 }
1334                 {
1335                     \__zrefclever_opt_tl_gset_if_new:cn
1336                     {
1337                         \__zrefclever_opt_varname_lang_type:een
1338                         { \l__zrefclever_setup_language_tl }
1339                         { \l__zrefclever_setup_type_tl }
1340                         { \l__zrefclever_lang_decl_case_tl - #1 } { t1 }
1341                     }
1342                     {##1}
1343                 }
1344             }
1345         },
1346     }
1347 }
1348 \seq_map_inline:Nn
1349 \g__zrefclever_rf_opts_seq_refbounds_seq
1350 {
1351     \keys_define:nn { zref-clever/langfile }
1352     {
1353         #1 .value_required:n = true ,

```

```

1354 #1 .code:n =
1355 {
1356   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1357   {
1358     \__zrefclever_opt_seq_if_set:cF
1359     {
1360       \__zrefclever_opt_varname_lang_default:enn
1361       { \l__zrefclever_setup_language_tl } {#1} { seq }
1362     }
1363     {
1364       \seq_gclear:N \g_tmpa_seq
1365       \__zrefclever_opt_seq_gset_clist_split:Nn
1366       \g_tmpa_seq {##1}
1367       \bool_lazy_or:nnTF
1368       { \tl_if_empty_p:n {##1} }
1369       {
1370         \int_compare_p:nNn
1371         { \seq_count:N \g_tmpa_seq } = { 4 }
1372       }
1373       {
1374         \__zrefclever_opt_seq_gset_eq:cN
1375         {
1376           \__zrefclever_opt_varname_lang_default:enn
1377           { \l__zrefclever_setup_language_tl }
1378           {#1} { seq }
1379         }
1380         \g_tmpa_seq
1381       }
1382       {
1383         \msg_info:nxxx { zref-clever }
1384         { refbounds-must-be-four }
1385         {#1} { \seq_count:N \g_tmpa_seq }
1386       }
1387     }
1388   }
1389   {
1390     \__zrefclever_opt_seq_if_set:cF
1391     {
1392       \__zrefclever_opt_varname_lang_type:eenn
1393       { \l__zrefclever_setup_language_tl }
1394       { \l__zrefclever_setup_type_tl } {#1} { seq }
1395     }
1396     {
1397       \seq_gclear:N \g_tmpa_seq
1398       \__zrefclever_opt_seq_gset_clist_split:Nn
1399       \g_tmpa_seq {##1}
1400       \bool_lazy_or:nnTF
1401       { \tl_if_empty_p:n {##1} }
1402       {
1403         \int_compare_p:nNn
1404         { \seq_count:N \g_tmpa_seq } = { 4 }
1405       }
1406       {
1407         \__zrefclever_opt_seq_gset_eq:cN

```

```

1408         {
1409             \__zrefclever_opt_varname_lang_type:eenn
1410             { \l__zrefclever_setup_language_tl }
1411             { \l__zrefclever_setup_type_tl }
1412             {#1} { seq }
1413         }
1414         \g_tmpa_seq
1415     }
1416     {
1417         \msg_info:nxxx { zref-clever }
1418         { refbounds-must-be-four }
1419         {#1} { \seq_count:N \g_tmpa_seq }
1420     }
1421 }
1422 }
1423 } ,
1424 }
1425 }
1426 \seq_map_inline:Nn
1427 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
1428 {
1429     \keys_define:nn { zref-clever/langfile }
1430     {
1431         #1 .choice: ,
1432         #1 / true .code:n =
1433         {
1434             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1435             {
1436                 \__zrefclever_opt_bool_if_set:cF
1437                 {
1438                     \__zrefclever_opt_varname_lang_default:enn
1439                     { \l__zrefclever_setup_language_tl }
1440                     {#1} { bool }
1441                 }
1442                 {
1443                     \__zrefclever_opt_bool_gset_true:c
1444                     {
1445                         \__zrefclever_opt_varname_lang_default:enn
1446                         { \l__zrefclever_setup_language_tl }
1447                         {#1} { bool }
1448                     }
1449                 }
1450             }
1451         }
1452         \__zrefclever_opt_bool_if_set:cF
1453         {
1454             \__zrefclever_opt_varname_lang_type:eenn
1455             { \l__zrefclever_setup_language_tl }
1456             { \l__zrefclever_setup_type_tl }
1457             {#1} { bool }
1458         }
1459         {
1460             \__zrefclever_opt_bool_gset_true:c
1461             {

```



```

1462         \_zrefclever_opt_varname_lang_type:eenn
1463         { \l__zrefclever_setup_language_tl }
1464         { \l__zrefclever_setup_type_tl }
1465         {#1} { bool }
1466     }
1467 }
1468 }
1469 },
1470 #1 / false .code:n =
1471 {
1472     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1473     {
1474         \_zrefclever_opt_bool_if_set:cF
1475         {
1476             \_zrefclever_opt_varname_lang_default:enn
1477             { \l__zrefclever_setup_language_tl }
1478             {#1} { bool }
1479         }
1480         {
1481             \_zrefclever_opt_bool_gset_false:c
1482             {
1483                 \_zrefclever_opt_varname_lang_default:enn
1484                 { \l__zrefclever_setup_language_tl }
1485                 {#1} { bool }
1486             }
1487         }
1488     }
1489     {
1490         \_zrefclever_opt_bool_if_set:cF
1491         {
1492             \_zrefclever_opt_varname_lang_type:eenn
1493             { \l__zrefclever_setup_language_tl }
1494             { \l__zrefclever_setup_type_tl }
1495             {#1} { bool }
1496         }
1497         {
1498             \_zrefclever_opt_bool_gset_false:c
1499             {
1500                 \_zrefclever_opt_varname_lang_type:eenn
1501                 { \l__zrefclever_setup_language_tl }
1502                 { \l__zrefclever_setup_type_tl }
1503                 {#1} { bool }
1504             }
1505         }
1506     }
1507 },
1508 #1 .default:n = true ,
1509 no #1 .meta:n = { #1 = false } ,
1510 no #1 .value_forbidden:n = true ,
1511 }
1512 }

```

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.

```

1513 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1514 {
1515   \tl_const:cn
1516   { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1517 }
1518 \keyval_parse:nnn
1519 { }
1520 { \__zrefclever_opt_tl_cset_fallback:nn }
1521 {
1522   tpairsep = {,~} ,
1523   tlistsep = {,~} ,
1524   tlastsep = {,~} ,
1525   notesep  = {~} ,
1526   namesep  = {\nobreakspace} ,
1527   pairsep  = {,~} ,
1528   listsep  = {,~} ,
1529   lastsep  = {,~} ,
1530   rangesep = {\textendash} ,
1531 }

```

4.8 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn \langle property list \rangle { \langle key \rangle } { \langle value \rangle }

1532 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1533 {
1534   \tl_if_empty:nTF {#3}
1535   { \prop_remove:Nn #1 {#2} }
1536   { \prop_put:Nnn #1 {#2} {#3} }
1537 }

```

(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`.

```

1538 \tl_new:N \l__zrefclever_ref_property_tl
1539 \keys_define:nn { zref-clever/reference }
1540 {
1541   ref .code:n =
1542   {
1543     \tl_if_empty:nTF {#1}
1544     {
1545       \msg_warning:nnn { zref-clever }
1546       { zref-property-undefined } {#1}
1547       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1548     }
1549     {
1550       \zref@ifpropundefined {#1}
1551       {
1552         \msg_warning:nnn { zref-clever }
1553         { zref-property-undefined } {#1}
1554         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1555       }
1556       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1557     }
1558   } ,
1559   ref .initial:n = default ,
1560   ref .value_required:n = true ,
1561   page .meta:n = { ref = page } ,
1562   page .value_forbidden:n = true ,
1563 }

```

typeset option

```

1564 \bool_new:N \l__zrefclever_typeset_ref_bool
1565 \bool_new:N \l__zrefclever_typeset_name_bool
1566 \keys_define:nn { zref-clever/reference }
1567 {
1568   typeset .choice: ,
1569   typeset / both .code:n =
1570   {
1571     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1572     \bool_set_true:N \l__zrefclever_typeset_name_bool
1573   } ,
1574   typeset / ref .code:n =
1575   {
1576     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1577     \bool_set_false:N \l__zrefclever_typeset_name_bool
1578   } ,
1579   typeset / name .code:n =
1580   {
1581     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1582     \bool_set_true:N \l__zrefclever_typeset_name_bool
1583   } ,
1584   typeset .initial:n = both ,
1585   typeset .value_required:n = true ,
1586
1587   noname .meta:n = { typeset = ref } ,
1588   noname .value_forbidden:n = true ,

```

```

1589     noref .meta:n = { typeset = name } ,
1590     noref .value_forbidden:n = true ,
1591 }

```

sort option

```

1592 \bool_new:N \l__zrefclever_typeset_sort_bool
1593 \keys_define:nn { zref-clever/reference }
1594 {
1595     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1596     sort .initial:n = true ,
1597     sort .default:n = true ,
1598     nosort .meta:n = { sort = false },
1599     nosort .value_forbidden:n = true ,
1600 }

```

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`.

```

1601 \seq_new:N \l__zrefclever_typesort_seq
1602 \keys_define:nn { zref-clever/reference }
1603 {
1604     typesort .code:n =
1605     {
1606         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1607         \seq_reverse:N \l__zrefclever_typesort_seq
1608     } ,
1609     typesort .initial:n =
1610     { part , chapter , section , paragraph },
1611     typesort .value_required:n = true ,
1612     notypesort .code:n =
1613     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1614     notypesort .value_forbidden:n = true ,
1615 }

```

comp option

```

1616 \bool_new:N \l__zrefclever_typeset_compress_bool
1617 \keys_define:nn { zref-clever/reference }
1618 {
1619     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1620     comp .initial:n = true ,
1621     comp .default:n = true ,
1622     nocomp .meta:n = { comp = false },
1623     nocomp .value_forbidden:n = true ,
1624 }

```

endrange option

The working of `endrange` option depends on two underlying option values / variables: `endrangefunc` and `endrangeprop`. `endrangefunc` is the more general one,

and `endrangeprop` is used when the first is set to `_zrefclever_get_endrange_property:VVN`, which is the case when the user is setting `endrange` to an arbitrary `zref` property, instead of one of the `\str_case:nn` matches.

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

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

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

```

1625 \NewHook { zref-clever/endrange-setup }
1626 \keys_define:nn { zref-clever/reference }
1627 {
1628   endrange .code:n =
1629   {
1630     \str_case:nnF {#1}
1631     {
1632       { ref }
1633       {
1634         \_zrefclever_opt_tl_clear:c
1635         {
1636           \_zrefclever_opt_varname_general:nn
1637           { endrangefunc } { tl }
1638         }
1639         \_zrefclever_opt_tl_clear:c
1640         {
1641           \_zrefclever_opt_varname_general:nn
1642           { endrangeprop } { tl }
1643         }
1644       }
1645     }
1646     { stripprefix }
1647     {

```

```

1648 \_zrefclever_opt_tl_set:cn
1649 {
1650   \_zrefclever_opt_varname_general:nn
1651   { endrangefunc } { t1 }
1652 }
1653 { \_zrefclever_get_endrange_stripprefix }
1654 \_zrefclever_opt_tl_clear:c
1655 {
1656   \_zrefclever_opt_varname_general:nn
1657   { endrangeprop } { t1 }
1658 }
1659 }
1660
1661 { pagecomp }
1662 {
1663   \_zrefclever_opt_tl_set:cn
1664   {
1665     \_zrefclever_opt_varname_general:nn
1666     { endrangefunc } { t1 }
1667   }
1668   { \_zrefclever_get_endrange_pagecomp }
1669   \_zrefclever_opt_tl_clear:c
1670   {
1671     \_zrefclever_opt_varname_general:nn
1672     { endrangeprop } { t1 }
1673   }
1674 }
1675
1676 { pagecomp2 }
1677 {
1678   \_zrefclever_opt_tl_set:cn
1679   {
1680     \_zrefclever_opt_varname_general:nn
1681     { endrangefunc } { t1 }
1682   }
1683   { \_zrefclever_get_endrange_pagecomptwo }
1684   \_zrefclever_opt_tl_clear:c
1685   {
1686     \_zrefclever_opt_varname_general:nn
1687     { endrangeprop } { t1 }
1688   }
1689 }
1690
1691 { unset }
1692 {
1693   \_zrefclever_opt_tl_unset:c
1694   {
1695     \_zrefclever_opt_varname_general:nn
1696     { endrangefunc } { t1 }
1697   }
1698   \_zrefclever_opt_tl_unset:c
1699   {
1700     \_zrefclever_opt_varname_general:nn
1701     { endrangeprop } { t1 }

```

```

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

```

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.

```

1750 \bool_if:NTF \l__zrefclever_typeset_range_bool
1751 {
1752   \group_begin:
1753   \bool_set_false:N \l_tmpa_bool
1754   \exp_args:Nxx \tl_if_eq:nnT
1755   {
1756     \__zrefclever_extract_unexp:nnn
1757     {#1} { externaldocument } { }
1758   }
1759   {
1760     \__zrefclever_extract_unexp:nnn
1761     {#2} { externaldocument } { }
1762   }
1763   {
1764     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1765     {
1766       \exp_args:Nxx \tl_if_eq:nnT
1767       {
1768         \__zrefclever_extract_unexp:nnn
1769         {#1} { zc@pgfmt } { }
1770       }
1771       {
1772         \__zrefclever_extract_unexp:nnn
1773         {#2} { zc@pgfmt } { }
1774       }
1775       { \bool_set_true:N \l_tmpa_bool }
1776     }
1777     {
1778       \exp_args:Nxx \tl_if_eq:nnT
1779       {
1780         \__zrefclever_extract_unexp:nnn
1781         {#1} { zc@counter } { }
1782       }
1783       {
1784         \__zrefclever_extract_unexp:nnn
1785         {#2} { zc@counter } { }
1786       }
1787       {
1788         \exp_args:Nxx \tl_if_eq:nnT
1789         {
1790           \__zrefclever_extract_unexp:nnn
1791           {#1} { zc@enclval } { }
1792         }
1793         {
1794           \__zrefclever_extract_unexp:nnn
1795           {#2} { zc@enclval } { }
1796         }
1797         { \bool_set_true:N \l_tmpa_bool }
1798       }
1799     }
1800   }
1801   \bool_if:NTF \l_tmpa_bool
1802   {
1803     \__zrefclever_extract_default:Nnvn \l_tmpb_tl

```



```

1804         {#2} { l__zrefclever_endrangeprop_tl } { }
1805     }
1806     {
1807         \zref@ifrefcontainsprop
1808         {#2} { \l__zrefclever_ref_property_tl }
1809         {
1810             \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1811             {#2} { l__zrefclever_ref_property_tl } { }
1812         }
1813         { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1814     }
1815     \exp_args:NNNV
1816     \group_end:
1817     \tl_set:Nn #3 \l_tmpb_tl
1818 }
1819 {
1820     \__zrefclever_extract_default:Nnvn #3
1821     {#2} { l__zrefclever_endrangeprop_tl } { }
1822 }
1823 }
1824 {
1825     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1826     {
1827         \__zrefclever_extract_default:Nnvn #3
1828         {#2} { l__zrefclever_ref_property_tl } { }
1829     }
1830     { \tl_set:Nn #3 { zc@missingproperty } }
1831 }
1832 }
1833 }
1834 \cs_generate_variant:Nn \__zrefclever_get_endrange_property:nnN { VVN }

```

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

```

1835 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1836 {
1837     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1838     {
1839         \group_begin:
1840         \UseHook { zref-clever/endrange-setup }
1841         \tl_set:Nx \l_tmpa_tl
1842         {
1843             \__zrefclever_extract:nnn
1844             {#1} { \l__zrefclever_ref_property_tl } { }
1845         }
1846         \tl_set:Nx \l_tmpb_tl
1847         {
1848             \__zrefclever_extract:nnn
1849             {#2} { \l__zrefclever_ref_property_tl } { }
1850         }
1851         \bool_set_false:N \l_tmpa_bool
1852         \bool_until_do:Nn \l_tmpa_bool
1853         {
1854             \exp_args:Nxx \tl_if_eq:nnTF

```

```

1855         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1856         {
1857             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1858             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1859             \tl_if_empty:NT \l_tmpb_tl
1860                 { \bool_set_true:N \l_tmpa_bool }
1861         }
1862         { \bool_set_true:N \l_tmpa_bool }
1863     }
1864     \exp_args:NNNV
1865     \group_end:
1866     \tl_set:Nn #3 \l_tmpb_tl
1867 }
1868 { \tl_set:Nn #3 { zc@missingproperty } }
1869 }
1870 \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>).

```

1871 \prg_new_protected_conditional:Npnn
1872   \__zrefclever_is_integer_rgx:n #1 { F , TF }
1873 {
1874     \regex_match:nnTF { \A\d+\Z } {#1}
1875     { \prg_return_true: }
1876     { \prg_return_false: }
1877 }
1878 \prg_generate_conditional_variant:Nnn
1879   \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End definition for `__zrefclever_is_integer_rgx:n`.)

```

1880 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1881 {
1882     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1883     {
1884         \group_begin:
1885         \UseHook { zref-clever/endrange-setup }
1886         \tl_set:Nx \l_tmpa_tl
1887         {
1888             \__zrefclever_extract:nnn
1889             {#1} { \l__zrefclever_ref_property_tl } { }
1890         }
1891         \tl_set:Nx \l_tmpb_tl
1892         {
1893             \__zrefclever_extract:nnn
1894             {#2} { \l__zrefclever_ref_property_tl } { }
1895         }
1896         \bool_set_false:N \l_tmpa_bool
1897         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1898         {
1899             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1900             { \bool_set_true:N \l_tmpa_bool }
1901         }
1902         { \bool_set_true:N \l_tmpa_bool }
1903         \bool_until_do:Nn \l_tmpa_bool

```

```

1904     {
1905         \exp_args:Nxx \tl_if_eq:nnTF
1906         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1907         {
1908             \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1909             \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1910             \tl_if_empty:NT \l_tmpb_tl
1911             { \bool_set_true:N \l_tmpa_bool }
1912         }
1913         { \bool_set_true:N \l_tmpa_bool }
1914     }
1915     \exp_args:NNNV
1916     \group_end:
1917     \tl_set:Nn #3 \l_tmpb_tl
1918 }
1919 { \tl_set:Nn #3 { zc@missingproperty } }
1920 }
1921 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1922 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1923 {
1924     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1925     {
1926         \group_begin:
1927         \UseHook { zref-clever/endrange-setup }
1928         \tl_set:Nx \l_tmpa_tl
1929         {
1930             \__zrefclever_extract:nnn
1931             {#1} { \l__zrefclever_ref_property_tl } { }
1932         }
1933         \tl_set:Nx \l_tmpb_tl
1934         {
1935             \__zrefclever_extract:nnn
1936             {#2} { \l__zrefclever_ref_property_tl } { }
1937         }
1938         \bool_set_false:N \l_tmpa_bool
1939         \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1940         {
1941             \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1942             { \bool_set_true:N \l_tmpa_bool }
1943         }
1944         { \bool_set_true:N \l_tmpa_bool }
1945         \bool_until_do:Nn \l_tmpa_bool
1946         {
1947             \exp_args:Nxx \tl_if_eq:nnTF
1948             { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1949             {
1950                 \bool_lazy_or:nnTF
1951                 { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1952                 { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1953                 {
1954                     \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1955                     \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1956                 }
1957                 { \bool_set_true:N \l_tmpa_bool }

```

```

1958         }
1959         { \bool_set_true:N \l_tmpa_bool }
1960     }
1961     \exp_args:NNNV
1962     \group_end:
1963     \tl_set:Nn #3 \l_tmpb_tl
1964 }
1965 { \tl_set:Nn #3 { zc@missingproperty } }
1966 }
1967 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomptwo:nnN { VVN }

```

range and rangetopair options

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

```

1968 \bool_new:N \l__zrefclever_typeset_range_bool
1969 \keys_define:nn { zref-clever/reference }
1970 {
1971     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1972     range .initial:n = false ,
1973     range .default:n = true ,
1974 }

```

cap and capfirst options

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

```

1975 \bool_new:N \l__zrefclever_capfirst_bool
1976 \keys_define:nn { zref-clever/reference }
1977 {
1978     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1979     capfirst .initial:n = false ,
1980     capfirst .default:n = true ,
1981 }

```

abbrev and noabbrevfirst options

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

```

1982 \bool_new:N \l__zrefclever_noabbrev_first_bool
1983 \keys_define:nn { zref-clever/reference }
1984 {
1985     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1986     noabbrevfirst .initial:n = false ,
1987     noabbrevfirst .default:n = true ,
1988 }

```

S option

```

1989 \keys_define:nn { zref-clever/reference }
1990 {
1991     S .meta:n =

```

```

1992     { capfirst = {#1} , noabbrevfirst = {#1} },
1993     S .default:n = true ,
1994 }

```

hyperref option

```

1995 \bool_new:N \l__zrefclever_hyperlink_bool
1996 \bool_new:N \l__zrefclever_hyperref_warn_bool
1997 \keys_define:nn { zref-clever/reference }
1998 {
1999     hyperref .choice: ,
2000     hyperref / auto .code:n =
2001     {
2002         \bool_set_true:N \l__zrefclever_hyperlink_bool
2003         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2004     } ,
2005     hyperref / true .code:n =
2006     {
2007         \bool_set_true:N \l__zrefclever_hyperlink_bool
2008         \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2009     } ,
2010     hyperref / false .code:n =
2011     {
2012         \bool_set_false:N \l__zrefclever_hyperlink_bool
2013         \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2014     } ,
2015     hyperref .initial:n = auto ,
2016     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.

```

2017     nohyperref .meta:n = { hyperref = false } ,
2018     nohyperref .value_forbidden:n = true ,
2019 }
2020 \AddToHook { begindocument }
2021 {
2022     \__zrefclever_if_package_loaded:nTF { hyperref }
2023     {
2024         \bool_if:NT \l__zrefclever_hyperlink_bool
2025         { \RequirePackage { zref-hyperref } }
2026     }
2027     {
2028         \bool_if:NT \l__zrefclever_hyperref_warn_bool
2029         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2030         \bool_set_false:N \l__zrefclever_hyperlink_bool
2031     }
2032     \keys_define:nn { zref-clever/reference }
2033     {
2034         hyperref .code:n =
2035         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2036         nohyperref .code:n =
2037         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,

```

```

2038     }
2039 }

```

nameinlink option

```

2040 \str_new:N \l__zrefclever_nameinlink_str
2041 \keys_define:nn { zref-clever/reference }
2042 {
2043   nameinlink .choice: ,
2044   nameinlink / true .code:n =
2045     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2046   nameinlink / false .code:n =
2047     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2048   nameinlink / single .code:n =
2049     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2050   nameinlink / tsingle .code:n =
2051     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2052   nameinlink .initial:n = tsingle ,
2053   nameinlink .default:n = true ,
2054 }

```

preposinlink option (deprecated)

```

2055 \keys_define:nn { zref-clever/reference }
2056 {
2057   preposinlink .code:n =
2058     {
2059       % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2060       \msg_warning:nnnn { zref-clever }{ option-deprecated }
2061       { preposinlink } { refbounds }
2062     } ,
2063 }

```

lang option

The overall setup here seems a little roundabout, but this is actually required. In the preamble, we (potentially) don’t yet have values for the “current” and “main” document languages, this must be retrieved at a `begindocument` hook. The `begindocument` hook is responsible to get values for `\l__zrefclever_current_language_tl` and `\l__zrefclever_main_language_tl`, and to set the default for `\l__zrefclever_ref_language_tl`. Package options, or preamble calls to `\zcsetup` are also hooked at `begindocument`, but come after the first hook, so that the pertinent variables have been set when they are executed. Finally, we set a third `begindocument` hook, at `begindocument/before`, so that it runs after any options set in the preamble. This hook redefines the `lang` option for immediate execution in the document body, and ensures the `current` language’s language file gets loaded, if it hadn’t been already.

For the `babel` and `polyglossia` variables which store the “current” and “main” languages, see <https://tex.stackexchange.com/a/233178>, including comments, particularly the one by Javier Bezos. For the `babel` and `polyglossia` variables which store the list of loaded languages, see <https://tex.stackexchange.com/a/281220>, including comments, particularly PLK’s. Note, however, that languages loaded by `\babelprovide`, either directly, “on the fly”, or with the `provide` option, do not get included in `\bbl@loaded`.

```

2064 \AddToHook { begindocument }
2065 {

```

```

2066 \__zrefclever_if_package_loaded:nTF { babel }
2067 {
2068   \tl_set:Nn \l__zrefclever_current_language_tl { \languagename }
2069   \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2070 }
2071 {
2072   \__zrefclever_if_package_loaded:nTF { polyglossia }
2073   {
2074     \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2075     \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2076   }
2077   {
2078     \tl_set:Nn \l__zrefclever_current_language_tl { english }
2079     \tl_set:Nn \l__zrefclever_main_language_tl { english }
2080   }
2081 }
2082 }
2083 \keys_define:nn { zref-clever/reference }
2084 {
2085   lang .code:n =
2086   {
2087     \AddToHook { begindocument }
2088     {
2089       \str_case:nnF {#1}
2090       {
2091         { current }
2092         {
2093           \tl_set:Nn \l__zrefclever_ref_language_tl
2094             { \l__zrefclever_current_language_tl }
2095         }
2096
2097         { main }
2098         {
2099           \tl_set:Nn \l__zrefclever_ref_language_tl
2100             { \l__zrefclever_main_language_tl }
2101         }
2102       }
2103       {
2104         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2105         \__zrefclever_language_if_declared:nF {#1}
2106         {
2107           \msg_warning:nnn { zref-clever }
2108             { unknown-language-opt } {#1}
2109         }
2110       }
2111       \__zrefclever_provide_langfile:x
2112         { \l__zrefclever_ref_language_tl }
2113     }
2114   } ,
2115   lang .initial:n = current ,
2116   lang .value_required:n = true ,
2117 }
2118 \AddToHook { begindocument / before }

```

```

2119 {
2120   \AddToHook { begindocument }
2121   {

```

Redefinition of the `lang` key option for the document body. Also, drop the language file loading in the document body, it is somewhat redundant, since `__zrefclever-zcref:nnn` already ensures it.

```

2122     \keys_define:nn { zref-clever/reference }
2123     {
2124       lang .code:n =
2125       {
2126         \str_case:nnF {#1}
2127         {
2128           { current }
2129           {
2130             \tl_set:Nn \l__zrefclever_ref_language_tl
2131             { \l__zrefclever_current_language_tl }
2132           }
2133
2134           { main }
2135           {
2136             \tl_set:Nn \l__zrefclever_ref_language_tl
2137             { \l__zrefclever_main_language_tl }
2138           }
2139         }
2140         {
2141           \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2142           \__zrefclever_language_if_declared:nF {#1}
2143           {
2144             \msg_warning:nnn { zref-clever }
2145             { unknown-language-opt } {#1}
2146           }
2147         }
2148       } ,
2149     }
2150   }
2151 }

```

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.

```

2152 \tl_new:N \l__zrefclever_ref_decl_case_tl
2153 \keys_define:nn { zref-clever/reference }
2154 {
2155   d .code:n =
2156   { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2157 }
2158 \AddToHook { begindocument }

```



```

2159 {
2160   \keys_define:nn { zref-clever/reference }
2161   {

```

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

```

2162     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2163     d .value_required:n = true ,
2164   }
2165 }

```

nudge & co. options

```

2166 \bool_new:N \l__zrefclever_nudge_enabled_bool
2167 \bool_new:N \l__zrefclever_nudge_multitype_bool
2168 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2169 \bool_new:N \l__zrefclever_nudge_singular_bool
2170 \bool_new:N \l__zrefclever_nudge_gender_bool
2171 \tl_new:N \l__zrefclever_ref_gender_tl
2172 \keys_define:nn { zref-clever/reference }
2173 {
2174   nudge .choice: ,
2175   nudge / true .code:n =
2176   { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2177   nudge / false .code:n =
2178   { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2179   nudge / ifdraft .code:n =
2180   {
2181     \ifdraft
2182       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2183       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2184     } ,
2185   nudge / iffina .code:n =
2186   {
2187     \ifoptionfinal
2188       { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2189       { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2190     } ,
2191   nudge .initial:n = false ,
2192   nudge .default:n = true ,
2193   nonnudge .meta:n = { nudge = false } ,
2194   nonnudge .value_forbidden:n = true ,
2195   nudgeif .code:n =
2196   {
2197     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2198     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2199     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2200     \clist_map_inline:nn {#1}
2201     {
2202       \str_case:nnF {##1}
2203       {
2204         { multitype }
2205         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2206         { comptosing }

```

```

2207         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2208         { gender }
2209         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2210         { all }
2211         {
2212             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2213             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2214             \bool_set_true:N \l__zrefclever_nudge_gender_bool
2215         }
2216     }
2217     {
2218         \msg_warning:nnn { zref-clever }
2219         { nudgeif-unknown-value } {##1}
2220     }
2221 }
2222 },
2223 nudgeif .value_required:n = true ,
2224 nudgeif .initial:n = all ,
2225 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2226 sg .initial:n = false ,
2227 sg .default:n = true ,
2228 g .code:n =
2229 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2230 }
2231 \AddToHook { begindocument }
2232 {
2233     \keys_define:nn { zref-clever/reference }
2234     {

```

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

```

2235         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2236         g .value_required:n = true ,
2237     }
2238 }

```

font option

```

2239 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2240 \keys_define:nn { zref-clever/reference }
2241 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2242 \keys_define:nn { zref-clever/reference }
2243 {
2244     titleref .code:n =
2245     {
2246         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2247         \msg_warning:nxxx { zref-clever } { option-deprecated } { titleref }
2248         { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2249     } ,
2250 }

```

vario option

```

2251 \keys_define:nn { zref-clever/reference }

```

```

2252 {
2253     vario .code:n =
2254     {
2255         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2256         \msg_warning:nnxx { zref-clever }{ option-deprecated } { vario }
2257         { \iow_char:N\usepackage\iow_char:N{zref-vario\iow_char:N\} }
2258     } ,
2259 }

```

note option

```

2260 \tl_new:N \l__zrefclever_zcref_note_tl
2261 \keys_define:nn { zref-clever/reference }
2262 {
2263     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2264     note .value_required:n = true ,
2265 }

```

check option

Integration with zref-check.

```

2266 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2267 \bool_new:N \l__zrefclever_zcref_with_check_bool
2268 \keys_define:nn { zref-clever/reference }
2269 {
2270     check .code:n =
2271     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2272 }
2273 \AddToHook { begindocument }
2274 {
2275     \__zrefclever_if_package_loaded:nTF { zref-check }
2276     {
2277         \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2278         {
2279             \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2280             \keys_define:nn { zref-clever/reference }
2281             {
2282                 check .code:n =
2283                 {
2284                     \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2285                     \keys_set:nn { zref-check / zcheck } {#1}
2286                 } ,
2287                 check .value_required:n = true ,
2288             }
2289         }
2290         {
2291             \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2292             \keys_define:nn { zref-clever/reference }
2293             {
2294                 check .code:n =
2295                 {
2296                     \msg_warning:nnn { zref-clever }
2297                     { zref-check-too-old } { 2021-09-16-v0.2.1 }
2298                 } ,
2299             }

```

```

2300     }
2301   }
2302   {
2303     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2304     \keys_define:nn { zref-clever/reference }
2305     {
2306       check .code:n =
2307       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2308     }
2309   }
2310 }

```

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`.

```

2311 \prop_new:N \l__zrefclever_counter_type_prop
2312 \keys_define:nn { zref-clever/label }
2313 {
2314   countertype .code:n =
2315   {
2316     \keyval_parse:nnn
2317     {
2318       \msg_warning:nnnn { zref-clever }
2319       { key-requires-value } { countertype }
2320     }
2321     {
2322       \__zrefclever_prop_put_non_empty:Nnn
2323       \l__zrefclever_counter_type_prop
2324     }
2325     {#1}
2326   } ,
2327   countertype .value_required:n = true ,
2328   countertype .initial:n =
2329   {
2330     subsection    = section ,
2331     subsubsection = section ,
2332     subparagraph  = paragraph ,
2333     enumi         = item ,
2334     enumii        = item ,
2335     enumiii       = item ,
2336     enumiv        = item ,
2337     mpfootnote    = footnote ,
2338   } ,
2339 }

```

One interesting comment I received (by Denis Bitouzé, at issue [#1](#)) about the most appropriate type for `paragraph` and `subparagraph` counters was that the reader of the document does not care whether that particular document structure element has been introduced by `\paragraph` or, e.g. by the `\subsubsection` command. This is a difference the author knows, as they’re using L^AT_EX, but to the reader the difference between them

is not really relevant, and it may be just confusing to refer to them by different names. In this case the type for `paragraph` and `subparagraph` should just be `section`. I don't have a strong opinion about this, and the matter was not pursued further. Besides, I presume not many people would set `secnumdepth` so high to start with. But, for the time being, I left the `paragraph` type for them, since there is actually a visual difference to the reader between the `\subsubsection` and `\paragraph` in the standard classes: up to the former, the sectioning commands break a line before the following text, while, from the later on, the sectioning commands and the following text are part of the same line. So, `\paragraph` is actually different from "just a shorter way to write `\subsubsection`".

counterresetters option

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

```

2340 \seq_new:N \l__zrefclever_counter_resetters_seq
2341 \keys_define:nn { zref-clever/label }
2342 {
2343   counterresetters .code:n =
2344   {
2345     \clist_map_inline:nn {#1}
2346     {
2347       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
2348       {
2349         \seq_put_right:Nn
2350         \l__zrefclever_counter_resetters_seq {##1}
2351       }
2352     }
2353   } ,
2354   counterresetters .initial:n =
2355   {
2356     part ,
2357     chapter ,
2358     section ,
2359     subsection ,
2360     subsubsection ,
2361     paragraph ,
2362     subparagraph ,
2363   } ,
2364   counterresetters .value_required:n = true ,
2365 }
```

counterresetby option

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

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

```

2366 \prop_new:N \l__zrefclever_counter_resetby_prop
2367 \keys_define:nn { zref-clever/label }
2368 {
2369     counterresetby .code:n =
2370     {
2371         \keyval_parse:nnn
2372         {
2373             \msg_warning:nnn { zref-clever }
2374             { key-requires-value } { counterresetby }
2375         }
2376         {
2377             \__zrefclever_prop_put_non_empty:Nnn
2378             \l__zrefclever_counter_resetby_prop
2379         }
2380         {#1}
2381     } ,
2382     counterresetby .value_required:n = true ,
2383     counterresetby .initial:n =
2384     {

```

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.

```

2385         enumii = enumi ,
2386         enumiii = enumii ,
2387         enumiv = enumiii ,
2388     } ,
2389 }

```

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.

```

2390 \tl_new:N \l__zrefclever_current_counter_tl
2391 \keys_define:nn { zref-clever/label }
2392 {
2393     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2394     currentcounter .value_required:n = true ,
2395     currentcounter .initial:n = \@currentcounter ,
2396 }

```

nocompat option

```

2397 \bool_new:N \g__zrefclever_nocompat_bool
2398 \seq_new:N \g__zrefclever_nocompat_modules_seq
2399 \keys_define:nn { zref-clever/reference }
2400 {
2401     nocompat .code:n =

```

```

2402 {
2403   \tl_if_empty:nTF {#1}
2404     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2405     {
2406       \clist_map_inline:nn {#1}
2407       {
2408         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2409         {
2410           \seq_gput_right:Nn
2411             \g__zrefclever_nocompat_modules_seq {##1}
2412         }
2413       }
2414     }
2415   } ,
2416 }
2417 \AddToHook { begindocument }
2418 {
2419   \keys_define:nn { zref-clever/reference }
2420   {
2421     nocompat .code:n =
2422     {
2423       \msg_warning:nnn { zref-clever }
2424       { option-preamble-only } { nocompat }
2425     }
2426   }
2427 }
2428 \AtEndOfPackage
2429 {
2430   \AddToHook { begindocument }
2431   {
2432     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2433     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2434   }
2435 }

```

`_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`. 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>}

2436 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2437 {
2438   \AddToHook { begindocument }
2439   {
2440     \bool_if:NF \g__zrefclever_nocompat_bool
2441     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }

```

```

2442     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2443   }
2444 }

```

(End definition for `__zrefclever_compat_module:nn`.)

Reference options

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

```

2445 \seq_map_inline:Nn
2446   \g__zrefclever_rf_opts_tl_reference_seq
2447 {
2448   \keys_define:nn { zref-clever/reference }
2449   {
2450     #1 .default:o = \c_novalue_tl ,
2451     #1 .code:n =
2452     {
2453       \tl_if_novalue:nTF {##1}
2454       {
2455         \__zrefclever_opt_tl_unset:c
2456         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2457       }
2458       {
2459         \__zrefclever_opt_tl_set:cn
2460         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2461         {##1}
2462       }
2463     } ,
2464   }
2465 }
2466 \keys_define:nn { zref-clever/reference }
2467 {
2468   refpre .code:n =
2469   {
2470     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2471     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2472     { refpre } { refbounds }
2473   } ,
2474   refpos .code:n =
2475   {
2476     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2477     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2478     { refpos } { refbounds }
2479   } ,
2480   preref .code:n =
2481   {
2482     % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2483     \msg_warning:nnnn { zref-clever }{ option-deprecated }
2484     { preref } { refbounds }
2485   } ,
2486   postref .code:n =

```



```

2487     {
2488         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2489         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2490         { postref } { refbounds }
2491     } ,
2492 }
2493 \seq_map_inline:Nn
2494   \g__zrefclever_rf_opts_seq_refbounds_seq
2495   {
2496     \keys_define:nn { zref-clever/reference }
2497     {
2498       #1 .default:o = \c_novalue_tl ,
2499       #1 .code:n =
2500       {
2501         \tl_if_novalue:nTF {##1}
2502         {
2503           \__zrefclever_opt_seq_unset:c
2504           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2505         }
2506         {
2507           \seq_clear:N \l_tmpa_seq
2508           \__zrefclever_opt_seq_set_clist_split:Nn
2509           \l_tmpa_seq {##1}
2510           \bool_lazy_or:nnTF
2511           { \tl_if_empty_p:n {##1} }
2512           { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2513           {
2514             \__zrefclever_opt_seq_set_eq:cN
2515             { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2516             \l_tmpa_seq
2517           }
2518           {
2519             \msg_warning:nxxx { zref-clever }
2520             { refbounds-must-be-four }
2521             {#1} { \seq_count:N \l_tmpa_seq }
2522           }
2523         }
2524       } ,
2525     }
2526   }
2527 \seq_map_inline:Nn
2528   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2529   {
2530     \keys_define:nn { zref-clever/reference }
2531     {
2532       #1 .choice: ,
2533       #1 / true .code:n =
2534       {
2535         \__zrefclever_opt_bool_set_true:c
2536         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2537       } ,
2538       #1 / false .code:n =
2539       {
2540         \__zrefclever_opt_bool_set_false:c

```

```

2541         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2542     } ,
2543     #1 / unset .code:n =
2544     {
2545         \_zrefclever_opt_bool_unset:c
2546         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2547     } ,
2548     #1 .default:n = true ,
2549     no #1 .meta:n = { #1 = false } ,
2550     no #1 .value_forbidden:n = true ,
2551 }
2552 }

```

Package options

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

```

2553 \keys_define:nn { }
2554 {
2555     zref-clever/zcsetup .inherit:n =
2556     {
2557         zref-clever/label ,
2558         zref-clever/reference ,
2559     }
2560 }

```

`zref-clever` does not accept load-time options. Despite the tradition of so doing, Joseph Wright has a point in recommending otherwise at <https://chat.stackexchange.com/transcript/message/60360822#60360822>: separating “loading the package” from “configuring the package” grants less trouble with “option clashes” and with expansion of options at load-time.

```

2561 \bool_lazy_and:nnT
2562 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2563 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2564 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

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

```

2565 \NewDocumentCommand \zcsetup { m }
2566 { \_zrefclever_zcsetup:n {#1} }

```

(End definition for `\zcsetup`.)

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

```

    \__zrefclever_zcsetup:n{<options>}

2567 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2568 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2569 \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>}

2570 \NewDocumentCommand \zcRefTypeSetup { m m }
2571 {
2572   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2573   \keys_set:nn { zref-clever/typesetup } {#2}
2574   \tl_clear:N \l__zrefclever_setup_type_tl
2575 }

(End definition for \zcRefTypeSetup.)

2576 \seq_map_inline:Nn
2577 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2578 {
2579   \keys_define:nn { zref-clever/typesetup }
2580   {
2581     #1 .code:n =
2582     {
2583       \msg_warning:nnn { zref-clever }
2584       { option-not-type-specific } {#1}
2585     } ,
2586   }
2587 }
2588 \seq_map_inline:Nn
2589 \g__zrefclever_rf_opts_tl_typesetup_seq
2590 {
2591   \keys_define:nn { zref-clever/typesetup }
2592   {
2593     #1 .default:o = \c_novalue_tl ,
2594     #1 .code:n =
2595     {
2596       \tl_if_novalue:nTF {##1}
2597       {
2598         \__zrefclever_opt_tl_unset:c
2599         {
2600           \__zrefclever_opt_varname_type:enn
2601           { \l__zrefclever_setup_type_tl } {#1} { tl }
2602         }

```

```

2603     }
2604     {
2605         \__zrefclever_opt_t1_set:cn
2606         {
2607             \__zrefclever_opt_varname_type:enn
2608             { \l__zrefclever_setup_type_t1 } {#1} { t1 }
2609         }
2610         {##1}
2611     }
2612 } ,
2613 }
2614 }
2615 \keys_define:nn { zref-clever/typesetup }
2616 {
2617     endrange .code:n =
2618     {
2619         \str_case:nnF {#1}
2620         {
2621             { ref }
2622             {
2623                 \__zrefclever_opt_t1_clear:c
2624                 {
2625                     \__zrefclever_opt_varname_type:enn
2626                     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2627                 }
2628                 \__zrefclever_opt_t1_clear:c
2629                 {
2630                     \__zrefclever_opt_varname_type:enn
2631                     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2632                 }
2633             }
2634
2635             { stripprefix }
2636             {
2637                 \__zrefclever_opt_t1_set:cn
2638                 {
2639                     \__zrefclever_opt_varname_type:enn
2640                     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2641                 }
2642                 { __zrefclever_get_endrange_stripprefix }
2643                 \__zrefclever_opt_t1_clear:c
2644                 {
2645                     \__zrefclever_opt_varname_type:enn
2646                     { \l__zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2647                 }
2648             }
2649
2650             { pagecomp }
2651             {
2652                 \__zrefclever_opt_t1_set:cn
2653                 {
2654                     \__zrefclever_opt_varname_type:enn
2655                     { \l__zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2656                 }

```

```

2657         { __zrefclever_get_endrange_pagecomp }
2658     \__zrefclever_opt_tl_clear:c
2659     {
2660         \__zrefclever_opt_varname_type:enn
2661         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2662     }
2663 }
2664
2665 { pagecomp2 }
2666 {
2667     \__zrefclever_opt_tl_set:cn
2668     {
2669         \__zrefclever_opt_varname_type:enn
2670         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2671     }
2672     { __zrefclever_get_endrange_pagecomptwo }
2673     \__zrefclever_opt_tl_clear:c
2674     {
2675         \__zrefclever_opt_varname_type:enn
2676         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2677     }
2678 }
2679
2680 { unset }
2681 {
2682     \__zrefclever_opt_tl_unset:c
2683     {
2684         \__zrefclever_opt_varname_type:enn
2685         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2686     }
2687     \__zrefclever_opt_tl_unset:c
2688     {
2689         \__zrefclever_opt_varname_type:enn
2690         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2691     }
2692 }
2693 }
2694 {
2695     \tl_if_empty:nTF {#1}
2696     {
2697         \msg_warning:nnn { zref-clever }
2698         { endrange-property-undefined } {#1}
2699     }
2700     {
2701         \zref@ifpropundefined {#1}
2702         {
2703             \msg_warning:nnn { zref-clever }
2704             { endrange-property-undefined } {#1}
2705         }
2706         {
2707             \__zrefclever_opt_tl_set:cn
2708             {
2709                 \__zrefclever_opt_varname_type:enn
2710                 { \l__zrefclever_setup_type_tl }

```

```

2711         { endrangefunc } { t1 }
2712     }
2713     { __zrefclever_get_endrange_property }
2714     \__zrefclever_opt_t1_set:cn
2715     {
2716         \__zrefclever_opt_varname_type:enn
2717         { \l__zrefclever_setup_type_t1 }
2718         { endrangeprop } { t1 }
2719     }
2720     {#1}
2721 }
2722 }
2723 }
2724 } ,
2725     endrange .value_required:n = true ,
2726 }
2727 \keys_define:nn { zref-clever/typesetup }
2728 {
2729     refpre .code:n =
2730     {
2731         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2732         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2733         { refpre } { refbounds }
2734     } ,
2735     refpos .code:n =
2736     {
2737         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2738         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2739         { refpos } { refbounds }
2740     } ,
2741     preref .code:n =
2742     {
2743         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2744         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2745         { preref } { refbounds }
2746     } ,
2747     postref .code:n =
2748     {
2749         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2750         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2751         { postref } { refbounds }
2752     } ,
2753 }
2754 \seq_map_inline:Nn
2755 \g__zrefclever_rf_opts_seq_refbounds_seq
2756 {
2757     \keys_define:nn { zref-clever/typesetup }
2758     {
2759         #1 .default:o = \c_novalue_tl ,
2760         #1 .code:n =
2761         {
2762             \tl_if_novalue:nTF {##1}
2763             {
2764                 \__zrefclever_opt_seq_unset:c

```

```

2765         {
2766             \__zrefclever_opt_varname_type:enn
2767             { \l__zrefclever_setup_type_tl } {#1} { seq }
2768         }
2769     }
2770     {
2771         \seq_clear:N \l_tmpa_seq
2772         \__zrefclever_opt_seq_set_clist_split:Nn
2773         \l_tmpa_seq {##1}
2774         \bool_lazy_or:nnTF
2775         { \tl_if_empty_p:n {##1} }
2776         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2777         {
2778             \__zrefclever_opt_seq_set_eq:cN
2779             {
2780                 \__zrefclever_opt_varname_type:enn
2781                 { \l__zrefclever_setup_type_tl } {#1} { seq }
2782             }
2783             \l_tmpa_seq
2784         }
2785         {
2786             \msg_warning:nxxx { zref-clever }
2787             { refbounds-must-be-four }
2788             {#1} { \seq_count:N \l_tmpa_seq }
2789         }
2790     }
2791 } ,
2792 }
2793 }
2794 \seq_map_inline:Nn
2795 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2796 {
2797     \keys_define:nn { zref-clever/typesetup }
2798     {
2799         #1 .choice: ,
2800         #1 / true .code:n =
2801         {
2802             \__zrefclever_opt_bool_set_true:c
2803             {
2804                 \__zrefclever_opt_varname_type:enn
2805                 { \l__zrefclever_setup_type_tl }
2806                 {#1} { bool }
2807             }
2808         } ,
2809         #1 / false .code:n =
2810         {
2811             \__zrefclever_opt_bool_set_false:c
2812             {
2813                 \__zrefclever_opt_varname_type:enn
2814                 { \l__zrefclever_setup_type_tl }
2815                 {#1} { bool }
2816             }
2817         } ,
2818         #1 / unset .code:n =

```

```

2819     {
2820       \__zrefclever_opt_bool_unset:c
2821       {
2822         \__zrefclever_opt_varname_type:enn
2823         { \l__zrefclever_setup_type_tl }
2824         {#1} { bool }
2825       }
2826     } ,
2827     #1 .default:n = true ,
2828     no #1 .meta:n = { #1 = false } ,
2829     no #1 .value_forbidden:n = true ,
2830   }
2831 }

```

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>}
2832 \NewDocumentCommand \zcLanguageSetup { m m }
2833 {
2834   \group_begin:
2835   \__zrefclever_language_if_declared:NTF {#1}
2836   {
2837     \tl_clear:N \l__zrefclever_setup_type_tl
2838     \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
2839     \__zrefclever_opt_seq_get:cNF
2840     {
2841       \__zrefclever_opt_varname_language:nnn
2842       {#1} { declension } { seq }
2843     }
2844     \l__zrefclever_lang_declension_seq
2845     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2846     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2847     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2848     {
2849       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2850       \l__zrefclever_lang_decl_case_tl
2851     }
2852     \__zrefclever_opt_seq_get:cNF
2853     {
2854       \__zrefclever_opt_varname_language:nnn
2855       {#1} { gender } { seq }
2856     }
2857     \l__zrefclever_lang_gender_seq
2858     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2859     \keys_set:nn { zref-clever/langsetup } {#2}

```



```

2860     }
2861     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2862     \group_end:
2863   }
2864   \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

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

```

2865 \keys_define:nn { zref-clever/langsetup }
2866 {
2867   type .code:n =
2868   {
2869     \tl_if_empty:NTF {#1}
2870     { \tl_clear:N \l__zrefclever_setup_type_tl }
2871     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2872   } ,
2873
2874   case .code:n =
2875   {
2876     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2877     {
2878       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2879       { \l__zrefclever_setup_language_tl } {#1}
2880     }
2881     {
2882       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2883       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2884       {
2885         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2886         {#1} { \l__zrefclever_setup_language_tl }
2887         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2888         \l__zrefclever_lang_decl_case_tl
2889       }
2890     }
2891   } ,
2892   case .value_required:n = true ,
2893
2894   gender .value_required:n = true ,
2895   gender .code:n =
2896   {
2897     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2898     {
2899       \msg_warning:nxxxx { zref-clever } { language-no-gender }
2900       { \l__zrefclever_setup_language_tl } { gender } {#1}
2901     }
2902     {
2903       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2904       {
2905         \msg_warning:nnn { zref-clever }
2906         { option-only-type-specific } { gender }
2907       }
2908       {
2909         \seq_clear:N \l_tmpa_seq

```

```

2910         \clist_map_inline:nn {#1}
2911         {
2912             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2913             { \seq_put_right:Nn \l_tmpa_seq {##1} }
2914             {
2915                 \msg_warning:nnxx { zref-clever }
2916                 { gender-not-declared }
2917                 { \l__zrefclever_setup_language_tl } {##1}
2918             }
2919         }
2920     \__zrefclever_opt_seq_gset_eq:cN
2921     {
2922         \__zrefclever_opt_varname_lang_type:eenn
2923         { \l__zrefclever_setup_language_tl }
2924         { \l__zrefclever_setup_type_tl }
2925         { gender }
2926         { seq }
2927     }
2928     \l_tmpa_seq
2929 }
2930 }
2931 },
2932 }
2933 \seq_map_inline:Nn
2934 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2935 {
2936     \keys_define:nn { zref-clever/langsetup }
2937     {
2938         #1 .value_required:n = true ,
2939         #1 .code:n =
2940         {
2941             \tl_if_empty:NnTF \l__zrefclever_setup_type_tl
2942             {
2943                 \__zrefclever_opt_tl_gset:cn
2944                 {
2945                     \__zrefclever_opt_varname_lang_default:enn
2946                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2947                 }
2948                 {##1}
2949             }
2950             {
2951                 \msg_warning:nnn { zref-clever }
2952                 { option-not-type-specific } {#1}
2953             }
2954         } ,
2955     }
2956 }
2957 \seq_map_inline:Nn
2958 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2959 {
2960     \keys_define:nn { zref-clever/langsetup }
2961     {
2962         #1 .value_required:n = true ,
2963         #1 .code:n =

```

```

2964 {
2965   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2966   {
2967     \__zrefclever_opt_tl_gset:cn
2968     {
2969       \__zrefclever_opt_varname_lang_default:enn
2970       { \l__zrefclever_setup_language_tl } {#1} { t1 }
2971     }
2972     {##1}
2973   }
2974   {
2975     \__zrefclever_opt_tl_gset:cn
2976     {
2977       \__zrefclever_opt_varname_lang_type:eenn
2978       { \l__zrefclever_setup_language_tl }
2979       { \l__zrefclever_setup_type_tl }
2980       {#1} { t1 }
2981     }
2982     {##1}
2983   }
2984   } ,
2985 }
2986 }
2987 \keys_define:nn { zref-clever/langsetup }
2988 {
2989   endrange .value_required:n = true ,
2990   endrange .code:n =
2991   {
2992     \str_case:nnF {#1}
2993     {
2994       { ref }
2995       {
2996         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2997         {
2998           \__zrefclever_opt_tl_gclear:c
2999           {
3000             \__zrefclever_opt_varname_lang_default:enn
3001             { \l__zrefclever_setup_language_tl }
3002             { endrangefunc } { t1 }
3003           }
3004           \__zrefclever_opt_tl_gclear:c
3005           {
3006             \__zrefclever_opt_varname_lang_default:enn
3007             { \l__zrefclever_setup_language_tl }
3008             { endrangeprop } { t1 }
3009           }
3010         }
3011         {
3012           \__zrefclever_opt_tl_gclear:c
3013           {
3014             \__zrefclever_opt_varname_lang_type:eenn
3015             { \l__zrefclever_setup_language_tl }
3016             { \l__zrefclever_setup_type_tl }
3017             { endrangefunc } { t1 }

```

```

3018     }
3019     \__zrefclever_opt_tl_gclear:c
3020     {
3021         \__zrefclever_opt_varname_lang_type:enn
3022         { \l__zrefclever_setup_language_tl }
3023         { \l__zrefclever_setup_type_tl }
3024         { endrangeprop } { tl }
3025     }
3026 }
3027 }
3028
3029 { stripprefix }
3030 {
3031     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3032     {
3033         \__zrefclever_opt_tl_gset:cn
3034         {
3035             \__zrefclever_opt_varname_lang_default:enn
3036             { \l__zrefclever_setup_language_tl }
3037             { endrangefunc } { tl }
3038         }
3039         { __zrefclever_get_endrange_stripprefix }
3040         \__zrefclever_opt_tl_gclear:c
3041         {
3042             \__zrefclever_opt_varname_lang_default:enn
3043             { \l__zrefclever_setup_language_tl }
3044             { endrangeprop } { tl }
3045         }
3046     }
3047     {
3048         \__zrefclever_opt_tl_gset:cn
3049         {
3050             \__zrefclever_opt_varname_lang_type:enn
3051             { \l__zrefclever_setup_language_tl }
3052             { \l__zrefclever_setup_type_tl }
3053             { endrangefunc } { tl }
3054         }
3055         { __zrefclever_get_endrange_stripprefix }
3056         \__zrefclever_opt_tl_gclear:c
3057         {
3058             \__zrefclever_opt_varname_lang_type:enn
3059             { \l__zrefclever_setup_language_tl }
3060             { \l__zrefclever_setup_type_tl }
3061             { endrangeprop } { tl }
3062         }
3063     }
3064 }
3065
3066 { pagecomp }
3067 {
3068     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3069     {
3070         \__zrefclever_opt_tl_gset:cn
3071         {

```

```

3072         \_zrefclever_opt_varname_lang_default:enn
3073         { \l_zrefclever_setup_language_tl }
3074         { endrangefunc } { tl }
3075     }
3076     { __zrefclever_get_endrange_pagecomp }
3077     \_zrefclever_opt_tl_gclear:c
3078     {
3079         \_zrefclever_opt_varname_lang_default:enn
3080         { \l_zrefclever_setup_language_tl }
3081         { endrangeprop } { tl }
3082     }
3083 }
3084 {
3085     \_zrefclever_opt_tl_gset:cn
3086     {
3087         \_zrefclever_opt_varname_lang_type:eenn
3088         { \l_zrefclever_setup_language_tl }
3089         { \l_zrefclever_setup_type_tl }
3090         { endrangefunc } { tl }
3091     }
3092     { __zrefclever_get_endrange_pagecomp }
3093     \_zrefclever_opt_tl_gclear:c
3094     {
3095         \_zrefclever_opt_varname_lang_type:eenn
3096         { \l_zrefclever_setup_language_tl }
3097         { \l_zrefclever_setup_type_tl }
3098         { endrangeprop } { tl }
3099     }
3100 }
3101 }
3102
3103 { pagecomp2 }
3104 {
3105     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3106     {
3107         \_zrefclever_opt_tl_gset:cn
3108         {
3109             \_zrefclever_opt_varname_lang_default:enn
3110             { \l_zrefclever_setup_language_tl }
3111             { endrangefunc } { tl }
3112         }
3113         { __zrefclever_get_endrange_pagecomptwo }
3114         \_zrefclever_opt_tl_gclear:c
3115         {
3116             \_zrefclever_opt_varname_lang_default:enn
3117             { \l_zrefclever_setup_language_tl }
3118             { endrangeprop } { tl }
3119         }
3120     }
3121     {
3122         \_zrefclever_opt_tl_gset:cn
3123         {
3124             \_zrefclever_opt_varname_lang_type:eenn
3125             { \l_zrefclever_setup_language_tl }

```

```

3126         { \l__zrefclever_setup_type_tl }
3127         { endrangefunc } { tl }
3128     }
3129     { __zrefclever_get_endrange_pagecomptwo }
3130 \__zrefclever_opt_tl_gclear:c
3131 {
3132     \__zrefclever_opt_varname_lang_type:enn
3133     { \l__zrefclever_setup_language_tl }
3134     { \l__zrefclever_setup_type_tl }
3135     { endrangeprop } { tl }
3136 }
3137 }
3138 }
3139 }
3140 {
3141     \tl_if_empty:nTF {#1}
3142     {
3143         \msg_warning:nnn { zref-clever }
3144         { endrange-property-undefined } {#1}
3145     }
3146     {
3147         \zref@ifpropundefined {#1}
3148         {
3149             \msg_warning:nnn { zref-clever }
3150             { endrange-property-undefined } {#1}
3151         }
3152         {
3153             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3154             {
3155                 \__zrefclever_opt_tl_gset:cn
3156                 {
3157                     \__zrefclever_opt_varname_lang_default:enn
3158                     { \l__zrefclever_setup_language_tl }
3159                     { endrangefunc } { tl }
3160                 }
3161                 { __zrefclever_get_endrange_property }
3162                 \__zrefclever_opt_tl_gset:cn
3163                 {
3164                     \__zrefclever_opt_varname_lang_default:enn
3165                     { \l__zrefclever_setup_language_tl }
3166                     { endrangeprop } { tl }
3167                 }
3168                 {#1}
3169             }
3170         }
3171         {
3172             \__zrefclever_opt_tl_gset:cn
3173             {
3174                 \__zrefclever_opt_varname_lang_type:enn
3175                 { \l__zrefclever_setup_language_tl }
3176                 { \l__zrefclever_setup_type_tl }
3177                 { endrangefunc } { tl }
3178             }
3179             { __zrefclever_get_endrange_property }
3180             \__zrefclever_opt_tl_gset:cn

```

```

3180         {
3181             \__zrefclever_opt_varname_lang_type:eenm
3182             { \l__zrefclever_setup_language_tl }
3183             { \l__zrefclever_setup_type_tl }
3184             { endrangeprop } { tl }
3185         }
3186         {#1}
3187     }
3188 }
3189 }
3190 }
3191 } ,
3192 }
3193 \keys_define:nn { zref-clever/langsetup }
3194 {
3195     refpre .code:n =
3196     {
3197         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3198         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3199         { refpre } { refbounds }
3200     } ,
3201     refpos .code:n =
3202     {
3203         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3204         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3205         { refpos } { refbounds }
3206     } ,
3207     preref .code:n =
3208     {
3209         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3210         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3211         { preref } { refbounds }
3212     } ,
3213     postref .code:n =
3214     {
3215         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3216         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3217         { postref } { refbounds }
3218     } ,
3219 }
3220 \seq_map_inline:Nn
3221 \g__zrefclever_rf_opts_tl_type_names_seq
3222 {
3223     \keys_define:nn { zref-clever/langsetup }
3224     {
3225         #1 .value_required:n = true ,
3226         #1 .code:n =
3227         {
3228             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3229             {
3230                 \msg_warning:nnn { zref-clever }
3231                 { option-only-type-specific } {#1}
3232             }
3233             {

```

```

3234         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3235         {
3236             \__zrefclever_opt_tl_gset:cn
3237             {
3238                 \__zrefclever_opt_varname_lang_type:een
3239                 { \l__zrefclever_setup_language_tl }
3240                 { \l__zrefclever_setup_type_tl }
3241                 {#1} { tl }
3242             }
3243             {##1}
3244         }
3245         {
3246             \__zrefclever_opt_tl_gset:cn
3247             {
3248                 \__zrefclever_opt_varname_lang_type:een
3249                 { \l__zrefclever_setup_language_tl }
3250                 { \l__zrefclever_setup_type_tl }
3251                 { \l__zrefclever_lang_decl_case_tl - #1 }
3252                 { tl }
3253             }
3254             {##1}
3255         }
3256     } ,
3257 }
3258 }
3259 }
3260 \seq_map_inline:Nn
3261 \g__zrefclever_rf_opts_seq_refbounds_seq
3262 {
3263     \keys_define:nn { zref-clever/langsetup }
3264     {
3265         #1 .value_required:n = true ,
3266         #1 .code:n =
3267         {
3268             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3269             {
3270                 \seq_gclear:N \g_tmpa_seq
3271                 \__zrefclever_opt_seq_gset_clist_split:Nn
3272                 \g_tmpa_seq {##1}
3273                 \bool_lazy_or:nnTF
3274                 { \tl_if_empty_p:n {##1} }
3275                 {
3276                     \int_compare_p:nNn
3277                     { \seq_count:N \g_tmpa_seq } = { 4 }
3278                 }
3279                 {
3280                     \__zrefclever_opt_seq_gset_eq:cn
3281                     {
3282                         \__zrefclever_opt_varname_lang_default:enn
3283                         { \l__zrefclever_setup_language_tl }
3284                         {#1} { seq }
3285                     }
3286                     \g_tmpa_seq
3287                 }

```



```

3288         {
3289             \msg_warning:nxxx { zref-clever }
3290             { rebounds-must-be-four }
3291             {#1} { \seq_count:N \g_tmpa_seq }
3292         }
3293     }
3294     {
3295         \seq_gclear:N \g_tmpa_seq
3296         \__zrefclever_opt_seq_gset_clist_split:Nn
3297             \g_tmpa_seq {##1}
3298         \bool_lazy_or:nnTF
3299             { \tl_if_empty_p:n {##1} }
3300             {
3301                 \int_compare_p:nNn
3302                     { \seq_count:N \g_tmpa_seq } = { 4 }
3303             }
3304             {
3305                 \__zrefclever_opt_seq_gset_eq:cN
3306                 {
3307                     \__zrefclever_opt_varname_lang_type:eenn
3308                     { \l__zrefclever_setup_language_tl }
3309                     { \l__zrefclever_setup_type_tl } {#1} { seq }
3310                 }
3311                 \g_tmpa_seq
3312             }
3313             {
3314                 \msg_warning:nxxx { zref-clever }
3315                 { rebounds-must-be-four }
3316                 {#1} { \seq_count:N \g_tmpa_seq }
3317             }
3318         }
3319     } ,
3320 }
3321 }
3322 \seq_map_inline:Nn
3323 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3324 {
3325     \keys_define:nn { zref-clever/langsetup }
3326     {
3327         #1 .choice: ,
3328         #1 / true .code:n =
3329         {
3330             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3331             {
3332                 \__zrefclever_opt_bool_gset_true:c
3333                 {
3334                     \__zrefclever_opt_varname_lang_default:enn
3335                     { \l__zrefclever_setup_language_tl }
3336                     {#1} { bool }
3337                 }
3338             }
3339             {
3340                 \__zrefclever_opt_bool_gset_true:c
3341                 {

```

```

3342         \_zrefclever_opt_varname_lang_type:enn
3343         { \l__zrefclever_setup_language_tl }
3344         { \l__zrefclever_setup_type_tl }
3345         {#1} { bool }
3346     }
3347 }
3348 },
3349 #1 / false .code:n =
3350 {
3351     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3352     {
3353         \_zrefclever_opt_bool_gset_false:c
3354         {
3355             \_zrefclever_opt_varname_lang_default:enn
3356             { \l__zrefclever_setup_language_tl }
3357             {#1} { bool }
3358         }
3359     }
3360     {
3361         \_zrefclever_opt_bool_gset_false:c
3362         {
3363             \_zrefclever_opt_varname_lang_type:enn
3364             { \l__zrefclever_setup_language_tl }
3365             { \l__zrefclever_setup_type_tl }
3366             {#1} { bool }
3367         }
3368     }
3369 },
3370 #1 .default:n = true ,
3371 no #1 .meta:n = { #1 = false } ,
3372 no #1 .value_forbidden:n = true ,
3373 }
3374 }

```

6 User interface

6.1 \zcref

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

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

```

3375 \NewDocumentCommand \zcref { s O { } m }
3376 { \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}
```

```

3377 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3378 {
3379   \group_begin:

```

Set options.

```

3380   \keys_set:nn { zref-clever/reference } {#3}

```

Store arguments values.

```

3381   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3382   \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.

```

3383   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }

```

Process language settings.

```

3384   \__zrefclever_process_language_settings:

```

Integration with zref-check.

```

3385   \bool_lazy_and:nnT
3386     { \l__zrefclever_zrefcheck_available_bool }
3387     { \l__zrefclever_zcref_with_check_bool }
3388     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

3389   \bool_lazy_or:nnT
3390     { \l__zrefclever_typeset_sort_bool }
3391     { \l__zrefclever_typeset_range_bool }
3392     { \__zrefclever_sort_labels: }

```

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

```

3393   \group_begin:
3394   \l__zrefclever_ref_typeset_font_tl
3395   \__zrefclever_typeset_refs:
3396   \group_end:

```

Typeset note.

```

3397   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3398   {
3399     \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3400     { \l__zrefclever_label_type_a_tl }
3401     { \l__zrefclever_ref_language_tl }
3402     \l_tmpa_tl
3403     \l_tmpa_tl
3404     \l__zrefclever_zcref_note_tl
3405   }

```

Integration with zref-check.

```

3406   \bool_lazy_and:nnT
3407     { \l__zrefclever_zrefcheck_available_bool }
3408     { \l__zrefclever_zcref_with_check_bool }
3409     {
3410       \zrefcheck_zcref_end_label_maybe:
3411       \zrefcheck_zcref_run_checks_on_labels:n

```

```

3412         { \l__zrefclever_zcref_labels_seq }
3413     }

```

Integration with mathtools.

```

3414     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3415     {
3416         \__zrefclever_mathtools_showonlyrefs:n
3417         { \l__zrefclever_zcref_labels_seq }
3418     }
3419     \group_end:
3420 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3421 \seq_new:N \l__zrefclever_zcref_labels_seq
3422 \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>}

```

```

3423 \NewDocumentCommand \zcpageref { s O { } m }
3424 {
3425     \group_begin:
3426     \IfBooleanT {#1}
3427     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3428     \zcref [#2, ref = page] {#3}
3429     \group_end:
3430 }

```

(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.

$\backslash l_zrefclever_label_type_a_tl$ $\backslash l_zrefclever_label_type_b_tl$ $\backslash l_zrefclever_label_enclval_a_tl$ $\backslash l_zrefclever_label_enclval_b_tl$ $\backslash l_zrefclever_label_extdoc_a_tl$ $\backslash l_zrefclever_label_extdoc_b_tl$	<p>Auxiliary variables, for use in sorting, and some also in typesetting. Used to store reference information – label properties – of the “current” (a) and “next” (b) labels.</p> <pre> 3431 \tl_new:N \l__zrefclever_label_type_a_tl 3432 \tl_new:N \l__zrefclever_label_type_b_tl 3433 \tl_new:N \l__zrefclever_label_enclval_a_tl 3434 \tl_new:N \l__zrefclever_label_enclval_b_tl 3435 \tl_new:N \l__zrefclever_label_extdoc_a_tl 3436 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre> <p><i>(End definition for $\backslash l_zrefclever_label_type_a_tl$ and others.)</i></p>
$\backslash l_zrefclever_sort_decided_bool$	<p>Auxiliary variable for $\backslash _zrefclever_sort_default_same_type:nn$, signals if the sorting between two labels has been decided or not.</p> <pre> 3437 \bool_new:N \l__zrefclever_sort_decided_bool </pre> <p><i>(End definition for $\backslash l_zrefclever_sort_decided_bool$.)</i></p>
$\backslash l_zrefclever_sort_prior_a_int$ $\backslash l_zrefclever_sort_prior_b_int$	<p>Auxiliary variables for $\backslash _zrefclever_sort_default_different_types:nn$. Store the sort priority of the “current” and “next” labels.</p> <pre> 3438 \int_new:N \l__zrefclever_sort_prior_a_int 3439 \int_new:N \l__zrefclever_sort_prior_b_int </pre> <p><i>(End definition for $\backslash l_zrefclever_sort_prior_a_int$ and $\backslash l_zrefclever_sort_prior_b_int$.)</i></p>
$\backslash l_zrefclever_label_types_seq$	<p>Stores the order in which reference types appear in the label list supplied by the user in $\backslash zcref$. This variable is populated by $\backslash _zrefclever_label_type_put_new_right:n$ at the start of $\backslash _zrefclever_sort_labels:$. This order is required as a “last resort” sort criterion between the reference types, for use in $\backslash _zrefclever_sort_default_different_types:nn$.</p> <pre> 3440 \seq_new:N \l__zrefclever_label_types_seq </pre> <p><i>(End definition for $\backslash l_zrefclever_label_types_seq$.)</i></p>
$\backslash _zrefclever_sort_labels:$	<p>The main sorting function. It does not receive arguments, but it is expected to be run inside $\backslash _zrefclever_zcref:nnnn$ where a number of environment variables are to be set appropriately. In particular, $\backslash l_zrefclever_zcref_labels_seq$ should contain the labels received as argument to $\backslash zcref$, and the function performs its task by sorting this variable.</p> <pre> 3441 \cs_new_protected:Npn _zrefclever_sort_labels: 3442 { </pre> <p>Store label types sequence.</p> <pre> 3443 \seq_clear:N \l__zrefclever_label_types_seq 3444 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page } 3445 { 3446 \seq_map_function:NN \l__zrefclever_zcref_labels_seq 3447 _zrefclever_label_type_put_new_right:n 3448 } </pre>

Sort.

```

3449 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3450 {
3451   \zref@ifrefundefined {##1}
3452   {
3453     \zref@ifrefundefined {##2}
3454     {
3455       % Neither label is defined.
3456       \sort_return_same:
3457     }
3458     {
3459       % The second label is defined, but the first isn't, leave the
3460       % undefined first (to be more visible).
3461       \sort_return_same:
3462     }
3463   }
3464   {
3465     \zref@ifrefundefined {##2}
3466     {
3467       % The first label is defined, but the second isn't, bring the
3468       % second forward.
3469       \sort_return_swapped:
3470     }
3471     {
3472       % The interesting case: both labels are defined. References
3473       % to the "default" property or to the "page" are quite
3474       % different with regard to sorting, so we branch them here to
3475       % specialized functions.
3476       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3477       { \__zrefclever_sort_page:nn {##1} {##2} }
3478       { \__zrefclever_sort_default:nn {##1} {##2} }
3479     }
3480   }
3481 }
3482 }
```

(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>}

3483 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3484 {
3485   \__zrefclever_extract_default:Nnnn
3486   \l__zrefclever_label_type_a_tl {#1} {zc@type} {}
3487   \seq_if_in:NVF \l__zrefclever_label_types_seq
```

```

3488 \l__zrefclever_label_type_a_tl
3489 {
3490   \seq_put_right:NV \l__zrefclever_label_types_seq
3491   \l__zrefclever_label_type_a_tl
3492 }
3493 }

```

(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}}

3494 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3495 {
3496   \__zrefclever_extract_default:Nnnn
3497   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3498   \__zrefclever_extract_default:Nnnn
3499   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3500
3501   \tl_if_eq:NNTF
3502   \l__zrefclever_label_type_a_tl
3503   \l__zrefclever_label_type_b_tl
3504   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3505   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3506 }

```

(End definition for __zrefclever_sort_default:nn.)

```

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

3507 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3508 {
3509   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3510   {#1} {zc@enclval} { }
3511   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3512   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3513   {#2} {zc@enclval} { }
3514   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3515   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3516   {#1} {externaldocument} { }
3517   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3518   {#2} {externaldocument} { }
3519
3520   \bool_set_false:N \l__zrefclever_sort_decided_bool
3521
3522   % First we check if there's any "external document" difference (coming
3523   % from 'zref-xr') and, if so, sort based on that.
3524   \tl_if_eq:NNTF
3525   \l__zrefclever_label_extdoc_a_tl
3526   \l__zrefclever_label_extdoc_b_tl
3527   {

```

```

3528 \bool_if:nTF
3529 {
3530   \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3531   ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3532 }
3533 {
3534   \bool_set_true:N \l__zrefclever_sort_decided_bool
3535   \sort_return_same:
3536 }
3537 {
3538   \bool_if:nTF
3539   {
3540     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3541     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3542   }
3543   {
3544     \bool_set_true:N \l__zrefclever_sort_decided_bool
3545     \sort_return_swapped:
3546   }
3547   {
3548     \bool_set_true:N \l__zrefclever_sort_decided_bool
3549     % Two different "external documents": last resort, sort by the
3550     % document name itself.
3551     \str_compare:eNeTF
3552     { \l__zrefclever_label_extdoc_b_tl } <
3553     { \l__zrefclever_label_extdoc_a_tl }
3554     { \sort_return_swapped: }
3555     { \sort_return_same: }
3556   }
3557 }
3558 }
3559
3560 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3561 {
3562   \bool_if:nTF
3563   {
3564     % Both are empty: neither label has any (further) "enclosing
3565     % counters" (left).
3566     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3567     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3568   }
3569   {
3570     \bool_set_true:N \l__zrefclever_sort_decided_bool
3571     \int_compare:nNnTF
3572     { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
3573     >
3574     { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
3575     { \sort_return_swapped: }
3576     { \sort_return_same: }
3577   }
3578   {
3579     \bool_if:nTF
3580     {
3581       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.

```



```

3582     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3583   }
3584   {
3585     \bool_set_true:N \l__zrefclever_sort_decided_bool
3586     \int_compare:nNnTF
3587       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3588       >
3589       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3590       { \sort_return_swapped: }
3591       { \sort_return_same: }
3592   }
3593   {
3594     \bool_if:nTF
3595     {
3596       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3597       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3598     }
3599     {
3600       \bool_set_true:N \l__zrefclever_sort_decided_bool
3601       \int_compare:nNnTF
3602         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3603         <
3604         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3605         { \sort_return_same: }
3606         { \sort_return_swapped: }
3607     }
3608     {
3609       % Neither is empty: we can compare the values of the
3610       % current enclosing counter in the loop, if they are
3611       % equal, we are still in the loop, if they are not, a
3612       % sorting decision can be made directly.
3613       \int_compare:nNnTF
3614         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3615         =
3616         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3617         {
3618           \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3619             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3620           \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3621             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3622         }
3623         {
3624           \bool_set_true:N \l__zrefclever_sort_decided_bool
3625           \int_compare:nNnTF
3626             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3627             >
3628             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3629             { \sort_return_swapped: }
3630             { \sort_return_same: }
3631         }
3632     }
3633   }
3634 }
3635 }

```

```
3636 }
```

(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}
```

```
3637 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
3638 {
```

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

```
3639 \int_zero:N \l__zrefclever_sort_prior_a_int
3640 \int_zero:N \l__zrefclever_sort_prior_b_int
3641 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3642 {
3643   \tl_if_eq:nnTF {##2} {{othertypes}}
3644   {
3645     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3646     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3647     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3648     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3649   }
3650   {
3651     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3652     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3653     {
3654       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3655       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3656     }
3657   }
3658 }
```

Then do the actual sorting.

```
3659 \bool_if:nTF
3660 {
3661   \int_compare_p:nNn
3662   { \l__zrefclever_sort_prior_a_int } <
3663   { \l__zrefclever_sort_prior_b_int }
3664 }
3665 { \sort_return_same: }
3666 {
3667   \bool_if:nTF
3668   {
3669     \int_compare_p:nNn
3670     { \l__zrefclever_sort_prior_a_int } >
3671     { \l__zrefclever_sort_prior_b_int }
3672   }
3673   { \sort_return_swapped: }
3674   {
3675     % Sort priorities are equal: the type that occurs first in
3676     % ‘labels’, as given by the user, is kept (or brought) forward.
3677     \seq_map_inline:Nn \l__zrefclever_label_types_seq
3678     {
3679       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
```

```

3680         { \seq_map_break:n { \sort_return_same: } }
3681         {
3682             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3683             { \seq_map_break:n { \sort_return_swapped: } }
3684         }
3685     }
3686 }
3687 }
3688 }

```

(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}

3689 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3690 {
3691     \int_compare:nNnTF
3692     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3693     >
3694     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3695     { \sort_return_swapped: }
3696     { \sort_return_same: }
3697 }

```

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

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

Variables

\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_last_bool
\l_zrefclever_last_of_type_bool

Auxiliary variables for _zrefclever_typeset_refs: main stack control.

3698 \seq_new:N \l_zrefclever_typeset_labels_seq
3699 \bool_new:N \l_zrefclever_typeset_last_bool
3700 \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
\l_zrefclever_label_count_int
\l_zrefclever_ref_count_int

Auxiliary variables for _zrefclever_typeset_refs: main counters.

3701 \int_new:N \l_zrefclever_type_count_int
3702 \int_new:N \l_zrefclever_label_count_int
3703 \int_new:N \l_zrefclever_ref_count_int

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

\l_zrefclever_label_a_tl
\l_zrefclever_label_b_tl
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl

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

3704 \tl_new:N \l_zrefclever_label_a_tl
3705 \tl_new:N \l_zrefclever_label_b_tl
3706 \tl_new:N \l_zrefclever_typeset_queue_prev_tl
3707 \tl_new:N \l_zrefclever_typeset_queue_curr_tl
3708 \tl_new:N \l_zrefclever_type_first_label_tl
3709 \tl_new:N \l_zrefclever_type_first_label_type_tl

(End definition for \l_zrefclever_label_a_tl and others.)

\l_zrefclever_type_name_tl
\l_zrefclever_name_in_link_bool
\l_zrefclever_type_name_missing_bool
\l_zrefclever_name_format_tl
\l_zrefclever_name_format_fallback_tl
\l_zrefclever_type_name_gender_seq

Auxiliary variables for _zrefclever_typeset_refs: type name handling.

3710 \tl_new:N \l_zrefclever_type_name_tl
3711 \bool_new:N \l_zrefclever_name_in_link_bool
3712 \bool_new:N \l_zrefclever_type_name_missing_bool
3713 \tl_new:N \l_zrefclever_name_format_tl
3714 \tl_new:N \l_zrefclever_name_format_fallback_tl
3715 \seq_new:N \l_zrefclever_type_name_gender_seq

(End definition for \l_zrefclever_type_name_tl and others.)

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

Auxiliary variables for _zrefclever_typeset_refs: range handling.

3716 \int_new:N \l_zrefclever_range_count_int
3717 \int_new:N \l_zrefclever_range_same_count_int
3718 \tl_new:N \l_zrefclever_range_beg_label_tl
3719 \bool_new:N \l_zrefclever_range_beg_is_first_bool
3720 \tl_new:N \l_zrefclever_range_end_ref_tl
3721 \bool_new:N \l_zrefclever_next_maybe_range_bool
3722 \bool_new:N \l_zrefclever_next_is_same_bool

(End definition for \l_zrefclever_range_count_int and others.)

```

\l__zrefclever_tpairsep_tl
\l__zrefclever_tlistsep_tl
\l__zrefclever_tlastsep_tl
\l__zrefclever_namesep_tl
\l__zrefclever_pairsep_tl
\l__zrefclever_listsep_tl
\l__zrefclever_lastsep_tl
\l__zrefclever_rangeseq_tl
\l__zrefclever_namefont_tl
\l__zrefclever_reffont_tl
  \l__zrefclever_endrangefunc_tl
  \l__zrefclever_endrangeprop_tl
  \l__zrefclever_cap_bool
\l__zrefclever_abbrev_bool
  \l__zrefclever_rangetopair_bool

```

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

```

3723 \tl_new:N \l__zrefclever_tpairsep_tl
3724 \tl_new:N \l__zrefclever_tlistsep_tl
3725 \tl_new:N \l__zrefclever_tlastsep_tl
3726 \tl_new:N \l__zrefclever_namesep_tl
3727 \tl_new:N \l__zrefclever_pairsep_tl
3728 \tl_new:N \l__zrefclever_listsep_tl
3729 \tl_new:N \l__zrefclever_lastsep_tl
3730 \tl_new:N \l__zrefclever_rangeseq_tl
3731 \tl_new:N \l__zrefclever_namefont_tl
3732 \tl_new:N \l__zrefclever_reffont_tl
3733 \tl_new:N \l__zrefclever_endrangefunc_tl
3734 \tl_new:N \l__zrefclever_endrangeprop_tl
3735 \bool_new:N \l__zrefclever_cap_bool
3736 \bool_new:N \l__zrefclever_abbrev_bool
3737 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

```

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

```

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

```

3738 \seq_new:N \l__zrefclever_refbounds_first_seq
3739 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3740 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3741 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3742 \seq_new:N \l__zrefclever_refbounds_mid_seq
3743 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3744 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3745 \seq_new:N \l__zrefclever_refbounds_last_seq
3746 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3747 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3748 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3749 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

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

```

\l__zrefclever_verbose_testing_bool

```

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

```

3750 \bool_new:N \l__zrefclever_verbose_testing_bool

```

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

```

\__zrefclever_typeset_refs:

```

Main typesetting function for `\zceref`.

```

3751 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3752 {
3753   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3754   \l__zrefclever_zceref_labels_seq
3755   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3756   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3757   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

3758 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3759 \tl_clear:N \l__zrefclever_range_beg_label_tl
3760 \tl_clear:N \l__zrefclever_range_end_ref_tl
3761 \int_zero:N \l__zrefclever_label_count_int
3762 \int_zero:N \l__zrefclever_type_count_int
3763 \int_zero:N \l__zrefclever_ref_count_int
3764 \int_zero:N \l__zrefclever_range_count_int
3765 \int_zero:N \l__zrefclever_range_same_count_int
3766 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3767 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3768
3769 % Get type block options (not type-specific).
3770 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3771   { \l__zrefclever_label_type_a_tl }
3772   { \l__zrefclever_ref_language_tl }
3773   \l__zrefclever_tpairsep_tl
3774 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3775   { \l__zrefclever_label_type_a_tl }
3776   { \l__zrefclever_ref_language_tl }
3777   \l__zrefclever_tlistsep_tl
3778 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3779   { \l__zrefclever_label_type_a_tl }
3780   { \l__zrefclever_ref_language_tl }
3781   \l__zrefclever_tlastsep_tl
3782
3783 % Process label stack.
3784 \bool_set_false:N \l__zrefclever_typeset_last_bool
3785 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3786   {
3787     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3788     \l__zrefclever_label_a_tl
3789     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3790       {
3791         \tl_clear:N \l__zrefclever_label_b_tl
3792         \bool_set_true:N \l__zrefclever_typeset_last_bool
3793       }
3794       {
3795         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3796         \l__zrefclever_label_b_tl
3797       }
3798
3799     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3800       {
3801         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3802         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3803       }
3804       {
3805         \__zrefclever_extract_default:NVnn
3806         \l__zrefclever_label_type_a_tl
3807         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3808         \__zrefclever_extract_default:NVnn
3809         \l__zrefclever_label_type_b_tl
3810         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3811       }

```

```

3812
3813 % First, we establish whether the "current label" (i.e. 'a') is the
3814 % last one of its type. This can happen because the "next label"
3815 % (i.e. 'b') is of a different type (or different definition status),
3816 % or because we are at the end of the list.
3817 \bool_if:NTF \l__zrefclever_typeset_last_bool
3818 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3819 {
3820   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3821   {
3822     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3823     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3824     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3825   }
3826   {
3827     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3828     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3829     {
3830       % Neither is undefined, we must check the types.
3831       \tl_if_eq:NNTF
3832         \l__zrefclever_label_type_a_tl
3833         \l__zrefclever_label_type_b_tl
3834         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3835         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3836     }
3837   }
3838 }
3839
3840 % Handle warnings in case of reference or type undefined.
3841 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3842 \zref@refused { \l__zrefclever_label_a_tl }
3843 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3844 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3845 {}
3846 {
3847   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3848   {
3849     \msg_warning:nxx { zref-clever } { missing-type }
3850     { \l__zrefclever_label_a_tl }
3851   }
3852   \zref@ifrefcontainsprop
3853     { \l__zrefclever_label_a_tl }
3854     { \l__zrefclever_ref_property_tl }
3855     { }
3856     {
3857       \msg_warning:nxxx { zref-clever } { missing-property }
3858       { \l__zrefclever_ref_property_tl }
3859       { \l__zrefclever_label_a_tl }
3860     }
3861 }
3862
3863 % Get possibly type-specific separators, refbounds, font and other
3864 % options, once per type.
3865 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```



```

3866 {
3867   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3868   { \l__zrefclever_label_type_a_tl }
3869   { \l__zrefclever_ref_language_tl }
3870   \l__zrefclever_namesep_tl
3871   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3872   { \l__zrefclever_label_type_a_tl }
3873   { \l__zrefclever_ref_language_tl }
3874   \l__zrefclever_pairsep_tl
3875   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3876   { \l__zrefclever_label_type_a_tl }
3877   { \l__zrefclever_ref_language_tl }
3878   \l__zrefclever_listsep_tl
3879   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3880   { \l__zrefclever_label_type_a_tl }
3881   { \l__zrefclever_ref_language_tl }
3882   \l__zrefclever_lastsep_tl
3883   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3884   { \l__zrefclever_label_type_a_tl }
3885   { \l__zrefclever_ref_language_tl }
3886   \l__zrefclever_rangesep_tl
3887   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3888   { \l__zrefclever_label_type_a_tl }
3889   { \l__zrefclever_ref_language_tl }
3890   \l__zrefclever_namefont_tl
3891   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3892   { \l__zrefclever_label_type_a_tl }
3893   { \l__zrefclever_ref_language_tl }
3894   \l__zrefclever_reffont_tl
3895   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3896   { \l__zrefclever_label_type_a_tl }
3897   { \l__zrefclever_ref_language_tl }
3898   \l__zrefclever_endrangefunc_tl
3899   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3900   { \l__zrefclever_label_type_a_tl }
3901   { \l__zrefclever_ref_language_tl }
3902   \l__zrefclever_endrangeprop_tl
3903   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3904   { \l__zrefclever_label_type_a_tl }
3905   { \l__zrefclever_ref_language_tl }
3906   \l__zrefclever_cap_bool
3907   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3908   { \l__zrefclever_label_type_a_tl }
3909   { \l__zrefclever_ref_language_tl }
3910   \l__zrefclever_abbrev_bool
3911   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3912   { \l__zrefclever_label_type_a_tl }
3913   { \l__zrefclever_ref_language_tl }
3914   \l__zrefclever_rangetopair_bool
3915   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3916   { \l__zrefclever_label_type_a_tl }
3917   { \l__zrefclever_ref_language_tl }
3918   \l__zrefclever_refbounds_first_seq
3919   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3920         { \l__zrefclever_label_type_a_tl }
3921         { \l__zrefclever_ref_language_tl }
3922         \l__zrefclever_refbounds_first_sg_seq
3923     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3924         { \l__zrefclever_label_type_a_tl }
3925         { \l__zrefclever_ref_language_tl }
3926         \l__zrefclever_refbounds_first_pb_seq
3927     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3928         { \l__zrefclever_label_type_a_tl }
3929         { \l__zrefclever_ref_language_tl }
3930         \l__zrefclever_refbounds_first_rb_seq
3931     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3932         { \l__zrefclever_label_type_a_tl }
3933         { \l__zrefclever_ref_language_tl }
3934         \l__zrefclever_refbounds_mid_seq
3935     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3936         { \l__zrefclever_label_type_a_tl }
3937         { \l__zrefclever_ref_language_tl }
3938         \l__zrefclever_refbounds_mid_rb_seq
3939     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3940         { \l__zrefclever_label_type_a_tl }
3941         { \l__zrefclever_ref_language_tl }
3942         \l__zrefclever_refbounds_mid_re_seq
3943     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3944         { \l__zrefclever_label_type_a_tl }
3945         { \l__zrefclever_ref_language_tl }
3946         \l__zrefclever_refbounds_last_seq
3947     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3948         { \l__zrefclever_label_type_a_tl }
3949         { \l__zrefclever_ref_language_tl }
3950         \l__zrefclever_refbounds_last_pe_seq
3951     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3952         { \l__zrefclever_label_type_a_tl }
3953         { \l__zrefclever_ref_language_tl }
3954         \l__zrefclever_refbounds_last_re_seq
3955 }
3956
3957 % Here we send this to a couple of auxiliary functions.
3958 \bool_if:NTF \l__zrefclever_last_of_type_bool
3959 % There exists no next label of the same type as the current.
3960 { \__zrefclever_typeset_refs_last_of_type: }
3961 % There exists a next label of the same type as the current.
3962 { \__zrefclever_typeset_refs_not_last_of_type: }
3963 }
3964 }

```

(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.

```

3965 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3966 {
3967   % Process the current label to the current queue.
3968   \int_case:nnF { \l__zrefclever_label_count_int }
3969   {
3970     % It is the last label of its type, but also the first one, and that's
3971     % what matters here: just store it.
3972     % Test: 'zc-typeset01.lvt': "Last of type: single"
3973     { 0 }
3974     {
3975       \tl_set:NV \l__zrefclever_type_first_label_tl
3976       \l__zrefclever_label_a_tl
3977       \tl_set:NV \l__zrefclever_type_first_label_type_tl
3978       \l__zrefclever_label_type_a_tl
3979       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3980       \l__zrefclever_refbounds_first_sg_seq
3981       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3982     }
3983
3984     % The last is the second: we have a pair (if not repeated).
3985     % Test: 'zc-typeset01.lvt': "Last of type: pair"
3986     { 1 }
3987     {
3988       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
3989       {
3990         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
3991         \l__zrefclever_refbounds_first_sg_seq
3992         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
3993       }
3994       {
3995         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
3996         {
3997           \exp_not:V \l__zrefclever_pairsep_tl
3998           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
3999           \l__zrefclever_refbounds_last_pe_seq
4000         }
4001         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4002         \l__zrefclever_refbounds_first_pb_seq
4003         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4004       }
4005     }
4006   }
4007   % Last is third or more of its type: without repetition, we'd have the
4008   % last element on a list, but control for possible repetition.
4009   {
4010     \int_case:nnF { \l__zrefclever_range_count_int }
4011     {
4012       % There was no range going on.
4013       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4014       { 0 }

```

```

4015 {
4016   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4017   {
4018     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4019     {
4020       \exp_not:V \l__zrefclever_pairsep_tl
4021       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4022       \l__zrefclever_refbounds_last_pe_seq
4023     }
4024   }
4025   {
4026     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4027     {
4028       \exp_not:V \l__zrefclever_lastsep_tl
4029       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4030       \l__zrefclever_refbounds_last_seq
4031     }
4032   }
4033 }
4034 % Last in the range is also the second in it.
4035 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4036 { 1 }
4037 {
4038   \int_compare:nNnTF
4039   { \l__zrefclever_range_same_count_int } = { 1 }
4040   {
4041     % We know 'range_beg_is_first_bool' is false, since this is
4042     % the second element in the range, but the third or more in
4043     % the type list.
4044     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4045     {
4046       \exp_not:V \l__zrefclever_pairsep_tl
4047       \__zrefclever_get_ref:VN
4048       \l__zrefclever_range_beg_label_tl
4049       \l__zrefclever_refbounds_last_pe_seq
4050     }
4051     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4052     \l__zrefclever_refbounds_first_pb_seq
4053     \bool_set_true:N
4054     \l__zrefclever_type_first_refbounds_set_bool
4055   }
4056   {
4057     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4058     {
4059       \exp_not:V \l__zrefclever_listsep_tl
4060       \__zrefclever_get_ref:VN
4061       \l__zrefclever_range_beg_label_tl
4062       \l__zrefclever_refbounds_mid_seq
4063       \exp_not:V \l__zrefclever_lastsep_tl
4064       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4065       \l__zrefclever_refbounds_last_seq
4066     }
4067   }
4068 }

```

```

4069 }
4070 % Last in the range is third or more in it.
4071 {
4072   \int_case:nnF
4073   {
4074     \l__zrefclever_range_count_int -
4075     \l__zrefclever_range_same_count_int
4076   }
4077   {
4078     % Repetition, not a range.
4079     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4080     { 0 }
4081     {
4082       % If 'range_beg_is_first_bool' is true, it means it was also
4083       % the first of the type, and hence its typesetting was
4084       % already handled, and we just have to set rebounds.
4085       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4086       {
4087         \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4088         \l__zrefclever_rebounds_first_sg_seq
4089         \bool_set_true:N
4090         \l__zrefclever_type_first_rebounds_set_bool
4091       }
4092       {
4093         \int_compare:nNnTF
4094         { \l__zrefclever_ref_count_int } < { 2 }
4095         {
4096           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4097           {
4098             \exp_not:V \l__zrefclever_pairsep_tl
4099             \__zrefclever_get_ref:VN
4100             \l__zrefclever_range_beg_label_tl
4101             \l__zrefclever_rebounds_last_pe_seq
4102           }
4103         }
4104         {
4105           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4106           {
4107             \exp_not:V \l__zrefclever_lastsep_tl
4108             \__zrefclever_get_ref:VN
4109             \l__zrefclever_range_beg_label_tl
4110             \l__zrefclever_rebounds_last_seq
4111           }
4112         }
4113       }
4114     }
4115     % A 'range', but with no skipped value, treat as pair if range
4116     % started with first of type, otherwise as list.
4117     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4118     { 1 }
4119     {
4120       % Ditto.
4121       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4122       {

```

```

4123         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4124         \l__zrefclever_refbounds_first_pb_seq
4125     \bool_set_true:N
4126         \l__zrefclever_type_first_refbounds_set_bool
4127     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4128     {
4129         \exp_not:V \l__zrefclever_pairsep_tl
4130         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4131         \l__zrefclever_refbounds_last_pe_seq
4132     }
4133 }
4134 {
4135     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4136     {
4137         \exp_not:V \l__zrefclever_listsep_tl
4138         \__zrefclever_get_ref:VN
4139         \l__zrefclever_range_beg_label_tl
4140         \l__zrefclever_refbounds_mid_seq
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 \l__zrefclever_label_a_tl
4146         \l__zrefclever_refbounds_last_seq
4147     }
4148 }
4149 }
4150 }
4151 {
4152     % An actual range.
4153     % Test: 'zc-typeset01.lvt': "Last of type: range"
4154     % Ditto.
4155     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4156     {
4157         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4158         \l__zrefclever_refbounds_first_rb_seq
4159         \bool_set_true:N
4160         \l__zrefclever_type_first_refbounds_set_bool
4161     }
4162     {
4163         \int_compare:nNnTF
4164         { \l__zrefclever_ref_count_int } < { 2 }
4165         {
4166             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4167             {
4168                 \exp_not:V \l__zrefclever_pairsep_tl
4169                 \__zrefclever_get_ref:VN
4170                 \l__zrefclever_range_beg_label_tl
4171                 \l__zrefclever_refbounds_mid_rb_seq
4172             }
4173             \seq_set_eq:NN
4174             \l__zrefclever_type_first_refbounds_seq
4175             \l__zrefclever_refbounds_first_pb_seq
4176             \bool_set_true:N

```

```

4177         \l__zrefclever_type_first_refbounds_set_bool
4178     }
4179     {
4180         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4181         {
4182             \exp_not:V \l__zrefclever_lastsep_tl
4183             \__zrefclever_get_ref:VN
4184             \l__zrefclever_range_beg_label_tl
4185             \l__zrefclever_refbounds_mid_rb_seq
4186         }
4187     }
4188 }
4189 \bool_lazy_and:nnTF
4190 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4191 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4192 {
4193     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4194     \l__zrefclever_range_beg_label_tl
4195     \l__zrefclever_label_a_tl
4196     \l__zrefclever_range_end_ref_tl
4197     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4198     {
4199         \exp_not:V \l__zrefclever_rangesep_tl
4200         \__zrefclever_get_ref_endrange:VVN
4201         \l__zrefclever_label_a_tl
4202         \l__zrefclever_range_end_ref_tl
4203         \l__zrefclever_refbounds_last_re_seq
4204     }
4205 }
4206 {
4207     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4208     {
4209         \exp_not:V \l__zrefclever_rangesep_tl
4210         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4211         \l__zrefclever_refbounds_last_re_seq
4212     }
4213 }
4214 }
4215 }
4216 }
4217
4218 % Handle "range" option. The idea is simple: if the queue is not empty,
4219 % we replace it with the end of the range (or pair). We can still
4220 % retrieve the end of the range from 'label_a' since we know to be
4221 % processing the last label of its type at this point.
4222 \bool_if:NT \l__zrefclever_typeset_range_bool
4223 {
4224     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4225     {
4226         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4227         { }
4228         {
4229             \msg_warning:nxx { zref-clever } { single-element-range }
4230             { \l__zrefclever_type_first_label_type_tl }

```

```

4231     }
4232   }
4233   {
4234     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4235     \bool_if:NT \l__zrefclever_rangetopair_bool
4236     {
4237       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4238       { }
4239       {
4240         \__zrefclever_labels_in_sequence:nn
4241         { \l__zrefclever_type_first_label_tl }
4242         { \l__zrefclever_label_a_tl }
4243       }
4244     }
4245     % Test: 'zc-typeset01.lvt': "Last of type: option range"
4246     % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4247     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4248     {
4249       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4250       {
4251         \exp_not:V \l__zrefclever_pairsep_tl
4252         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4253         \l__zrefclever_refbounds_last_pe_seq
4254       }
4255       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4256       \l__zrefclever_refbounds_first_pb_seq
4257       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4258     }
4259     {
4260       \bool_lazy_and:nnTF
4261       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4262       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4263       {
4264         % We must get 'type_first_label_tl' instead of
4265         % 'range_beg_label_tl' here, since it is not necessary
4266         % that the first of type was actually starting a range for
4267         % the 'range' option to be used.
4268         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4269         \l__zrefclever_type_first_label_tl
4270         \l__zrefclever_label_a_tl
4271         \l__zrefclever_range_end_ref_tl
4272         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4273         {
4274           \exp_not:V \l__zrefclever_rangesep_tl
4275           \__zrefclever_get_ref_endrange:VVN
4276           \l__zrefclever_label_a_tl
4277           \l__zrefclever_range_end_ref_tl
4278           \l__zrefclever_refbounds_last_re_seq
4279         }
4280       }
4281     }
4282     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4283     {
4284       \exp_not:V \l__zrefclever_rangesep_tl

```



```

4285         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4286         \l__zrefclever_refbounds_last_re_seq
4287     }
4288 }
4289 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4290 \l__zrefclever_refbounds_first_rb_seq
4291 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4292 }
4293 }
4294 }
4295
4296 % If none of the special cases for the first of type refbounds have been
4297 % set, do it.
4298 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4299 {
4300     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4301     \l__zrefclever_refbounds_first_seq
4302 }
4303
4304 % Now that the type block is finished, we can add the name and the first
4305 % ref to the queue. Also, if "typeset" option is not "both", handle it
4306 % here as well.
4307 \_zrefclever_type_name_setup:
4308 \bool_if:nTF
4309 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4310 {
4311     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4312     { \_zrefclever_get_ref_first: }
4313 }
4314 {
4315     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4316     {
4317         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4318         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4319         {
4320             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4321             \l__zrefclever_type_first_refbounds_seq
4322         }
4323     }
4324     {
4325         \bool_if:NTF \l__zrefclever_typeset_name_bool
4326         {
4327             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4328             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4329             {
4330                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4331                 {
4332                     \exp_not:N \group_begin:
4333                     \exp_not:V \l__zrefclever_namefont_tl
4334                     \_zrefclever_hyperlink:nnn
4335                     {
4336                         \_zrefclever_extract_url_unexp:V
4337                         \l__zrefclever_type_first_label_tl
4338                     }

```

```

4339         {
4340             \__zrefclever_extract_unexp:Vnn
4341             \l__zrefclever_type_first_label_tl
4342             { anchor } { }
4343         }
4344         { \exp_not:V \l__zrefclever_type_name_tl }
4345         \exp_not:N \group_end:
4346     }
4347     {
4348         \exp_not:N \group_begin:
4349         \exp_not:V \l__zrefclever_namefont_tl
4350         \exp_not:V \l__zrefclever_type_name_tl
4351         \exp_not:N \group_end:
4352     }
4353 }
4354 }
4355 {
4356     % Logically, this case would correspond to "typeset=none", but
4357     % it should not occur, given that the options are set up to
4358     % typeset either "ref" or "name". Still, leave here a
4359     % sensible fallback, equal to the behavior of "both".
4360     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4361     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4362     { \__zrefclever_get_ref_first: }
4363 }
4364 }
4365 }
4366
4367 % Typeset the previous type block, if there is one.
4368 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4369 {
4370     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4371     { \l__zrefclever_tlistsep_tl }
4372     \l__zrefclever_typeset_queue_prev_tl
4373 }
4374
4375 % Extra log for testing.
4376 \bool_if:NT \l__zrefclever_verbose_testing_bool
4377 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4378
4379 % Wrap up loop, or prepare for next iteration.
4380 \bool_if:NTF \l__zrefclever_typeset_last_bool
4381 {
4382     % We are finishing, typeset the current queue.
4383     \int_case:nnF { \l__zrefclever_type_count_int }
4384     {
4385         % Single type.
4386         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4387         { 0 }
4388         { \l__zrefclever_typeset_queue_curr_tl }
4389         % Pair of types.
4390         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4391         { 1 }
4392     }

```

```

4393         \l__zrefclever_tpairsep_tl
4394         \l__zrefclever_typeset_queue_curr_tl
4395     }
4396 }
4397 {
4398     % Last in list of types.
4399     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4400     \l__zrefclever_tlastsep_tl
4401     \l__zrefclever_typeset_queue_curr_tl
4402 }
4403 % And nudge in case of multitype reference.
4404 \bool_lazy_all:nT
4405 {
4406     { \l__zrefclever_nudge_enabled_bool }
4407     { \l__zrefclever_nudge_multitype_bool }
4408     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4409 }
4410 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4411 }
4412 {
4413     % There are further labels, set variables for next iteration.
4414     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4415         \l__zrefclever_typeset_queue_curr_tl
4416     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4417     \tl_clear:N \l__zrefclever_type_first_label_tl
4418     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4419     \tl_clear:N \l__zrefclever_range_beg_label_tl
4420     \tl_clear:N \l__zrefclever_range_end_ref_tl
4421     \int_zero:N \l__zrefclever_label_count_int
4422     \int_zero:N \l__zrefclever_ref_count_int
4423     \int_incr:N \l__zrefclever_type_count_int
4424     \int_zero:N \l__zrefclever_range_count_int
4425     \int_zero:N \l__zrefclever_range_same_count_int
4426     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4427     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4428 }
4429 }

```

(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.

```

4430 \cs_new_protected:Npn \_zrefclever_typeset_refs_not_last_of_type:
4431 {
4432     % Signal if next label may form a range with the current one (only
4433     % considered if compression is enabled in the first place).
4434     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4435     \bool_set_false:N \l__zrefclever_next_is_same_bool
4436     \bool_if:NT \l__zrefclever_typeset_compress_bool
4437     {
4438         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4439         { }
4440         {
4441             \_zrefclever_labels_in_sequence:nn
4442             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4443     }
4444 }
4445
4446 % Process the current label to the current queue.
4447 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4448 {
4449     % Current label is the first of its type (also not the last, but it
4450     % doesn't matter here): just store the label.
4451     \tl_set:NV \l__zrefclever_type_first_label_tl
4452         \l__zrefclever_label_a_tl
4453     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4454         \l__zrefclever_label_type_a_tl
4455     \int_incr:N \l__zrefclever_ref_count_int
4456
4457     % If the next label may be part of a range, signal it (we deal with it
4458     % as the "first", and must do it there, to handle hyperlinking), but
4459     % also step the range counters.
4460     % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4461     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4462     {
4463         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4464         \tl_set:NV \l__zrefclever_range_beg_label_tl
4465             \l__zrefclever_label_a_tl
4466         \tl_clear:N \l__zrefclever_range_end_ref_tl
4467         \int_incr:N \l__zrefclever_range_count_int
4468         \bool_if:NT \l__zrefclever_next_is_same_bool
4469             { \int_incr:N \l__zrefclever_range_same_count_int }
4470     }
4471 }
4472 {
4473     % Current label is neither the first (nor the last) of its type.
4474     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4475     {
4476         % Starting, or continuing a range.
4477         \int_compare:nNnTF
4478             { \l__zrefclever_range_count_int } = { 0 }
4479         {
4480             % There was no range going, we are starting one.
4481             \tl_set:NV \l__zrefclever_range_beg_label_tl
4482                 \l__zrefclever_label_a_tl
4483             \tl_clear:N \l__zrefclever_range_end_ref_tl
4484             \int_incr:N \l__zrefclever_range_count_int
4485             \bool_if:NT \l__zrefclever_next_is_same_bool
4486                 { \int_incr:N \l__zrefclever_range_same_count_int }
4487         }
4488         {
4489             % Second or more in the range, but not the last.
4490             \int_incr:N \l__zrefclever_range_count_int
4491             \bool_if:NT \l__zrefclever_next_is_same_bool
4492                 { \int_incr:N \l__zrefclever_range_same_count_int }
4493         }
4494     }
4495 }
4496 % Next element is not in sequence: there was no range, or we are

```

```

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

```

```

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

```

```

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

```

```

4659             \_zrefclever_get_ref:VN \l_zrefclever_label_a_tl
4660             \l_zrefclever_refbounds_mid_re_seq
4661         }
4662     }
4663 }
4664 }
4665 % We just closed a range, reset 'range_beg_is_first' in case a
4666 % second range for the same type occurs, in which case its
4667 % 'range_beg' will no longer be 'first'.
4668 \bool_set_false:N \l_zrefclever_range_beg_is_first_bool
4669 % Reset counters.
4670 \int_zero:N \l_zrefclever_range_count_int
4671 \int_zero:N \l_zrefclever_range_same_count_int
4672 }
4673 }
4674 % Step label counter for next iteration.
4675 \int_incr:N \l_zrefclever_label_count_int
4676 }

```

(End definition for `_zrefclever_typeset_refs_not_last_of_type:.`)

Auxiliary functions

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

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

```

4677 \cs_new_protected:Npn \_zrefclever_ref_default:
4678 { \zref@default }
4679 \cs_new_protected:Npn \_zrefclever_name_default:
4680 { \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}}

4681 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4682 {
4683   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4684   {
4685     \bool_if:nTF
4686     {
4687       \l__zrefclever_hyperlink_bool &&
4688       ! \l__zrefclever_link_star_bool
4689     }
4690     {
4691       \exp_not:N \group_begin:
4692       \exp_not:V \l__zrefclever_reffont_tl
4693       \seq_item:Nn #2 { 1 }
4694       \__zrefclever_hyperlink:nnn
4695       { \__zrefclever_extract_url_unexp:n {#1} }
4696       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4697       {
4698         \seq_item:Nn #2 { 2 }
4699         \__zrefclever_extract_unexp:nvn {#1}
4700         { \l__zrefclever_ref_property_tl } { }
4701         \seq_item:Nn #2 { 3 }
4702       }
4703       \seq_item:Nn #2 { 4 }
4704       \exp_not:N \group_end:
4705     }
4706     {
4707       \exp_not:N \group_begin:
4708       \exp_not:V \l__zrefclever_reffont_tl
4709       \seq_item:Nn #2 { 1 }
4710       \seq_item:Nn #2 { 2 }
4711       \__zrefclever_extract_unexp:nvn {#1}
4712       { \l__zrefclever_ref_property_tl } { }
4713       \seq_item:Nn #2 { 3 }
4714       \seq_item:Nn #2 { 4 }
4715       \exp_not:N \group_end:
4716     }
4717   }
4718   { \__zrefclever_ref_default: }
4719 }
4720 \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}}

4721 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4722 {
4723   \str_if_eq:nnTF {#2} { zc@missingproperty }
4724   { \__zrefclever_ref_default: }
4725   {
4726     \bool_if:nTF
4727     {
4728       \l__zrefclever_hyperlink_bool &&
4729       ! \l__zrefclever_link_star_bool
4730     }
4731     {
4732       \exp_not:N \group_begin:
4733       \exp_not:V \l__zrefclever_reffont_tl
4734       \seq_item:Nn #3 { 1 }
4735       \__zrefclever_hyperlink:nnn
4736       { \__zrefclever_extract_url_unexp:n {#1} }
4737       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4738       {
4739         \seq_item:Nn #3 { 2 }
4740         \exp_not:n {#2}
4741         \seq_item:Nn #3 { 3 }
4742       }
4743       \seq_item:Nn #3 { 4 }
4744       \exp_not:N \group_end:
4745     }
4746     {
4747       \exp_not:N \group_begin:
4748       \exp_not:V \l__zrefclever_reffont_tl
4749       \seq_item:Nn #3 { 1 }
4750       \seq_item:Nn #3 { 2 }
4751       \exp_not:n {#2}
4752       \seq_item:Nn #3 { 3 }
4753       \seq_item:Nn #3 { 4 }
4754       \exp_not:N \group_end:
4755     }
4756   }
4757 }
4758 \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.

```

4759 \cs_new:Npn \__zrefclever_get_ref_first:
4760 {
4761   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4762   { \__zrefclever_ref_default: }
4763   {
4764     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4765 {
4766   \zref@ifrefcontainsprop
4767   { \l__zrefclever_type_first_label_tl }
4768   { \l__zrefclever_ref_property_tl }
4769   {
4770     \exp_not:N \group_begin:
4771     \__zrefclever_hyperlink:nnn
4772     {
4773       \__zrefclever_extract_url_unexp:V
4774       \l__zrefclever_type_first_label_tl
4775     }
4776     {
4777       \__zrefclever_extract_unexp:Vnn
4778       \l__zrefclever_type_first_label_tl { anchor } { }
4779     }
4780     {
4781       \exp_not:N \group_begin:
4782       \exp_not:V \l__zrefclever_namefont_tl
4783       \exp_not:V \l__zrefclever_type_name_tl
4784       \exp_not:N \group_end:
4785       \exp_not:V \l__zrefclever_namesep_tl
4786       \exp_not:N \group_begin:
4787       \exp_not:V \l__zrefclever_reffont_tl
4788       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4789       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4790       \__zrefclever_extract_unexp:Vnn
4791       \l__zrefclever_type_first_label_tl
4792       { \l__zrefclever_ref_property_tl } { }
4793       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4794       \exp_not:N \group_end:
4795     }
4796     \exp_not:V \l__zrefclever_reffont_tl
4797     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4798     \exp_not:N \group_end:
4799   }
4800   {
4801     \exp_not:N \group_begin:
4802     \exp_not:V \l__zrefclever_namefont_tl
4803     \exp_not:V \l__zrefclever_type_name_tl
4804     \exp_not:N \group_end:
4805     \exp_not:V \l__zrefclever_namesep_tl
4806     \__zrefclever_ref_default:
4807   }
4808 }
4809 {
4810   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4811   {
4812     \__zrefclever_name_default:
4813     \exp_not:V \l__zrefclever_namesep_tl
4814   }
4815   {
4816     \exp_not:N \group_begin:
4817     \exp_not:V \l__zrefclever_namefont_tl
4818     \exp_not:V \l__zrefclever_type_name_tl

```

```

4819         \exp_not:N \group_end:
4820         \tl_if_empty:NF \l__zrefclever_type_name_tl
4821         { \exp_not:V \l__zrefclever_namesep_tl }
4822     }
4823 \zref@ifrefcontainsprop
4824 { \l__zrefclever_type_first_label_tl }
4825 { \l__zrefclever_ref_property_tl }
4826 {
4827     \bool_if:nTF
4828     {
4829         \l__zrefclever_hyperlink_bool &&
4830         ! \l__zrefclever_link_star_bool
4831     }
4832     {
4833         \exp_not:N \group_begin:
4834         \exp_not:V \l__zrefclever_reffont_tl
4835         \seq_item:Nn
4836         \l__zrefclever_type_first_refbounds_seq { 1 }
4837         \__zrefclever_hyperlink:nnn
4838         {
4839             \__zrefclever_extract_url_unexp:V
4840             \l__zrefclever_type_first_label_tl
4841         }
4842         {
4843             \__zrefclever_extract_unexp:Vnn
4844             \l__zrefclever_type_first_label_tl { anchor } { }
4845         }
4846         {
4847             \seq_item:Nn
4848             \l__zrefclever_type_first_refbounds_seq { 2 }
4849             \__zrefclever_extract_unexp:Vvn
4850             \l__zrefclever_type_first_label_tl
4851             { \l__zrefclever_ref_property_tl } { }
4852             \seq_item:Nn
4853             \l__zrefclever_type_first_refbounds_seq { 3 }
4854         }
4855         \seq_item:Nn
4856         \l__zrefclever_type_first_refbounds_seq { 4 }
4857         \exp_not:N \group_end:
4858     }
4859     {
4860         \exp_not:N \group_begin:
4861         \exp_not:V \l__zrefclever_reffont_tl
4862         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4863         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4864         \__zrefclever_extract_unexp:Vvn
4865         \l__zrefclever_type_first_label_tl
4866         { \l__zrefclever_ref_property_tl } { }
4867         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4868         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4869         \exp_not:N \group_end:
4870     }
4871 }
4872 { \__zrefclever_ref_default: }

```

```

4873     }
4874   }
4875 }

```

(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`.

```

4876 \cs_new_protected:Npn \_zrefclever_type_name_setup:
4877 {
4878   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4879   {
4880     \tl_clear:N \l__zrefclever_type_name_tl
4881     \bool_set_true:N \l__zrefclever_type_name_missing_bool
4882   }
4883   {
4884     \tl_if_eq:NnTF
4885       \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4886       {
4887         \tl_clear:N \l__zrefclever_type_name_tl
4888         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4889       }
4890       {
4891         % Determine whether we should use capitalization, abbreviation,
4892         % and plural.
4893         \bool_lazy_or:nnTF
4894           { \l__zrefclever_cap_bool }
4895           {
4896             \l__zrefclever_capfirst_bool &&
4897             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4898           }
4899           { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4900           { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4901         % If the queue is empty, we have a singular, otherwise, plural.
4902         \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4903           { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4904           { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4905         \bool_lazy_and:nnTF
4906           { \l__zrefclever_abbrev_bool }
4907           {
4908             ! \int_compare_p:nNn
4909               { \l__zrefclever_type_count_int } = { 0 } ||
4910             ! \l__zrefclever_noabbrev_first_bool
4911           }

```

```

4912 {
4913   \tl_set:NV \l__zrefclever_name_format_fallback_tl
4914   \l__zrefclever_name_format_tl
4915   \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4916 }
4917 { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4918
4919 % Handle number and gender nudges.
4920 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4921 {
4922   \bool_if:NTF \l__zrefclever_nudge_singular_bool
4923   {
4924     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4925     {
4926       \msg_warning:nnx { zref-clever }
4927       { nudge-plural-when-sg }
4928       { \l__zrefclever_type_first_label_type_tl }
4929     }
4930   }
4931   {
4932     \bool_lazy_all:nT
4933     {
4934       { \l__zrefclever_nudge_comptosing_bool }
4935       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4936       {
4937         \int_compare_p:nNn
4938         { \l__zrefclever_label_count_int } > { 0 }
4939       }
4940     }
4941     {
4942       \msg_warning:nnx { zref-clever }
4943       { nudge-comptosing }
4944       { \l__zrefclever_type_first_label_type_tl }
4945     }
4946   }
4947   \bool_lazy_and:nnT
4948   { \l__zrefclever_nudge_gender_bool }
4949   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4950   {
4951     \__zrefclever_get_rf_opt_seq:nxxN { gender }
4952     { \l__zrefclever_type_first_label_type_tl }
4953     { \l__zrefclever_ref_language_tl }
4954     \l__zrefclever_type_name_gender_seq
4955     \seq_if_in:NVF
4956     \l__zrefclever_type_name_gender_seq
4957     \l__zrefclever_ref_gender_tl
4958     {
4959       \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4960       {
4961         \msg_warning:nxxxx { zref-clever }
4962         { nudge-gender-not-declared-for-type }
4963         { \l__zrefclever_ref_gender_tl }
4964         { \l__zrefclever_type_first_label_type_tl }
4965         { \l__zrefclever_ref_language_tl }

```

```

4966     }
4967     {
4968         \msg_warning:nnxxxx { zref-clever }
4969         { nudge-gender-mismatch }
4970         { \l__zrefclever_type_first_label_type_tl }
4971         { \l__zrefclever_ref_gender_tl }
4972         {
4973             \seq_use:Nn
4974             \l__zrefclever_type_name_gender_seq { ,~ }
4975         }
4976         { \l__zrefclever_ref_language_tl }
4977     }
4978 }
4979 }
4980 }
4981
4982 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
4983 {
4984     \__zrefclever_opt_tl_get:cNF
4985     {
4986         \__zrefclever_opt_varname_type:een
4987         { \l__zrefclever_type_first_label_type_tl }
4988         { \l__zrefclever_name_format_tl }
4989         { tl }
4990     }
4991     \l__zrefclever_type_name_tl
4992     {
4993         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
4994         {
4995             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
4996             \tl_put_left:NV \l__zrefclever_name_format_tl
4997             \l__zrefclever_ref_decl_case_tl
4998         }
4999         \__zrefclever_opt_tl_get:cNF
5000         {
5001             \__zrefclever_opt_varname_lang_type:eeen
5002             { \l__zrefclever_ref_language_tl }
5003             { \l__zrefclever_type_first_label_type_tl }
5004             { \l__zrefclever_name_format_tl }
5005             { tl }
5006         }
5007         \l__zrefclever_type_name_tl
5008         {
5009             \tl_clear:N \l__zrefclever_type_name_tl
5010             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5011             \msg_warning:nnxx { zref-clever } { missing-name }
5012             { \l__zrefclever_name_format_tl }
5013             { \l__zrefclever_type_first_label_type_tl }
5014         }
5015     }
5016 }
5017 {
5018     \__zrefclever_opt_tl_get:cNF
5019     {

```

```

5020 \_zrefclever_opt_varname_type:een
5021 { \l__zrefclever_type_first_label_type_tl }
5022 { \l__zrefclever_name_format_tl }
5023 { tl }
5024 }
5025 \l__zrefclever_type_name_tl
5026 {
5027 \_zrefclever_opt_tl_get:cNF
5028 {
5029 \_zrefclever_opt_varname_type:een
5030 { \l__zrefclever_type_first_label_type_tl }
5031 { \l__zrefclever_name_format_fallback_tl }
5032 { tl }
5033 }
5034 \l__zrefclever_type_name_tl
5035 {
5036 \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5037 {
5038 \tl_put_left:Nn
5039 \l__zrefclever_name_format_tl { - }
5040 \tl_put_left:NV \l__zrefclever_name_format_tl
5041 \l__zrefclever_ref_decl_case_tl
5042 \tl_put_left:Nn
5043 \l__zrefclever_name_format_fallback_tl { - }
5044 \tl_put_left:NV
5045 \l__zrefclever_name_format_fallback_tl
5046 \l__zrefclever_ref_decl_case_tl
5047 }
5048 \_zrefclever_opt_tl_get:cNF
5049 {
5050 \_zrefclever_opt_varname_lang_type:eeen
5051 { \l__zrefclever_ref_language_tl }
5052 { \l__zrefclever_type_first_label_type_tl }
5053 { \l__zrefclever_name_format_tl }
5054 { tl }
5055 }
5056 \l__zrefclever_type_name_tl
5057 {
5058 \_zrefclever_opt_tl_get:cNF
5059 {
5060 \_zrefclever_opt_varname_lang_type:eeen
5061 { \l__zrefclever_ref_language_tl }
5062 { \l__zrefclever_type_first_label_type_tl }
5063 { \l__zrefclever_name_format_fallback_tl }
5064 { tl }
5065 }
5066 \l__zrefclever_type_name_tl
5067 {
5068 \tl_clear:N \l__zrefclever_type_name_tl
5069 \bool_set_true:N
5070 \l__zrefclever_type_name_missing_bool
5071 \msg_warning:nxxx { zref-clever }
5072 { missing-name }
5073 { \l__zrefclever_name_format_tl }

```



```

5074                                     { \l__zrefclever_type_first_label_type_tl }
5075                                     }
5076                                 }
5077                            }
5078                       }
5079                   }
5080               }
5081           }
5082
5083       % Signal whether the type name is to be included in the hyperlink or not.
5084       \bool_lazy_any:nTF
5085       {
5086         { ! \l__zrefclever_hyperlink_bool }
5087         { \l__zrefclever_link_star_bool }
5088         { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5089         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5090       }
5091       { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5092       {
5093         \bool_lazy_any:nTF
5094         {
5095           { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5096           {
5097             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5098             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5099           }
5100           {
5101             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5102             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5103             \l__zrefclever_typeset_last_bool &&
5104             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5105           }
5106         }
5107         { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5108         { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5109       }
5110   }

```

(End definition for _zrefclever_type_name_setup:.)

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

```

\__zrefclever_hyperlink:nnn {<url/file>} {<anchor>} {<text>}
5111 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5112 {
5113   \tl_if_empty:nTF {#1}
5114     { \hyperlink {#2} {#3} }
5115     { \hyper@linkfile {#3} {#1} {#2} }
5116 }

```

(End definition for _zrefclever_hyperlink:nnn.)

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

```

5117 \cs_new:Npn \_zrefclever_extract_url_unexp:n #1
5118 {
5119   \zref@ifpropundefined { urluse }
5120   { \_zrefclever_extract_unexp:nnn {#1} { url } { } }
5121   {
5122     \zref@ifrefcontainsprop {#1} { urluse }
5123     { \_zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5124     { \_zrefclever_extract_unexp:nnn {#1} { url } { } }
5125   }
5126 }
5127 \cs_generate_variant:Nn \_zrefclever_extract_url_unexp:n { V }

```

(End definition for `_zrefclever_extract_url_unexp:n`.)

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

```

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

5128 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
5129 {
5130   \exp_args:Nxx \tl_if_eq:nnT
5131   { \_zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5132   { \_zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5133   {
5134     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5135     {
5136       \exp_args:Nxx \tl_if_eq:nnT
5137       { \_zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5138       { \_zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5139       {
5140         \int_compare:nNnTF
5141         { \_zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5142         =
5143         { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5144         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5145         {
5146           \int_compare:nNnT
5147           { \_zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5148           =
5149           { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5150           {
5151             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5152             \bool_set_true:N \l__zrefclever_next_is_same_bool
5153           }
5154         }
5155       }
5156     }
5157   }

```

```

5154     }
5155   }
5156 }
5157 {
5158   \exp_args:Nxx \tl_if_eq:nnT
5159   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5160   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5161   {
5162     \exp_args:Nxx \tl_if_eq:nnT
5163     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5164     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5165     {
5166       \int_compare:nNnTF
5167       { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5168       =
5169       { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5170       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5171       {
5172         \int_compare:nNnTF
5173         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5174         =
5175         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5176         {

```

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.

```

5177   \exp_args:Nxx \tl_if_eq:nnT
5178   {
5179     \__zrefclever_extract_unexp:nvn {#1}
5180     { l__zrefclever_ref_property_tl } { }
5181   }
5182   {
5183     \__zrefclever_extract_unexp:nvn {#2}
5184     { l__zrefclever_ref_property_tl } { }
5185   }
5186   {
5187     \bool_set_true:N
5188     \l__zrefclever_next_maybe_range_bool
5189     \bool_set_true:N
5190     \l__zrefclever_next_is_same_bool
5191   }
5192 }
5193 }
5194 }
5195 }
5196 }
5197 }
5198 }

```

(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>}
5199 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5200 {
5201   % First attempt: general options.
5202   \__zrefclever_opt_tl_get:cNF
5203   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5204   #4
5205   {
5206     % If not found, try type specific options.
5207     \__zrefclever_opt_tl_get:cNF
5208     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5209     #4
5210     {
5211       % If not found, try type- and language-specific.
5212       \__zrefclever_opt_tl_get:cNF
5213       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5214       #4
5215       {
5216         % If not found, try language-specific default.
5217         \__zrefclever_opt_tl_get:cNF
5218         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5219         #4
5220         {
5221           % If not found, try fallback.
5222           \__zrefclever_opt_tl_get:cNF
5223           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5224           #4
5225           { \tl_clear:N #4 }
5226         }
5227       }
5228     }
5229   }
5230 }
5231 \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>}
5232 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5233 {
5234   % First attempt: general options.
5235   \__zrefclever_opt_seq_get:cNF
5236   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5237   #4
5238   {
5239     % If not found, try type specific options.

```

```

5240 \__zrefclever_opt_seq_get:cNF
5241 { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5242 #4
5243 {
5244   % If not found, try type- and language-specific.
5245   \__zrefclever_opt_seq_get:cNF
5246   { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5247   #4
5248   {
5249     % If not found, try language-specific default.
5250     \__zrefclever_opt_seq_get:cNF
5251     { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5252     #4
5253     {
5254       % If not found, try fallback.
5255       \__zrefclever_opt_seq_get:cNF
5256       { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5257       #4
5258       { \seq_clear:N #4 }
5259     }
5260   }
5261 }
5262 }
5263 }
5264 \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>}
5265 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5266 {
5267   % First attempt: general options.
5268   \__zrefclever_opt_bool_get:cNF
5269   { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5270   #5
5271   {
5272     % If not found, try type specific options.
5273     \__zrefclever_opt_bool_get:cNF
5274     { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5275     #5
5276     {
5277       % If not found, try type- and language-specific.
5278       \__zrefclever_opt_bool_get:cNF
5279       { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5280       #5
5281       {
5282         % If not found, try language-specific default.
5283         \__zrefclever_opt_bool_get:cNF
5284         { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5285         #5
5286         {
5287           % If not found, try fallback.
5288           \__zrefclever_opt_bool_get:cNF

```

```

5289             { \_zrefclever_opt_varname_fallback:nn {#1} { bool } }
5290             #5
5291             { \use:c { bool_set_ #2 :N } #5 }
5292         }
5293     }
5294 }
5295 }
5296 }
5297 \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.

```

5298 \_zrefclever_compat_module:nn { appendix }
5299 {
5300     \AddToHook { cmd / appendix / before }
5301     {
5302         \_zrefclever_zcsetup:n
5303         {
5304             countertype =
5305             {
5306                 chapter      = appendix ,
5307                 section      = appendix ,
5308                 subsection   = appendix ,
5309                 subsubsection = appendix ,
5310                 paragraph    = appendix ,
5311                 subparagraph = appendix ,
5312             }
5313         }
5314     }
5315 }

```

```

5314     }
5315 }

```

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

9.2 appendices

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

```

5316 \__zrefclever_compat_module:nn { appendices }
5317 {
5318   \__zrefclever_if_package_loaded:nT { appendix }
5319   {
5320     \newcounter { zc@appendix }
5321     \newcounter { zc@save@appendix }
5322     \setcounter { zc@appendix } { 0 }
5323     \setcounter { zc@save@appendix } { 0 }
5324     \cs_if_exist:cTF { chapter }
5325     {
5326       \__zrefclever_zcsetup:n
5327       { counterresetby = { chapter = zc@appendix } }
5328     }
5329     {
5330       \cs_if_exist:cT { section }
5331       {
5332         \__zrefclever_zcsetup:n
5333         { counterresetby = { section = zc@appendix } }
5334       }
5335     }
5336   }
5337   \AddToHook { env / appendices / begin }
5338   {
5339     \stepcounter { zc@save@appendix }
5340     \setcounter { zc@appendix } { \value { zc@save@appendix } }
5341     \__zrefclever_zcsetup:n
5342     {
5343       countertype =
5344       {
5345         chapter      = appendix ,
5346         section      = appendix ,
5347         subsection   = appendix ,
5348         subsubsection = appendix ,
5349         paragraph    = appendix ,
5350         subparagraph = appendix ,
5351       }
5352     }
5353   }
5354 }

```

```

5353 \AddToHook { env / appendices / end }
5354 { \setcounter { zc@appendix } { 0 } }
5355 \AddToHook { cmd / appendix / before }
5356 {
5357   \stepcounter { zc@save@appendix }
5358   \setcounter { zc@appendix } { \value { zc@save@appendix } }
5359 }
5360 \AddToHook { env / subappendices / begin }
5361 {
5362   \__zrefclever_zcsetup:n
5363   {
5364     countertype =
5365     {
5366       section      = appendix ,
5367       subsection   = appendix ,
5368       subsubsection = appendix ,
5369       paragraph    = appendix ,
5370       subparagraph = appendix ,
5371     } ,
5372   }
5373 }
5374 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5375 }
5376 }

```

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.

```

5377 \__zrefclever_compat_module:nn { memoir }
5378 {
5379   \__zrefclever_if_class_loaded:nT { memoir }
5380   {

```

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`.

```

5381   \__zrefclever_zcsetup:n
5382   {
5383     countertype =
5384     {
5385       subfigure = figure ,
5386       subtable  = table ,
5387       poemline  = line ,
5388     } ,
5389     counterresetby =
5390     {
5391       subfigure = figure ,
5392       subtable  = table ,
5393     } ,
5394   }

```

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

```

5395 \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
5396 {
5397   \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
5398   \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
5399   {
5400     \__zrefclever_memoir_orig_label:n {##1}
5401     \zlabel{##1}
5402   }
5403   \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
5404 }
5405 \AddToHook { env / sidecaption / begin }
5406 { \__zrefclever_memoir_both_labels: }
5407 \AddToHook { env / sidecontcaption / begin }
5408 { \__zrefclever_memoir_both_labels: }
5409 \AddToHook{ cmd / bitwonumcaption / before }
5410 { \group_begin: \__zrefclever_memoir_both_labels: }
5411 \AddToHook{ cmd / bitwonumcaption / after }
5412 { \group_end: }
5413 \AddToHook{ cmd / bionenumcaption / before }
5414 { \group_begin: \__zrefclever_memoir_both_labels: }
5415 \AddToHook{ cmd / bionenumcaption / after }
5416 { \group_end: }
5417 \AddToHook{ cmd / bicaption / before }
5418 { \group_begin: \__zrefclever_memoir_both_labels: }
5419 \AddToHook{ cmd / bicaption / after }
5420 { \group_end: }

```

Support for subcaption reference.

```

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

```

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

```

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

```

```

5446         \msg_info:nnn { zref-clever } { compat-class } { memoir }
5447     }
5448 }

```

9.4 KOMA

Support for KOMA-Script document classes.

```

5449 \__zrefclever_compat_module:nn { KOMA }
5450 {
5451     \cs_if_exist:NT \KOMAClassName
5452     {

```

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.

```

5453     \AddToHook { env / captionbeside / after }
5454     {
5455         \tl_if_exist:NT \@capttype
5456         { \tl_set_eq:NN \@currentcounter \@capttype }
5457     }
5458     \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
5459     \AddToHook { env / captionofbeside / end }
5460     { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
5461     \AddToHook { env / captionofbeside / after }
5462     {
5463         \tl_if_eq:NnF \@currentenvir { document }
5464         {
5465             \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
5466             {
5467                 \tl_set_eq:NN
5468                 \@currentcounter \g__zrefclever_koma_capttype_tl
5469             }
5470         }
5471         \tl_gclear:N \g__zrefclever_koma_capttype_tl
5472     }
5473     \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
5474 }
5475 }

```

9.5 amsmath

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

```

5476 \__zrefclever_compat_module:nn { amsmath }
5477 {

```

```

5478     \__zrefclever_if_package_loaded:nT { amsmath }
5479     {

```

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...).

```

5480         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
5481         {
5482             \__zrefclever_orig_ltxlabel:n {#1}
5483             \zref@wrapper@babel \zref@label {#1}
5484         }

```

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>.

```

5485     \__zrefclever_if_package_loaded:nTF { hyperref }
5486     {
5487         \AddToHook { package / nameref / after }
5488         {
5489             \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5490             \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5491         }
5492     }
5493     {
5494         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
5495         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
5496     }

```

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).

```

5497     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5498     \AddToHook { env / subequations / begin }
5499     {
5500         \__zrefclever_zcsetup:x
5501         {
5502             counterresetby =
5503             {
5504                 parentequation =
5505                 \__zrefclever_counter_reset_by:n { equation } ,
5506                 equation = parentequation ,
5507             } ,
5508             currentcounter = parentequation ,
5509             countertype = { parentequation = equation } ,
5510         }
5511     \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5512 }

```

`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.

```

5513     \zref@newprop { subeq } { \alph { equation } }
5514     \clist_map_inline:nn
5515     {
5516         equation ,
5517         equation* ,
5518         align ,
5519         align* ,
5520         alignat ,
5521         alignat* ,
5522         flalign ,
5523         flalign* ,
5524         xalignat ,
5525         xalignat* ,
5526         gather ,
5527         gather* ,

```

```

5528         multiline ,
5529         multiline* ,
5530     }
5531     {
5532         \AddToHook { env / #1 / begin }
5533         {
5534             \__zrefclever_zcsetup:n { currentcounter = equation }
5535             \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5536                 { \zref@localaddprop \ZREF@mainlist { subeq } }
5537         }
5538     }

```

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

```

5539     \zcRefTypeSetup { equation }
5540     { reffont = \upshape }
5541     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5542 }
5543 }

```

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.

```

5544 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5545 \__zrefclever_compat_module:nn { mathtools }
5546 {
5547     \__zrefclever_if_package_loaded:nT { mathtools }
5548     {
5549         \MH_if_boolean:nT { show_only_refs }
5550         {
5551             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5552             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5553             {
5554                 \@bsphack
5555                 \seq_map_inline:Nn #1
5556                 {
5557                     \exp_args:Nx \tl_if_eq:nnTF
5558                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5559                     { equation }
5560                     {
5561                         \protected@write \@auxout { }
5562                         { \string \MT@newlabel {##1} }
5563                     }
5564                 }
5565                 \exp_args:Nx \tl_if_eq:nnT

```

```

5566         { \_zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5567         { parentequation }
5568         {
5569             \protected@write \@auxout { }
5570             { \string \MT@newlabel {##1} }
5571         }
5572     }
5573 }
5574 \@esphack
5575 }
5576 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5577 }
5578 }
5579 }

```

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”.

```

5580 \_zrefclever_compat_module:nn { breqn }
5581 {
5582     \_zrefclever_if_package_loaded:nT { breqn }
5583     {

```

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>).

```

5584     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5585     \AddToHook { env / dgroup / begin }
5586     {
5587         \_zrefclever_zcsetup:x
5588         {
5589             counterresetby =
5590             {
5591                 parentequation =
5592                 \_zrefclever_counter_reset_by:n { equation } ,
5593                 equation = parentequation ,
5594             } ,
5595             currentcounter = parentequation ,
5596             countertype = { parentequation = equation } ,
5597         }
5598         \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5599     }
5600 \zref@ifpropundefined { subeq }
5601 { \zref@newprop { subeq } { \alph { equation } } }
5602 { }

```

```

5603     \clist_map_inline:nn
5604     {
5605         dmath ,
5606         dseries ,
5607         darray ,
5608     }
5609     {
5610         \AddToHook { env / #1 / begin }
5611         {
5612             \__zrefclever_zcsetup:n { currentcounter = equation }
5613             \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5614             { \zref@localaddprop \ZREF@mainlist { subeq } }
5615         }
5616     }
5617     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5618 }
5619 }

```

9.8 listings

```

5620 \__zrefclever_compat_module:nn { listings }
5621 {
5622     \__zrefclever_if_package_loaded:nT { listings }
5623     {
5624         \__zrefclever_zcsetup:n
5625         {
5626             countertype =
5627             {
5628                 lstlisting = listing ,
5629                 lstnumber = line ,
5630             } ,
5631             counterresetby = { lstnumber = lstlisting } ,
5632         }
5633     }
5634 }

```

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.

```

5633     \lst@AddToHook { PreInit }
5634     { \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.

```

5635     \lst@AddToHook { Init }
5636     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }

```



```

5637         \msg_info:nnn { zref-clever } { compat-package } { listings }
5638     }
5639 }

```

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 $\langle \max\text{-depth} \rangle$. `\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.

```

5640 \__zrefclever_compat_module:nn { enumitem }
5641 {
5642     \__zrefclever_if_package_loaded:nT { enumitem }
5643     {
5644         \int_set:Nn \l_tmpa_int { 5 }
5645         \bool_while_do:nn
5646         {
5647             \cs_if_exist_p:c
5648             { c@ enum \int_to_roman:n { \l_tmpa_int } }
5649         }
5650         {
5651             \__zrefclever_zcsetup:x
5652             {
5653                 counterresetby =
5654                 {
5655                     enum \int_to_roman:n { \l_tmpa_int } =
5656                     enum \int_to_roman:n { \l_tmpa_int - 1 }
5657                 } ,
5658                 countertype =
5659                 { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5660             }
5661             \int_incr:N \l_tmpa_int
5662         }
5663         \int_compare:nNnT { \l_tmpa_int } > { 5 }
5664         { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5665     }
5666 }

```

9.10 subcaption

```

5667 \__zrefclever_compat_module:nn { subcaption }
5668 {
5669     \__zrefclever_if_package_loaded:nT { subcaption }
5670     {

```

```

5671 \__zrefclever_zcsetup:n
5672 {
5673     countertype =
5674     {
5675         subfigure = figure ,
5676         subtable = table ,
5677     } ,
5678     counterresetby =
5679     {
5680         subfigure = figure ,
5681         subtable = table ,
5682     } ,
5683 }

```

Support for `subref` reference.

```

5684 \zref@newprop { subref }
5685 { \cs_if_exist_use:c { thesub \@capttype } }
5686 \tl_put_right:Nn \caption@subtypetoolhook
5687 { \zref@localaddprop \ZREF@mainlist { subref } }
5688 }
5689 }

```

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.

```

5690 \__zrefclever_compat_module:nn { subfig }
5691 {
5692     \__zrefclever_if_package_loaded:nT { subfig }
5693     {
5694         \__zrefclever_zcsetup:n
5695         {
5696             countertype =
5697             {
5698                 subfigure = figure ,
5699                 subtable = table ,
5700             } ,
5701             counterresetby =
5702             {
5703                 subfigure = figure ,
5704                 subtable = table ,
5705             } ,
5706         }
5707     }
5708 }
5709 </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.

```
5710 <*package>
5711 \zcDeclareLanguage { english }
5712 \zcDeclareLanguageAlias { american } { english }
5713 \zcDeclareLanguageAlias { australian } { english }
5714 \zcDeclareLanguageAlias { british } { english }
5715 \zcDeclareLanguageAlias { canadian } { english }
5716 \zcDeclareLanguageAlias { newzealand } { english }
5717 \zcDeclareLanguageAlias { UKenglish } { english }
5718 \zcDeclareLanguageAlias { USenglish } { english }
5719 </package>

5720 <*lang-english>

5721 namesep = {\nobreakspace} ,
5722 pairsep = {\~and\nobreakspace} ,
5723 listsep = {\~,~} ,
5724 lastsep = {\~and\nobreakspace} ,
5725 tpairsep = {\~and\nobreakspace} ,
5726 tlistsep = {\~,~} ,
5727 tlastsep = {\~,~and\nobreakspace} ,
5728 notesep = {\~} ,
5729 rangesep = {\~to\nobreakspace} ,
5730
5731 type = book ,
5732   Name-sg = Book ,
5733   name-sg = book ,
5734   Name-pl = Books ,
5735   name-pl = books ,
5736
5737 type = part ,
5738   Name-sg = Part ,
5739   name-sg = part ,
5740   Name-pl = Parts ,
5741   name-pl = parts ,
5742
5743 type = chapter ,
5744   Name-sg = Chapter ,
5745   name-sg = chapter ,
5746   Name-pl = Chapters ,
5747   name-pl = chapters ,
5748
5749 type = section ,
5750   Name-sg = Section ,
5751   name-sg = section ,
5752   Name-pl = Sections ,
5753   name-pl = sections ,
5754
5755 type = paragraph ,
5756   Name-sg = Paragraph ,
5757   name-sg = paragraph ,
5758   Name-pl = Paragraphs ,
5759   name-pl = paragraphs ,
```

```

5760 Name-sg-ab = Par. ,
5761 name-sg-ab = par. ,
5762 Name-pl-ab = Par. ,
5763 name-pl-ab = par. ,
5764
5765 type = appendix ,
5766 Name-sg = Appendix ,
5767 name-sg = appendix ,
5768 Name-pl = Appendices ,
5769 name-pl = appendices ,
5770
5771 type = page ,
5772 Name-sg = Page ,
5773 name-sg = page ,
5774 Name-pl = Pages ,
5775 name-pl = pages ,
5776 rangsep = {\textendash} ,
5777 rangetopair = false ,
5778
5779 type = line ,
5780 Name-sg = Line ,
5781 name-sg = line ,
5782 Name-pl = Lines ,
5783 name-pl = lines ,
5784
5785 type = figure ,
5786 Name-sg = Figure ,
5787 name-sg = figure ,
5788 Name-pl = Figures ,
5789 name-pl = figures ,
5790 Name-sg-ab = Fig. ,
5791 name-sg-ab = fig. ,
5792 Name-pl-ab = Figs. ,
5793 name-pl-ab = figs. ,
5794
5795 type = table ,
5796 Name-sg = Table ,
5797 name-sg = table ,
5798 Name-pl = Tables ,
5799 name-pl = tables ,
5800
5801 type = item ,
5802 Name-sg = Item ,
5803 name-sg = item ,
5804 Name-pl = Items ,
5805 name-pl = items ,
5806
5807 type = footnote ,
5808 Name-sg = Footnote ,
5809 name-sg = footnote ,
5810 Name-pl = Footnotes ,
5811 name-pl = footnotes ,
5812
5813 type = endnote ,

```

```

5814 Name-sg = Note ,
5815 name-sg = note ,
5816 Name-pl = Notes ,
5817 name-pl = notes ,
5818
5819 type = note ,
5820 Name-sg = Note ,
5821 name-sg = note ,
5822 Name-pl = Notes ,
5823 name-pl = notes ,
5824
5825 type = equation ,
5826 Name-sg = Equation ,
5827 name-sg = equation ,
5828 Name-pl = Equations ,
5829 name-pl = equations ,
5830 Name-sg-ab = Eq. ,
5831 name-sg-ab = eq. ,
5832 Name-pl-ab = Eqs. ,
5833 name-pl-ab = eqs. ,
5834 refbounds-first-sg = {,(,),} ,
5835 refbounds = {(,,)} ,
5836
5837 type = theorem ,
5838 Name-sg = Theorem ,
5839 name-sg = theorem ,
5840 Name-pl = Theorems ,
5841 name-pl = theorems ,
5842
5843 type = lemma ,
5844 Name-sg = Lemma ,
5845 name-sg = lemma ,
5846 Name-pl = Lemmas ,
5847 name-pl = lemmas ,
5848
5849 type = corollary ,
5850 Name-sg = Corollary ,
5851 name-sg = corollary ,
5852 Name-pl = Corollaries ,
5853 name-pl = corollaries ,
5854
5855 type = proposition ,
5856 Name-sg = Proposition ,
5857 name-sg = proposition ,
5858 Name-pl = Propositions ,
5859 name-pl = propositions ,
5860
5861 type = definition ,
5862 Name-sg = Definition ,
5863 name-sg = definition ,
5864 Name-pl = Definitions ,
5865 name-pl = definitions ,
5866
5867 type = proof ,

```

```

5868   Name-sg = Proof ,
5869   name-sg = proof ,
5870   Name-pl = Proofs ,
5871   name-pl = proofs ,
5872
5873   type = result ,
5874   Name-sg = Result ,
5875   name-sg = result ,
5876   Name-pl = Results ,
5877   name-pl = results ,
5878
5879   type = remark ,
5880   Name-sg = Remark ,
5881   name-sg = remark ,
5882   Name-pl = Remarks ,
5883   name-pl = remarks ,
5884
5885   type = example ,
5886   Name-sg = Example ,
5887   name-sg = example ,
5888   Name-pl = Examples ,
5889   name-pl = examples ,
5890
5891   type = algorithm ,
5892   Name-sg = Algorithm ,
5893   name-sg = algorithm ,
5894   Name-pl = Algorithms ,
5895   name-pl = algorithms ,
5896
5897   type = listing ,
5898   Name-sg = Listing ,
5899   name-sg = listing ,
5900   Name-pl = Listings ,
5901   name-pl = listings ,
5902
5903   type = exercise ,
5904   Name-sg = Exercise ,
5905   name-sg = exercise ,
5906   Name-pl = Exercises ,
5907   name-pl = exercises ,
5908
5909   type = solution ,
5910   Name-sg = Solution ,
5911   name-sg = solution ,
5912   Name-pl = Solutions ,
5913   name-pl = solutions ,
5914 </lang-english>

```

10.2 German

German language file has been initially provided by the author.

```

5915 <*package>
5916 \zcDeclareLanguage

```

```

5917 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5918 { german }
5919 \zcDeclareLanguageAlias { austrian      } { german }
5920 \zcDeclareLanguageAlias { germanb       } { german }
5921 \zcDeclareLanguageAlias { ngerman       } { german }
5922 \zcDeclareLanguageAlias { naustrian     } { german }
5923 \zcDeclareLanguageAlias { nswissgerman } { german }
5924 \zcDeclareLanguageAlias { swissgerman  } { german }
5925 \end{package}
5926 \deflang-german

5927 namesep = {\nobreakspace} ,
5928 pairsep = {\simund\nobreakspace} ,
5929 listsep = { , \sim } ,
5930 lastsep = {\simund\nobreakspace} ,
5931 tpairsep = {\simund\nobreakspace} ,
5932 tlistsep = { , \sim } ,
5933 tlastsep = {\simund\nobreakspace} ,
5934 notesep = { \sim } ,
5935 rangesep = {\simbis\nobreakspace} ,
5936
5937 type = book ,
5938 gender = n ,
5939 case = N ,
5940   Name-sg = Buch ,
5941   Name-pl = Bücher ,
5942 case = A ,
5943   Name-sg = Buch ,
5944   Name-pl = Bücher ,
5945 case = D ,
5946   Name-sg = Buch ,
5947   Name-pl = Büchern ,
5948 case = G ,
5949   Name-sg = Buches ,
5950   Name-pl = Bücher ,
5951
5952 type = part ,
5953 gender = m ,
5954 case = N ,
5955   Name-sg = Teil ,
5956   Name-pl = Teile ,
5957 case = A ,
5958   Name-sg = Teil ,
5959   Name-pl = Teile ,
5960 case = D ,
5961   Name-sg = Teil ,
5962   Name-pl = Teilen ,
5963 case = G ,
5964   Name-sg = Teiles ,
5965   Name-pl = Teile ,
5966
5967 type = chapter ,
5968 gender = n ,
5969 case = N ,

```

```

5970     Name-sg = Kapitel ,
5971     Name-pl = Kapitel ,
5972     case = A ,
5973     Name-sg = Kapitel ,
5974     Name-pl = Kapitel ,
5975     case = D ,
5976     Name-sg = Kapitel ,
5977     Name-pl = Kapiteln ,
5978     case = G ,
5979     Name-sg = Kapiteln ,
5980     Name-pl = Kapitel ,
5981
5982 type = section ,
5983     gender = m ,
5984     case = N ,
5985     Name-sg = Abschnitt ,
5986     Name-pl = Abschnitte ,
5987     case = A ,
5988     Name-sg = Abschnitt ,
5989     Name-pl = Abschnitte ,
5990     case = D ,
5991     Name-sg = Abschnitt ,
5992     Name-pl = Abschnitten ,
5993     case = G ,
5994     Name-sg = Abschnitts ,
5995     Name-pl = Abschnitte ,
5996
5997 type = paragraph ,
5998     gender = m ,
5999     case = N ,
6000     Name-sg = Absatz ,
6001     Name-pl = Absätze ,
6002     case = A ,
6003     Name-sg = Absatz ,
6004     Name-pl = Absätze ,
6005     case = D ,
6006     Name-sg = Absatz ,
6007     Name-pl = Absätzen ,
6008     case = G ,
6009     Name-sg = Absatzes ,
6010     Name-pl = Absätze ,
6011
6012 type = appendix ,
6013     gender = m ,
6014     case = N ,
6015     Name-sg = Anhang ,
6016     Name-pl = Anhänge ,
6017     case = A ,
6018     Name-sg = Anhang ,
6019     Name-pl = Anhänge ,
6020     case = D ,
6021     Name-sg = Anhang ,
6022     Name-pl = Anhängen ,
6023     case = G ,

```



```

6024     Name-sg = Anhangs ,
6025     Name-pl = Anhänge ,
6026
6027 type = page ,
6028     gender = f ,
6029     case = N ,
6030     Name-sg = Seite ,
6031     Name-pl = Seiten ,
6032     case = A ,
6033     Name-sg = Seite ,
6034     Name-pl = Seiten ,
6035     case = D ,
6036     Name-sg = Seite ,
6037     Name-pl = Seiten ,
6038     case = G ,
6039     Name-sg = Seite ,
6040     Name-pl = Seiten ,
6041     rangesep = {\textendash} ,
6042     rangetopair = false ,
6043
6044 type = line ,
6045     gender = f ,
6046     case = N ,
6047     Name-sg = Zeile ,
6048     Name-pl = Zeilen ,
6049     case = A ,
6050     Name-sg = Zeile ,
6051     Name-pl = Zeilen ,
6052     case = D ,
6053     Name-sg = Zeile ,
6054     Name-pl = Zeilen ,
6055     case = G ,
6056     Name-sg = Zeile ,
6057     Name-pl = Zeilen ,
6058
6059 type = figure ,
6060     gender = f ,
6061     case = N ,
6062     Name-sg = Abbildung ,
6063     Name-pl = Abbildungen ,
6064     Name-sg-ab = Abb. ,
6065     Name-pl-ab = Abb. ,
6066     case = A ,
6067     Name-sg = Abbildung ,
6068     Name-pl = Abbildungen ,
6069     Name-sg-ab = Abb. ,
6070     Name-pl-ab = Abb. ,
6071     case = D ,
6072     Name-sg = Abbildung ,
6073     Name-pl = Abbildungen ,
6074     Name-sg-ab = Abb. ,
6075     Name-pl-ab = Abb. ,
6076     case = G ,
6077     Name-sg = Abbildung ,

```

```

6078     Name-pl = Abbildungen ,
6079     Name-sg-ab = Abb. ,
6080     Name-pl-ab = Abb. ,
6081
6082 type = table ,
6083     gender = f ,
6084     case = N ,
6085     Name-sg = Tabelle ,
6086     Name-pl = Tabellen ,
6087     case = A ,
6088     Name-sg = Tabelle ,
6089     Name-pl = Tabellen ,
6090     case = D ,
6091     Name-sg = Tabelle ,
6092     Name-pl = Tabellen ,
6093     case = G ,
6094     Name-sg = Tabelle ,
6095     Name-pl = Tabellen ,
6096
6097 type = item ,
6098     gender = m ,
6099     case = N ,
6100     Name-sg = Punkt ,
6101     Name-pl = Punkte ,
6102     case = A ,
6103     Name-sg = Punkt ,
6104     Name-pl = Punkte ,
6105     case = D ,
6106     Name-sg = Punkt ,
6107     Name-pl = Punkten ,
6108     case = G ,
6109     Name-sg = Punktes ,
6110     Name-pl = Punkte ,
6111
6112 type = footnote ,
6113     gender = f ,
6114     case = N ,
6115     Name-sg = Fußnote ,
6116     Name-pl = Fußnoten ,
6117     case = A ,
6118     Name-sg = Fußnote ,
6119     Name-pl = Fußnoten ,
6120     case = D ,
6121     Name-sg = Fußnote ,
6122     Name-pl = Fußnoten ,
6123     case = G ,
6124     Name-sg = Fußnote ,
6125     Name-pl = Fußnoten ,
6126
6127 type = endnote ,
6128     gender = f ,
6129     case = N ,
6130     Name-sg = Endnote ,
6131     Name-pl = Endnoten ,

```

```

6132 case = A ,
6133     Name-sg = Endnote ,
6134     Name-pl = Endnoten ,
6135 case = D ,
6136     Name-sg = Endnote ,
6137     Name-pl = Endnoten ,
6138 case = G ,
6139     Name-sg = Endnote ,
6140     Name-pl = Endnoten ,
6141
6142 type = note ,
6143     gender = f ,
6144     case = N ,
6145         Name-sg = Anmerkung ,
6146         Name-pl = Anmerkungen ,
6147     case = A ,
6148         Name-sg = Anmerkung ,
6149         Name-pl = Anmerkungen ,
6150     case = D ,
6151         Name-sg = Anmerkung ,
6152         Name-pl = Anmerkungen ,
6153     case = G ,
6154         Name-sg = Anmerkung ,
6155         Name-pl = Anmerkungen ,
6156
6157 type = equation ,
6158     gender = f ,
6159     case = N ,
6160         Name-sg = Gleichung ,
6161         Name-pl = Gleichungen ,
6162     case = A ,
6163         Name-sg = Gleichung ,
6164         Name-pl = Gleichungen ,
6165     case = D ,
6166         Name-sg = Gleichung ,
6167         Name-pl = Gleichungen ,
6168     case = G ,
6169         Name-sg = Gleichung ,
6170         Name-pl = Gleichungen ,
6171     refbounds-first-sg = {,(,),} ,
6172     refbounds = {(,,)} ,
6173
6174 type = theorem ,
6175     gender = n ,
6176     case = N ,
6177         Name-sg = Theorem ,
6178         Name-pl = Theoreme ,
6179     case = A ,
6180         Name-sg = Theorem ,
6181         Name-pl = Theoreme ,
6182     case = D ,
6183         Name-sg = Theorem ,
6184         Name-pl = Theoremen ,
6185     case = G ,

```

```

6186     Name-sg = Theorems ,
6187     Name-pl = Theoreme ,
6188
6189 type = lemma ,
6190     gender = n ,
6191     case = N ,
6192     Name-sg = Lemma ,
6193     Name-pl = Lemmata ,
6194     case = A ,
6195     Name-sg = Lemma ,
6196     Name-pl = Lemmata ,
6197     case = D ,
6198     Name-sg = Lemma ,
6199     Name-pl = Lemmata ,
6200     case = G ,
6201     Name-sg = Lemmas ,
6202     Name-pl = Lemmata ,
6203
6204 type = corollary ,
6205     gender = n ,
6206     case = N ,
6207     Name-sg = Korollar ,
6208     Name-pl = Korollare ,
6209     case = A ,
6210     Name-sg = Korollar ,
6211     Name-pl = Korollare ,
6212     case = D ,
6213     Name-sg = Korollar ,
6214     Name-pl = Korollaren ,
6215     case = G ,
6216     Name-sg = Korollars ,
6217     Name-pl = Korollare ,
6218
6219 type = proposition ,
6220     gender = m ,
6221     case = N ,
6222     Name-sg = Satz ,
6223     Name-pl = Sätze ,
6224     case = A ,
6225     Name-sg = Satz ,
6226     Name-pl = Sätze ,
6227     case = D ,
6228     Name-sg = Satz ,
6229     Name-pl = Sätzen ,
6230     case = G ,
6231     Name-sg = Satzes ,
6232     Name-pl = Sätze ,
6233
6234 type = definition ,
6235     gender = f ,
6236     case = N ,
6237     Name-sg = Definition ,
6238     Name-pl = Definitionen ,
6239     case = A ,

```

```

6240     Name-sg = Definition ,
6241     Name-pl = Definitionen ,
6242     case = D ,
6243     Name-sg = Definition ,
6244     Name-pl = Definitionen ,
6245     case = G ,
6246     Name-sg = Definition ,
6247     Name-pl = Definitionen ,
6248
6249     type = proof ,
6250     gender = m ,
6251     case = N ,
6252     Name-sg = Beweis ,
6253     Name-pl = Beweise ,
6254     case = A ,
6255     Name-sg = Beweis ,
6256     Name-pl = Beweise ,
6257     case = D ,
6258     Name-sg = Beweis ,
6259     Name-pl = Beweisen ,
6260     case = G ,
6261     Name-sg = Beweises ,
6262     Name-pl = Beweise ,
6263
6264     type = result ,
6265     gender = n ,
6266     case = N ,
6267     Name-sg = Ergebnis ,
6268     Name-pl = Ergebnisse ,
6269     case = A ,
6270     Name-sg = Ergebnis ,
6271     Name-pl = Ergebnisse ,
6272     case = D ,
6273     Name-sg = Ergebnis ,
6274     Name-pl = Ergebnissen ,
6275     case = G ,
6276     Name-sg = Ergebnisses ,
6277     Name-pl = Ergebnisse ,
6278
6279     type = remark ,
6280     gender = f ,
6281     case = N ,
6282     Name-sg = Bemerkung ,
6283     Name-pl = Bemerkungen ,
6284     case = A ,
6285     Name-sg = Bemerkung ,
6286     Name-pl = Bemerkungen ,
6287     case = D ,
6288     Name-sg = Bemerkung ,
6289     Name-pl = Bemerkungen ,
6290     case = G ,
6291     Name-sg = Bemerkung ,
6292     Name-pl = Bemerkungen ,
6293

```

```

6294 type = example ,
6295     gender = n ,
6296     case = N ,
6297         Name-sg = Beispiel ,
6298         Name-pl = Beispiele ,
6299     case = A ,
6300         Name-sg = Beispiel ,
6301         Name-pl = Beispiele ,
6302     case = D ,
6303         Name-sg = Beispiel ,
6304         Name-pl = Beispielen ,
6305     case = G ,
6306         Name-sg = Beispiels ,
6307         Name-pl = Beispiele ,
6308
6309 type = algorithm ,
6310     gender = m ,
6311     case = N ,
6312         Name-sg = Algorithmus ,
6313         Name-pl = Algorithmen ,
6314     case = A ,
6315         Name-sg = Algorithmus ,
6316         Name-pl = Algorithmen ,
6317     case = D ,
6318         Name-sg = Algorithmus ,
6319         Name-pl = Algorithmen ,
6320     case = G ,
6321         Name-sg = Algorithmus ,
6322         Name-pl = Algorithmen ,
6323
6324 type = listing ,
6325     gender = n ,
6326     case = N ,
6327         Name-sg = Listing ,
6328         Name-pl = Listings ,
6329     case = A ,
6330         Name-sg = Listing ,
6331         Name-pl = Listings ,
6332     case = D ,
6333         Name-sg = Listing ,
6334         Name-pl = Listings ,
6335     case = G ,
6336         Name-sg = Listings ,
6337         Name-pl = Listings ,
6338
6339 type = exercise ,
6340     gender = f ,
6341     case = N ,
6342         Name-sg = Übungsaufgabe ,
6343         Name-pl = Übungsaufgaben ,
6344     case = A ,
6345         Name-sg = Übungsaufgabe ,
6346         Name-pl = Übungsaufgaben ,
6347     case = D ,

```

```

6348     Name-sg = Übungsaufgabe ,
6349     Name-pl = Übungsaufgaben ,
6350     case = G ,
6351     Name-sg = Übungsaufgabe ,
6352     Name-pl = Übungsaufgaben ,
6353
6354     type = solution ,
6355     gender = f ,
6356     case = N ,
6357     Name-sg = Lösung ,
6358     Name-pl = Lösungen ,
6359     case = A ,
6360     Name-sg = Lösung ,
6361     Name-pl = Lösungen ,
6362     case = D ,
6363     Name-sg = Lösung ,
6364     Name-pl = Lösungen ,
6365     case = G ,
6366     Name-sg = Lösung ,
6367     Name-pl = Lösungen ,
6368 </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.

```

6369 <*package>
6370 \zcDeclareLanguage [ gender = { f , m } ] { french }
6371 \zcDeclareLanguageAlias { acadian } { french }
6372 \zcDeclareLanguageAlias { canadien } { french }
6373 \zcDeclareLanguageAlias { francais } { french }
6374 \zcDeclareLanguageAlias { frenchb } { french }
6375 </package>
6376 <*lang-french>
6377 namesep = {\nobreakspace} ,
6378 pairsep = {\~et\nobreakspace} ,
6379 listsep = { ,~ } ,
6380 lastsep = {\~et\nobreakspace} ,
6381 tpairsep = {\~et\nobreakspace} ,
6382 tlistsep = { ,~ } ,
6383 tlastsep = {\~et\nobreakspace} ,
6384 notesep = { ~ } ,
6385 rangesep = {\~à\nobreakspace} ,
6386
6387 type = book ,
6388 gender = m ,
6389 Name-sg = Livre ,
6390 name-sg = livre ,
6391 Name-pl = Livres ,

```

```

6392     name-pl = livres ,
6393
6394     type = part ,
6395     gender = f ,
6396     Name-sg = Partie ,
6397     name-sg = partie ,
6398     Name-pl = Parties ,
6399     name-pl = parties ,
6400
6401     type = chapter ,
6402     gender = m ,
6403     Name-sg = Chapitre ,
6404     name-sg = chapitre ,
6405     Name-pl = Chapitres ,
6406     name-pl = chapitres ,
6407
6408     type = section ,
6409     gender = f ,
6410     Name-sg = Section ,
6411     name-sg = section ,
6412     Name-pl = Sections ,
6413     name-pl = sections ,
6414
6415     type = paragraph ,
6416     gender = m ,
6417     Name-sg = Paragraphe ,
6418     name-sg = paragraphe ,
6419     Name-pl = Paragraphes ,
6420     name-pl = paragraphes ,
6421
6422     type = appendix ,
6423     gender = f ,
6424     Name-sg = Annexe ,
6425     name-sg = annexe ,
6426     Name-pl = Annexes ,
6427     name-pl = annexes ,
6428
6429     type = page ,
6430     gender = f ,
6431     Name-sg = Page ,
6432     name-sg = page ,
6433     Name-pl = Pages ,
6434     name-pl = pages ,
6435     rangesep = {-} ,
6436     rangetopair = false ,
6437
6438     type = line ,
6439     gender = f ,
6440     Name-sg = Ligne ,
6441     name-sg = ligne ,
6442     Name-pl = Lignes ,
6443     name-pl = lignes ,
6444
6445     type = figure ,

```



```

6446 gender = f ,
6447 Name-sg = Figure ,
6448 name-sg = figure ,
6449 Name-pl = Figures ,
6450 name-pl = figures ,
6451
6452 type = table ,
6453 gender = f ,
6454 Name-sg = Table ,
6455 name-sg = table ,
6456 Name-pl = Tables ,
6457 name-pl = tables ,
6458
6459 type = item ,
6460 gender = m ,
6461 Name-sg = Point ,
6462 name-sg = point ,
6463 Name-pl = Points ,
6464 name-pl = points ,
6465
6466 type = footnote ,
6467 gender = f ,
6468 Name-sg = Note ,
6469 name-sg = note ,
6470 Name-pl = Notes ,
6471 name-pl = notes ,
6472
6473 type = endnote ,
6474 gender = f ,
6475 Name-sg = Note ,
6476 name-sg = note ,
6477 Name-pl = Notes ,
6478 name-pl = notes ,
6479
6480 type = note ,
6481 gender = f ,
6482 Name-sg = Note ,
6483 name-sg = note ,
6484 Name-pl = Notes ,
6485 name-pl = notes ,
6486
6487 type = equation ,
6488 gender = f ,
6489 Name-sg = Équation ,
6490 name-sg = équation ,
6491 Name-pl = Équations ,
6492 name-pl = équations ,
6493 refbounds-first-sg = {(,),(,)} ,
6494 refbounds = {(,,)} ,
6495
6496 type = theorem ,
6497 gender = m ,
6498 Name-sg = Théorème ,
6499 name-sg = théorème ,

```

```

6500   Name-pl = Théorèmes ,
6501   name-pl = théorèmes ,
6502
6503   type = lemma ,
6504   gender = m ,
6505   Name-sg = Lemme ,
6506   name-sg = lemme ,
6507   Name-pl = Lemmes ,
6508   name-pl = lemmes ,
6509
6510   type = corollary ,
6511   gender = m ,
6512   Name-sg = Corollaire ,
6513   name-sg = corollaire ,
6514   Name-pl = Corollaires ,
6515   name-pl = corollaires ,
6516
6517   type = proposition ,
6518   gender = f ,
6519   Name-sg = Proposition ,
6520   name-sg = proposition ,
6521   Name-pl = Propositions ,
6522   name-pl = propositions ,
6523
6524   type = definition ,
6525   gender = f ,
6526   Name-sg = Définition ,
6527   name-sg = définition ,
6528   Name-pl = Définitions ,
6529   name-pl = définitions ,
6530
6531   type = proof ,
6532   gender = f ,
6533   Name-sg = Démonstration ,
6534   name-sg = démonstration ,
6535   Name-pl = Démonstrations ,
6536   name-pl = démonstrations ,
6537
6538   type = result ,
6539   gender = m ,
6540   Name-sg = Résultat ,
6541   name-sg = résultat ,
6542   Name-pl = Résultats ,
6543   name-pl = résultats ,
6544
6545   type = remark ,
6546   gender = f ,
6547   Name-sg = Remarque ,
6548   name-sg = remarque ,
6549   Name-pl = Remarques ,
6550   name-pl = remarques ,
6551
6552   type = example ,
6553   gender = m ,

```

```

6554 Name-sg = Exemple ,
6555 name-sg = exemple ,
6556 Name-pl = Exemples ,
6557 name-pl = exemples ,
6558
6559 type = algorithm ,
6560 gender = m ,
6561 Name-sg = Algorithme ,
6562 name-sg = algorithme ,
6563 Name-pl = Algorithmes ,
6564 name-pl = algorithmes ,
6565
6566 type = listing ,
6567 gender = m ,
6568 Name-sg = Listing ,
6569 name-sg = listing ,
6570 Name-pl = Listings ,
6571 name-pl = listings ,
6572
6573 type = exercise ,
6574 gender = m ,
6575 Name-sg = Exercice ,
6576 name-sg = exercice ,
6577 Name-pl = Exercices ,
6578 name-pl = exercices ,
6579
6580 type = solution ,
6581 gender = f ,
6582 Name-sg = Solution ,
6583 name-sg = solution ,
6584 Name-pl = Solutions ,
6585 name-pl = solutions ,
6586 </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.

```

6587 <*package>
6588 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6589 \zcDeclareLanguageAlias { brazilian } { portuguese }
6590 \zcDeclareLanguageAlias { brazil } { portuguese }
6591 \zcDeclareLanguageAlias { portuges } { portuguese }
6592 </package>
6593 <*lang-portuguese>
6594 namesep = {\nobreakspace} ,
6595 pairsep = {\sim\nobreakspace} ,
6596 listsep = { , ~ } ,
6597 lastsep = {\sim\nobreakspace} ,
6598 tpairsep = {\sim\nobreakspace} ,
6599 tlistsep = { , ~ } ,

```

```

6600 tlastsep = {\~e\nobreakspace} ,
6601 notesep = {\~} ,
6602 rangesep = {\~a\nobreakspace} ,
6603
6604 type = book ,
6605     gender = m ,
6606     Name-sg = Livro ,
6607     name-sg = livro ,
6608     Name-pl = Livros ,
6609     name-pl = livros ,
6610
6611 type = part ,
6612     gender = f ,
6613     Name-sg = Parte ,
6614     name-sg = parte ,
6615     Name-pl = Partes ,
6616     name-pl = partes ,
6617
6618 type = chapter ,
6619     gender = m ,
6620     Name-sg = Capítulo ,
6621     name-sg = capítulo ,
6622     Name-pl = Capítulos ,
6623     name-pl = capítulos ,
6624
6625 type = section ,
6626     gender = f ,
6627     Name-sg = Seção ,
6628     name-sg = seção ,
6629     Name-pl = Seções ,
6630     name-pl = seções ,
6631
6632 type = paragraph ,
6633     gender = m ,
6634     Name-sg = Parágrafo ,
6635     name-sg = parágrafo ,
6636     Name-pl = Parágrafos ,
6637     name-pl = parágrafos ,
6638     Name-sg-ab = Par. ,
6639     name-sg-ab = par. ,
6640     Name-pl-ab = Par. ,
6641     name-pl-ab = par. ,
6642
6643 type = appendix ,
6644     gender = m ,
6645     Name-sg = Apêndice ,
6646     name-sg = apêndice ,
6647     Name-pl = Apêndices ,
6648     name-pl = apêndices ,
6649
6650 type = page ,
6651     gender = f ,
6652     Name-sg = Página ,
6653     name-sg = página ,

```

```

6654 Name-pl = Páginas ,
6655 name-pl = páginas ,
6656 rangesep = {\textendash} ,
6657 rangetopair = false ,
6658
6659 type = line ,
6660 gender = f ,
6661 Name-sg = Linha ,
6662 name-sg = linha ,
6663 Name-pl = Linhas ,
6664 name-pl = linhas ,
6665
6666 type = figure ,
6667 gender = f ,
6668 Name-sg = Figura ,
6669 name-sg = figura ,
6670 Name-pl = Figuras ,
6671 name-pl = figuras ,
6672 Name-sg-ab = Fig. ,
6673 name-sg-ab = fig. ,
6674 Name-pl-ab = Figs. ,
6675 name-pl-ab = figs. ,
6676
6677 type = table ,
6678 gender = f ,
6679 Name-sg = Tabela ,
6680 name-sg = tabela ,
6681 Name-pl = Tabelas ,
6682 name-pl = tabelas ,
6683
6684 type = item ,
6685 gender = m ,
6686 Name-sg = Item ,
6687 name-sg = item ,
6688 Name-pl = Itens ,
6689 name-pl = itens ,
6690
6691 type = footnote ,
6692 gender = f ,
6693 Name-sg = Nota ,
6694 name-sg = nota ,
6695 Name-pl = Notas ,
6696 name-pl = notas ,
6697
6698 type = endnote ,
6699 gender = f ,
6700 Name-sg = Nota ,
6701 name-sg = nota ,
6702 Name-pl = Notas ,
6703 name-pl = notas ,
6704
6705 type = note ,
6706 gender = f ,
6707 Name-sg = Nota ,

```

```

6708     name-sg = nota ,
6709     Name-pl = Notas ,
6710     name-pl = notas ,
6711
6712 type = equation ,
6713     gender = f ,
6714     Name-sg = Equação ,
6715     name-sg = equação ,
6716     Name-pl = Equações ,
6717     name-pl = equações ,
6718     Name-sg-ab = Eq. ,
6719     name-sg-ab = eq. ,
6720     Name-pl-ab = Eqs. ,
6721     name-pl-ab = eqs. ,
6722     refbounds-first-sg = {,(,),} ,
6723     refbounds = {(,,)} ,
6724
6725 type = theorem ,
6726     gender = m ,
6727     Name-sg = Teorema ,
6728     name-sg = teorema ,
6729     Name-pl = Teoremas ,
6730     name-pl = teoremas ,
6731
6732 type = lemma ,
6733     gender = m ,
6734     Name-sg = Lema ,
6735     name-sg = lema ,
6736     Name-pl = Lemas ,
6737     name-pl = lemas ,
6738
6739 type = corollary ,
6740     gender = m ,
6741     Name-sg = Corolário ,
6742     name-sg = corolário ,
6743     Name-pl = Corolários ,
6744     name-pl = corolários ,
6745
6746 type = proposition ,
6747     gender = f ,
6748     Name-sg = Proposição ,
6749     name-sg = proposição ,
6750     Name-pl = Proposições ,
6751     name-pl = proposições ,
6752
6753 type = definition ,
6754     gender = f ,
6755     Name-sg = Definição ,
6756     name-sg = definição ,
6757     Name-pl = Definições ,
6758     name-pl = definições ,
6759
6760 type = proof ,
6761     gender = f ,

```

```

6762 Name-sg = Demonstração ,
6763 name-sg = demonstração ,
6764 Name-pl = Demonstrações ,
6765 name-pl = demonstrações ,
6766
6767 type = result ,
6768 gender = m ,
6769 Name-sg = Resultado ,
6770 name-sg = resultado ,
6771 Name-pl = Resultados ,
6772 name-pl = resultados ,
6773
6774 type = remark ,
6775 gender = f ,
6776 Name-sg = Observação ,
6777 name-sg = observação ,
6778 Name-pl = Observações ,
6779 name-pl = observações ,
6780
6781 type = example ,
6782 gender = m ,
6783 Name-sg = Exemplo ,
6784 name-sg = exemplo ,
6785 Name-pl = Exemplos ,
6786 name-pl = exemplos ,
6787
6788 type = algorithm ,
6789 gender = m ,
6790 Name-sg = Algoritmo ,
6791 name-sg = algoritmo ,
6792 Name-pl = Algoritmos ,
6793 name-pl = algoritmos ,
6794
6795 type = listing ,
6796 gender = f ,
6797 Name-sg = Listagem ,
6798 name-sg = listagem ,
6799 Name-pl = Listagens ,
6800 name-pl = listagens ,
6801
6802 type = exercise ,
6803 gender = m ,
6804 Name-sg = Exercício ,
6805 name-sg = exercício ,
6806 Name-pl = Exercícios ,
6807 name-pl = exercícios ,
6808
6809 type = solution ,
6810 gender = f ,
6811 Name-sg = Solução ,
6812 name-sg = solução ,
6813 Name-pl = Soluções ,
6814 name-pl = soluções ,
6815 </lang-portuguese>

```

10.5 Spanish

Spanish language file has been initially provided by the author.

```
6816 <*package>
6817 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6818 </package>

6819 <*lang-spanish>

6820 namesep = {\nobreakspace} ,
6821 pairsep = {\~y\nobreakspace} ,
6822 listsep = { ,~ } ,
6823 lastsep = {\~y\nobreakspace} ,
6824 tpairsep = {\~y\nobreakspace} ,
6825 tlistsep = { ,~ } ,
6826 tlastsep = {\~y\nobreakspace} ,
6827 notesep = {~} ,
6828 rangesep = {\~a\nobreakspace} ,
6829
6830 type = book ,
6831   gender = m ,
6832   Name-sg = Libro ,
6833   name-sg = libro ,
6834   Name-pl = Libros ,
6835   name-pl = libros ,
6836
6837 type = part ,
6838   gender = f ,
6839   Name-sg = Parte ,
6840   name-sg = parte ,
6841   Name-pl = Partes ,
6842   name-pl = partes ,
6843
6844 type = chapter ,
6845   gender = m ,
6846   Name-sg = Capítulo ,
6847   name-sg = capítulo ,
6848   Name-pl = Capítulos ,
6849   name-pl = capítulos ,
6850
6851 type = section ,
6852   gender = f ,
6853   Name-sg = Sección ,
6854   name-sg = sección ,
6855   Name-pl = Secciones ,
6856   name-pl = secciones ,
6857
6858 type = paragraph ,
6859   gender = m ,
6860   Name-sg = Párrafo ,
6861   name-sg = párrafo ,
6862   Name-pl = Párrafos ,
6863   name-pl = párrafos ,
6864
6865 type = appendix ,
```



```

6866     gender = m ,
6867     Name-sg = Apéndice ,
6868     name-sg = apéndice ,
6869     Name-pl = Apéndices ,
6870     name-pl = apéndices ,
6871
6872     type = page ,
6873     gender = f ,
6874     Name-sg = Página ,
6875     name-sg = página ,
6876     Name-pl = Páginas ,
6877     name-pl = páginas ,
6878     rangesep = {\textendash} ,
6879     rangetopair = false ,
6880
6881     type = line ,
6882     gender = f ,
6883     Name-sg = Línea ,
6884     name-sg = línea ,
6885     Name-pl = Líneas ,
6886     name-pl = líneas ,
6887
6888     type = figure ,
6889     gender = f ,
6890     Name-sg = Figura ,
6891     name-sg = figura ,
6892     Name-pl = Figuras ,
6893     name-pl = figuras ,
6894
6895     type = table ,
6896     gender = m ,
6897     Name-sg = Cuadro ,
6898     name-sg = cuadro ,
6899     Name-pl = Cuadros ,
6900     name-pl = cuadros ,
6901
6902     type = item ,
6903     gender = m ,
6904     Name-sg = Punto ,
6905     name-sg = punto ,
6906     Name-pl = Puntos ,
6907     name-pl = puntos ,
6908
6909     type = footnote ,
6910     gender = f ,
6911     Name-sg = Nota ,
6912     name-sg = nota ,
6913     Name-pl = Notas ,
6914     name-pl = notas ,
6915
6916     type = endnote ,
6917     gender = f ,
6918     Name-sg = Nota ,
6919     name-sg = nota ,

```

```

6920   Name-pl = Notas ,
6921   name-pl = notas ,
6922
6923   type = note ,
6924   gender = f ,
6925   Name-sg = Nota ,
6926   name-sg = nota ,
6927   Name-pl = Notas ,
6928   name-pl = notas ,
6929
6930   type = equation ,
6931   gender = f ,
6932   Name-sg = Ecuación ,
6933   name-sg = ecuación ,
6934   Name-pl = Ecuaciones ,
6935   name-pl = ecuaciones ,
6936   refbounds-first-sg = {,(,),} ,
6937   refbounds = {(,,)} ,
6938
6939   type = theorem ,
6940   gender = m ,
6941   Name-sg = Teorema ,
6942   name-sg = teorema ,
6943   Name-pl = Teoremas ,
6944   name-pl = teoremas ,
6945
6946   type = lemma ,
6947   gender = m ,
6948   Name-sg = Lema ,
6949   name-sg = lema ,
6950   Name-pl = Lemas ,
6951   name-pl = lemas ,
6952
6953   type = corollary ,
6954   gender = m ,
6955   Name-sg = Corolario ,
6956   name-sg = corolario ,
6957   Name-pl = Corolarios ,
6958   name-pl = corolarios ,
6959
6960   type = proposition ,
6961   gender = f ,
6962   Name-sg = Proposición ,
6963   name-sg = proposición ,
6964   Name-pl = Proposiciones ,
6965   name-pl = proposiciones ,
6966
6967   type = definition ,
6968   gender = f ,
6969   Name-sg = Definición ,
6970   name-sg = definición ,
6971   Name-pl = Definiciones ,
6972   name-pl = definiciones ,
6973

```

```

6974 type = proof ,
6975     gender = f ,
6976     Name-sg = Demostración ,
6977     name-sg = demostración ,
6978     Name-pl = Demostraciones ,
6979     name-pl = demostraciones ,
6980
6981 type = result ,
6982     gender = m ,
6983     Name-sg = Resultado ,
6984     name-sg = resultado ,
6985     Name-pl = Resultados ,
6986     name-pl = resultados ,
6987
6988 type = remark ,
6989     gender = f ,
6990     Name-sg = Observación ,
6991     name-sg = observación ,
6992     Name-pl = Observaciones ,
6993     name-pl = observaciones ,
6994
6995 type = example ,
6996     gender = m ,
6997     Name-sg = Ejemplo ,
6998     name-sg = ejemplo ,
6999     Name-pl = Ejemplos ,
7000     name-pl = ejemplos ,
7001
7002 type = algorithm ,
7003     gender = m ,
7004     Name-sg = Algoritmo ,
7005     name-sg = algoritmo ,
7006     Name-pl = Algoritmos ,
7007     name-pl = algoritmos ,
7008
7009 type = listing ,
7010     gender = m ,
7011     Name-sg = Listado ,
7012     name-sg = listado ,
7013     Name-pl = Listados ,
7014     name-pl = listados ,
7015
7016 type = exercise ,
7017     gender = m ,
7018     Name-sg = Ejercicio ,
7019     name-sg = ejercicio ,
7020     Name-pl = Ejercicios ,
7021     name-pl = ejercicios ,
7022
7023 type = solution ,
7024     gender = f ,
7025     Name-sg = Solución ,
7026     name-sg = solución ,
7027     Name-pl = Soluciones ,

```

```

7028   name-pl = soluciones ,
7029 </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.

```

7030 <*package>
7031 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7032 </package>
7033 <*lang-dutch>
7034 namesep   = {\nobreakspace} ,
7035 pairsep    = {\sim\nobreakspace} ,
7036 listsep    = {,~} ,
7037 lastsep    = {\sim\nobreakspace} ,
7038 tpairsep   = {\sim\nobreakspace} ,
7039 tlistsep   = {,~} ,
7040 tlastsep   = {\sim\nobreakspace} ,
7041 notesep    = {~} ,
7042 rangesep   = {\sim/m\nobreakspace} ,
7043
7044 type = book ,
7045   gender = n ,
7046   Name-sg = Boek ,
7047   name-sg = boek ,
7048   Name-pl = Boeken ,
7049   name-pl = boeken ,
7050
7051 type = part ,
7052   gender = n ,
7053   Name-sg = Deel ,
7054   name-sg = deel ,
7055   Name-pl = Delen ,
7056   name-pl = delen ,
7057
7058 type = chapter ,
7059   gender = n ,
7060   Name-sg = Hoofdstuk ,
7061   name-sg = hoofdstuk ,
7062   Name-pl = Hoofdstukken ,
7063   name-pl = hoofdstukken ,
7064
7065 type = section ,
7066   gender = m ,
7067   Name-sg = Paragraaf ,
7068   name-sg = paragraaf ,
7069   Name-pl = Paragrafen ,
7070   name-pl = paragrafen ,
7071
7072 type = paragraph ,
7073   gender = f ,
7074   Name-sg = Alinea ,

```

```

7075 name-sg = alinea ,
7076 Name-pl = Alinea's ,
7077 name-pl = alinea's ,
7078
7079 type = appendix ,
7080 gender = { m , n } ,
7081 Name-sg = Appendix ,
7082 name-sg = appendix ,
7083 Name-pl = Appendices ,
7084 name-pl = appendices ,
7085
7086 type = page ,
7087 gender = { f , m } ,
7088 Name-sg = Pagina ,
7089 name-sg = pagina ,
7090 Name-pl = Pagina's ,
7091 name-pl = pagina's ,
7092 rangesep = {\textendash} ,
7093 rangetopair = false ,
7094
7095 type = line ,
7096 gender = m ,
7097 Name-sg = Regel ,
7098 name-sg = regel ,
7099 Name-pl = Regels ,
7100 name-pl = regels ,
7101
7102 type = figure ,
7103 gender = { n , f , m } ,
7104 Name-sg = Figuur ,
7105 name-sg = figuur ,
7106 Name-pl = Figuren ,
7107 name-pl = figuren ,
7108
7109 type = table ,
7110 gender = { f , m } ,
7111 Name-sg = Tabel ,
7112 name-sg = tabel ,
7113 Name-pl = Tabellen ,
7114 name-pl = tabellen ,
7115
7116 type = item ,
7117 gender = n ,
7118 Name-sg = Punt ,
7119 name-sg = punt ,
7120 Name-pl = Punten ,
7121 name-pl = punten ,
7122
7123 type = footnote ,
7124 gender = { f , m } ,
7125 Name-sg = Voetnoot ,
7126 name-sg = voetnoot ,
7127 Name-pl = Voetnoten ,
7128 name-pl = voetnoten ,

```

```

7129
7130 type = endnote ,
7131   gender = { f , m } ,
7132   Name-sg = Eindnoot ,
7133   name-sg = eindnoot ,
7134   Name-pl = Eindnoten ,
7135   name-pl = eindnoten ,
7136
7137 type = note ,
7138   gender = f ,
7139   Name-sg = Opmerking ,
7140   name-sg = opmerking ,
7141   Name-pl = Opmerkingen ,
7142   name-pl = opmerkingen ,
7143
7144 type = equation ,
7145   gender = f ,
7146   Name-sg = Vergelijking ,
7147   name-sg = vergelijking ,
7148   Name-pl = Vergelijkingen ,
7149   name-pl = vergelijkingen ,
7150   Name-sg-ab = Vgl. ,
7151   name-sg-ab = vgl. ,
7152   Name-pl-ab = Vgl.'s ,
7153   name-pl-ab = vgl.'s ,
7154   refbounds-first-sg = {,(,),} ,
7155   refbounds = {(,,)} ,
7156
7157 type = theorem ,
7158   gender = f ,
7159   Name-sg = Stelling ,
7160   name-sg = stelling ,
7161   Name-pl = Stellingen ,
7162   name-pl = stellingen ,
7163

```

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”.

```

7164 type = lemma ,
7165   gender = n ,
7166   Name-sg = Lemma ,
7167   name-sg = lemma ,
7168   Name-pl = Lemma's ,
7169   name-pl = lemma's ,
7170
7171 type = corollary ,
7172   gender = n ,
7173   Name-sg = Gevolg ,
7174   name-sg = gevolg ,
7175   Name-pl = Gevolgen ,
7176   name-pl = gevolgen ,
7177
7178 type = proposition ,

```

```

7179     gender = f ,
7180     Name-sg = Propositie ,
7181     name-sg = propositie ,
7182     Name-pl = Propositiones ,
7183     name-pl = proposities ,
7184
7185 type = definition ,
7186     gender = f ,
7187     Name-sg = Definitie ,
7188     name-sg = definitie ,
7189     Name-pl = Definities ,
7190     name-pl = definities ,
7191
7192 type = proof ,
7193     gender = n ,
7194     Name-sg = Bewijs ,
7195     name-sg = bewijs ,
7196     Name-pl = Bewijzen ,
7197     name-pl = bewijzen ,
7198
7199 type = result ,
7200     gender = n ,
7201     Name-sg = Resultaat ,
7202     name-sg = resultaat ,
7203     Name-pl = Resultaten ,
7204     name-pl = resultaten ,
7205
7206 type = remark ,
7207     gender = f ,
7208     Name-sg = Opmerking ,
7209     name-sg = opmerking ,
7210     Name-pl = Opmerkingen ,
7211     name-pl = opmerkingen ,
7212
7213 type = example ,
7214     gender = n ,
7215     Name-sg = Voorbeeld ,
7216     name-sg = voorbeeld ,
7217     Name-pl = Voorbeelden ,
7218     name-pl = voorbeelden ,
7219
7220 type = algorithm ,
7221     gender = { n , f , m } ,
7222     Name-sg = Algoritme ,
7223     name-sg = algoritme ,
7224     Name-pl = Algoritmes ,
7225     name-pl = algoritmes ,
7226

```

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

```

7227 type = listing ,
7228     gender = m ,
7229     Name-sg = Listing ,

```

```

7230 name-sg = listing ,
7231 Name-pl = Listings ,
7232 name-pl = listings ,
7233
7234 type = exercise ,
7235 gender = { f , m } ,
7236 Name-sg = Opgave ,
7237 name-sg = opgave ,
7238 Name-pl = Opgaven ,
7239 name-pl = opgaven ,
7240
7241 type = solution ,
7242 gender = f ,
7243 Name-sg = Oplossing ,
7244 name-sg = oplossing ,
7245 Name-pl = Oplossingen ,
7246 name-pl = oplossingen ,
7247 </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	
\A	1874
\AddToHook	98, 2020,
	2064, 2087, 2118, 2120, 2158, 2231,
	2273, 2417, 2430, 2438, 5300, 5336,
	5353, 5355, 5360, 5405, 5407, 5409,
	5411, 5413, 5415, 5417, 5419, 5423,
	5427, 5429, 5434, 5444, 5453, 5459,
	5461, 5487, 5498, 5532, 5585, 5610
\alph	5513, 5601
\appendix	2, 126, 127, 129
\appendixname	126
\AtEndOfPackage	2428
B	
\babelname	2074
\babelprovide	29, 54
\bicaption	128
\bionenumcaption	128
\bitwonumcaption	128
bool commands:	
\bool_case_true:	2
\bool_gset_false:N	530
\bool_gset_true:N	523, 2404
\bool_if:NTF	374, 451, 489,
	547, 1750, 1801, 2024, 2028, 2440,
	3414, 3817, 3958, 4085, 4121, 4155,
	4222, 4235, 4247, 4298, 4315, 4325,
	4330, 4376, 4380, 4436, 4461, 4468,
	4474, 4485, 4491, 4519, 4564, 4586,
	4615, 4764, 4920, 4922, 5535, 5613
\bool_if:nTF	67,
	3528, 3538, 3562, 3579, 3594, 3659,
	3667, 4308, 4685, 4726, 4810, 4827
\bool_if_exist:NTF	328, 338,
	362, 372, 422, 439, 449, 474, 477,
	485, 487, 501, 504, 511, 514, 521, 528
\bool_lazy_all:nTF	4404, 4932
\bool_lazy_and:nnTF ..	2561, 3385,
	3406, 4189, 4260, 4638, 4905, 4947
\bool_lazy_any:nTF	5084, 5093
\bool_lazy_or:nnTF	1367, 1400, 1950,
	2510, 2774, 3273, 3298, 3389, 4893
\bool_new:N	329, 339, 364, 423,
	441, 479, 502, 505, 512, 515, 522,
	529, 789, 1564, 1565, 1592, 1616,
	1968, 1975, 1982, 1995, 1996, 2166,
	2167, 2168, 2169, 2170, 2266, 2267,
	2397, 3422, 3437, 3699, 3700, 3711,
	3712, 3719, 3721, 3722, 3735, 3736,
	3737, 3749, 3750, 5497, 5544, 5584
\bool_set:Nn	3382
\bool_set_eq:NN	537
\bool_set_false:N ..	363, 440, 476,
	478, 513, 1577, 1581, 1753, 1851,
	1896, 1938, 2003, 2012, 2013, 2030,

2037, 2178, 2182, 2189, 2197, 2198, 2199, 2291, 2303, 3427, 3520, 3766, 3767, 3784, 3823, 3834, 4234, 4426, 4427, 4434, 4435, 4668, 5091, 5108	427, 434, 472, 499, 509, 519, 526, 802, 904, 1513, 1532, 1736, 1835, 1880, 1922, 2436, 2567, 3377, 3441, 3483, 3494, 3507, 3637, 3689, 3751, 3965, 4430, 4677, 4679, 4876, 5111, 5128, 5199, 5232, 5265, 5395, 5552
\bool_set_true:N 330, 340, 424, 503, 506, 516, 1571, 1572, 1576, 1582, 1775, 1797, 1860, 1862, 1900, 1902, 1911, 1913, 1942, 1944, 1957, 1959, 2002, 2007, 2008, 2176, 2183, 2188, 2205, 2207, 2209, 2212, 2213, 2214, 2279, 2284, 3534, 3544, 3548, 3570, 3585, 3600, 3624, 3792, 3818, 3824, 3828, 3835, 3981, 3992, 4003, 4053, 4089, 4125, 4159, 4176, 4257, 4291, 4463, 4523, 4569, 4591, 4620, 4881, 4888, 5010, 5069, 5107, 5144, 5151, 5152, 5170, 5187, 5189, 5511, 5551, 5598	\cs_set:Npn 5398 \cs_set_eq:NN 711, 5397, 5403, 5490, 5495 \cs_set_nopar:Npn 5480 \cs_to_str:N 322
\bool_until_do:Nn 1852, 1903, 1945, 3560, 3785 \bool_while_do:nn 5645 \l_tmpa_bool 1753, 1775, 1797, 1801, 1851, 1852, 1860, 1862, 1896, 1900, 1902, 1903, 1911, 1913, 1938, 1942, 1944, 1945, 1957, 1959	D \d 1874
C \caption 131 clist commands: \clist_map_inline:nn 622, 670, 687, 1006, 2200, 2345, 2406, 2910, 5514, 5603 \contsubbottom 129 \contsubcaption 129 \counterwithin 5 \crefstripprefix 45 cs commands: \cs_generate_variant:Nn 64, 271, 277, 284, 295, 306, 317, 332, 342, 349, 356, 367, 391, 401, 426, 433, 444, 482, 508, 518, 525, 532, 960, 1834, 1870, 1921, 1967, 2569, 4720, 4758, 5127, 5231, 5264, 5297 \cs_if_exist:NTF 24, 27, 45, 48, 57, 77, 5324, 5330, 5451 \cs_if_exist_p:N 4191, 4262, 4640, 5647 \cs_if_exist_use:N 5422, 5685 \cs_new:Npn 55, 65, 75, 86, 272, 278, 280, 282, 285, 296, 307, 319, 321, 709, 4681, 4721, 4759, 5117 \cs_new_eq:NN 5489, 5494 \cs_new_protected:Npn 266, 323, 333, 343, 350, 357, 382, 392, 413, 415, 417,	E \endinput 12 \endsidecaption 128 exp commands: \exp_args:NNe 34, 37 \exp_args:NNNo 268 \exp_args:NNNV 1815, 1864, 1915, 1961 \exp_args:NNo 268, 274 \exp_args:No 274 \exp_args:Nx 914, 5557, 5565 \exp_args:Nxx 1754, 1766, 1778, 1788, 1854, 1905, 1947, 5130, 5136, 5158, 5162, 5177 \exp_not:N 112, 4332, 4345, 4348, 4351, 4691, 4704, 4707, 4715, 4732, 4744, 4747, 4754, 4770, 4781, 4784, 4786, 4794, 4798, 4801, 4804, 4816, 4819, 4833, 4857, 4860, 4869 \exp_not:n .. 275, 3997, 4020, 4028, 4046, 4059, 4063, 4098, 4107, 4129, 4137, 4144, 4168, 4182, 4199, 4209, 4251, 4274, 4284, 4333, 4344, 4349, 4350, 4507, 4530, 4542, 4576, 4598, 4607, 4627, 4648, 4658, 4692, 4708, 4733, 4740, 4748, 4751, 4782, 4783, 4785, 4787, 4796, 4802, 4803, 4805, 4813, 4817, 4818, 4821, 4834, 4861 \ExplSyntaxOn 30, 920
F file commands: \file_get:nnNTF 914 \fmtversion 3 \footnote 2, 129, 130	G group commands: \group_begin: 100, 725, 906, 1752, 1839, 1884, 1926, 2834, 3379, 3393, 3425, 4332, 4348, 4691, 4707,

4732, 4747, 4770, 4781, 4786, 4801, 4816, 4833, 4860, 5410, 5414, 5418	
\group_end:	103, 737, 958, 1816, 1865, 1916, 1962, 2862, 3396, 3419, 3429, 4345, 4351, 4704, 4715, 4744, 4754, 4784, 4794, 4798, 4804, 4819, 4857, 4869, 5412, 5416, 5420
H	
\hyperlink	5114
I	
\IfBooleanT	3426
\IfClassLoadedTF	110
\ifdraft	2181
\IfFormatAtLeastTF	3, 4
\ifoptionfinal	2187
\IfPackageAtLeastTF	2277
\IfPackageLoadedTF	108
\input	29, 30
int commands:	
\int_case:nnTF 3968, 4010, 4072, 4383, 4498, 4555
\int_compare:nNnTF	3571, 3586, 3601, 3613, 3625, 3645, 3647, 3691, 3865, 3988, 4016, 4038, 4093, 4163, 4368, 4370, 4447, 4477, 4536, 5140, 5146, 5166, 5172, 5663
\int_compare_p:nNn	1370, 1403, 1951, 1952, 2512, 2776, 3276, 3301, 3661, 3669, 4408, 4897, 4908, 4937, 5104
\int_incr:N 4423, 4455, 4467, 4469, 4484, 4486, 4490, 4492, 4504, 4527, 4539, 4573, 4595, 4604, 4624, 4675, 5661
\int_new:N	3438, 3439, 3701, 3702, 3703, 3716, 3717
\int_rand:n	293, 304, 315
\int_set:Nn 101, 3646, 3648, 3652, 3655, 5644
\int_to_roman:n	5648, 5655, 5656, 5659
\int_use:N	46, 49, 53, 59
\int_zero:N	3639, 3640, 3761, 3762, 3763, 3764, 3765, 4421, 4422, 4424, 4425, 4670, 4671
\l_tmpa_int	5644, 5648, 5655, 5656, 5659, 5661, 5663
iow commands:	
\iow_char:N 114, 129, 130, 135, 136, 141, 142, 147, 148, 200, 217, 264, 2248, 2257
\iow_newline:	258
J	
\jobname	5444
K	
keys commands:	
\l_keys_choice_tl	794
\keys_define:nn 20, 630, 676, 693, 754, 961, 1050, 1075, 1103, 1312, 1351, 1429, 1539, 1566, 1593, 1602, 1617, 1626, 1969, 1976, 1983, 1989, 1997, 2032, 2041, 2055, 2083, 2122, 2153, 2160, 2172, 2233, 2240, 2242, 2251, 2261, 2268, 2280, 2292, 2304, 2312, 2341, 2367, 2391, 2399, 2419, 2448, 2466, 2496, 2530, 2553, 2579, 2591, 2615, 2727, 2757, 2797, 2865, 2936, 2960, 2987, 3193, 3223, 3263, 3325
\keys_set:nn	27, 30, 57, 58, 83, 734, 882, 945, 2285, 2568, 2573, 2859, 3380
keyval commands:	
\keyval_parse:nnn	1518, 2316, 2371
\KOMAClassName	5451, 5473
L	
\label	128, 132, 135, 5397, 5403
\labelformat	3
\language name	24, 2068
M	
\mainbabelname	24, 2075
\MessageBreak	10
MH commands:	
\MH_if_boolean:nTF	5549
msg commands:	
\msg_info:nnn	948, 1001, 1066, 1259, 1265, 1319, 5374, 5446, 5473, 5541, 5576, 5617, 5637, 5664
\msg_info:nnnn 974, 981, 1011, 1383, 1417
\msg_info:nnnnn	995
\msg_line_context:	113, 119, 123, 125, 128, 134, 140, 146, 152, 157, 162, 167, 172, 178, 183, 186, 189, 194, 198, 205, 210, 215, 222, 231, 236, 241, 245, 247, 249, 251, 258
\msg_new:nnn	111, 117, 122, 124, 126, 132, 138, 144, 150, 155, 160, 165, 170, 175, 180, 185, 187, 192, 197, 199, 201, 203, 208, 213, 219, 221, 223, 228, 234, 239, 244, 246, 248, 250, 252, 254, 256, 261
\msg_warning:nn 2029, 2035, 2307, 2564, 4410
\msg_warning:nnn 729, 750, 1545, 1552, 1708, 1714, 2107, 2144, 2156, 2218, 2229, 2271, 2296, 2373, 2423, 2433, 2583,

2697, 2703, 2861, 2905, 2951, 3143, 3149, 3230, 3849, 4229, 4926, 4942	\prop_new:N 2311, 2366
\msg_warning:nnnn 818, 835, 869, 887, 2060, 2247, 2256, 2318, 2471, 2477, 2483, 2489, 2519, 2732, 2738, 2744, 2750, 2786, 2878, 2885, 2915, 3198, 3204, 3210, 3216, 3289, 3314, 3857, 5011, 5071	\prop_put:Nnn 1536
\msg_warning:nnnnn 855, 894, 2899, 4961	\prop_remove:Nn 1535
\msg_warning:nnnnnn 4968	\providecommand 3
	\ProvidesExplPackage 14
	\ProvidesFile 29
	R
	\refstepcounter 3, 128, 131–133, 135, 136
	regex commands:
	\regex_match:nnTF 1874
	\renewlist 137
	\RequirePackage 16, 17, 18, 19, 2025
N	S
\newcounter 5, 5320, 5321	\scantokens 127
\NewDocumentCommand 723, 740, 2565, 2570, 2832, 3375, 3423	seq commands:
\newfloat 129	\seq_clear:N 438, 813, 850, 931, 944, 1005, 1613, 2507, 2771, 2845, 2858, 2909, 3443, 5258
\NewHook 1625	\seq_const_from_clist:Nn 21
\newsfloat 129	\seq_count:N 1371, 1385, 1404, 1419, 2512, 2521, 2776, 2788, 3277, 3291, 3302, 3316
\nobreakspace 1526, 5721, 5722, 5724, 5725, 5727, 5729, 5927, 5928, 5930, 5931, 5933, 5935, 6377, 6378, 6380, 6381, 6383, 6385, 6594, 6595, 6597, 6598, 6600, 6602, 6820, 6821, 6823, 6824, 6826, 6828, 7034, 7035, 7037, 7038, 7040, 7042	\seq_gclear:N . 1364, 1397, 3270, 3295
\NumCheckSetup 45	\seq_gconcat:NNN 615, 619
\NumsCheckSetup 45	\seq_get_left:NN 828, 839, 935, 983, 2849, 2887, 3795
P	\seq_gput_right:Nn ... 946, 952, 2410
\PackageError 7	\seq_gremove_all:Nn 2442
\pagenote 129, 130	\seq_gset_eq:NN 431, 1033
\pagenumbering 7	\seq_gset_from_clist:Nn 558, 567, 579, 594, 602, 763, 778
\pageref 84	\seq_gset_split:Nnn 416
\PagesCheckSetup 45	\seq_if_empty:NTF .. 814, 851, 932, 972, 993, 2846, 2876, 2897, 3789, 4959
\paragraph 60, 61	\seq_if_exist:NTF .. 419, 429, 436, 447
prg commands:	\seq_if_in:NnTF 832, 866, 910, 978, 1008, 2347, 2408, 2441, 2882, 2912, 3487, 4955
\prg_generate_conditional_- variant:Nnn 411, 459, 470, 497, 542, 550, 719, 1878	\seq_item:Nn 4693, 4698, 4701, 4703, 4709, 4710, 4713, 4714, 4734, 4739, 4741, 4743, 4749, 4750, 4752, 4753, 4788, 4789, 4793, 4797, 4835, 4847, 4852, 4855, 4862, 4863, 4867, 4868
\prg_new_conditional:Npnn 107, 109, 368, 445, 483, 544, 713	\seq_map_break:n 89, 3680, 3683
\prg_new_protected_conditional:Npnn 402, 461, 533, 1871	\seq_map_function:NN 3446
\prg_return_false: 108, 110, 376, 380, 409, 453, 457, 468, 491, 495, 540, 547, 548, 717, 1876	\seq_map_indexed_inline:Nn . 44, 3641
\prg_return_true: 108, 110, 375, 378, 407, 452, 455, 466, 490, 493, 538, 547, 716, 1875	\seq_map_inline:Nn ... 1047, 1072, 1309, 1348, 1426, 2432, 2445, 2493, 2527, 2576, 2588, 2754, 2794, 2933, 2957, 3220, 3260, 3322, 3677, 5555
\prg_set_eq_conditional:NNn ... 721	\seq_map_tokens:Nn 71
prop commands:	
\prop_if_in:NnTF 34	
\prop_if_in_p:Nn 68	
\prop_item:Nn 37, 69	

<code>\ZREF@mainlist</code>	21, 31, 42,	<code>\tl_if_exist_p:N</code>	2562
	52, 54, 96, 106, 5424, 5536, 5614, 5687	<code>\tl_if_novalue:nTF</code>	
<code>\zref@newprop</code>	5, 7, 20, 22, 32,		2453, 2501, 2596, 2762
	43, 53, 91, 105, 5421, 5513, 5601, 5684	<code>\tl_map_break:n</code>	89
<code>\zref@refused</code>	3842	<code>\tl_map_tokens:Nn</code>	81
<code>\zref@wrapper@babel</code>	82, 132, 3376, 5483	<code>\tl_new:N</code>	97, 326, 336,
<code>\textendash</code>			346, 353, 387, 397, 552, 553, 554,
	1530, 5776, 6041, 6656, 6878, 7092		704, 705, 706, 707, 731, 746, 1538,
<code>\textup</code>	134		2152, 2171, 2239, 2260, 2390, 3431,
<code>\thechapter</code>	126		3432, 3433, 3434, 3435, 3436, 3704,
<code>\thelstnumber</code>	136		3705, 3706, 3707, 3708, 3709, 3710,
<code>\thepage</code>	7, 102		3713, 3714, 3718, 3720, 3723, 3724,
<code>\thesection</code>	126		3725, 3726, 3727, 3728, 3729, 3730,
tl commands:			3731, 3732, 3733, 3734, 5425, 5458
<code>\c_novalue_tl</code>	678, 679, 680, 681,	<code>\tl_put_left:Nn</code>	4311, 4318, 4361,
	682, 683, 684, 2450, 2498, 2593, 2759		4995, 4996, 5038, 5040, 5042, 5044
<code>\tl_clear:N</code>	337, 361, 822,	<code>\tl_put_right:Nn</code>	3995, 4018, 4026,
	860, 873, 890, 899, 924, 933, 966,		4044, 4057, 4096, 4105, 4127, 4135,
	2574, 2837, 2847, 2870, 3755, 3756,		4142, 4166, 4180, 4197, 4207, 4505,
	3757, 3758, 3759, 3760, 3791, 4416,		4528, 4540, 4574, 4596, 4605, 4625,
	4417, 4418, 4419, 4420, 4466, 4483,		4646, 4656, 4903, 4904, 4915, 5686
	4880, 4887, 4917, 5009, 5068, 5225	<code>\tl_reverse:N</code>	3511, 3514
<code>\tl_const:Nn</code>	1515	<code>\tl_set:Nn</code>	
<code>\tl_gclear:N</code>	354, 398, 5471		268, 327, 708, 733, 923, 967, 979,
<code>\tl_gset:Nn</code>	102, 347, 388, 732, 747		1547, 1554, 1556, 1745, 1813, 1817,
<code>\tl_gset_eq:NN</code>	5460		1830, 1841, 1846, 1857, 1858, 1866,
<code>\tl_head:N</code>			1868, 1886, 1891, 1908, 1909, 1917,
	3589, 3602, 3614, 3616, 3626, 3628		1919, 1928, 1933, 1954, 1955, 1963,
<code>\tl_head:n</code>	1855, 1906, 1948, 1952		1965, 2068, 2069, 2074, 2075, 2078,
<code>\tl_if_empty:NnTF</code>	79, 816, 826,		2079, 2093, 2099, 2104, 2130, 2136,
	853, 864, 885, 892, 999, 1055, 1080,		2141, 2572, 2838, 2871, 2883, 3618,
	1112, 1147, 1184, 1221, 1269, 1317,		3620, 3801, 3802, 3975, 3977, 4249,
	1323, 1356, 1434, 1472, 1738, 1859,		4272, 4282, 4328, 4451, 4453, 4464,
	1910, 2903, 2941, 2965, 2996, 3031,		4481, 4899, 4900, 4913, 5426, 5428
	3068, 3105, 3153, 3228, 3234, 3268,	<code>\tl_set_eq:NN</code>	406, 4414, 5456, 5467
	3330, 3351, 3397, 4224, 4820, 4902,	<code>\tl_show:N</code>	4377
	4924, 4982, 4993, 5036, 5465, 5634	<code>\tl_tail:N</code>	3619, 3621
<code>\tl_if_empty:nTF</code>		<code>\tl_tail:n</code>	
	726, 742, 965, 1257, 1534, 1543,		1857, 1858, 1908, 1909, 1954, 1955
	1706, 2403, 2695, 2869, 3141, 5113	<code>\tl_use:N</code>	290,
<code>\tl_if_empty_p:N</code>	2563, 4190, 4261,		301, 312, 748, 912, 917, 947, 949, 953
	4639, 4935, 4949, 5088, 5098, 5102	<code>\l_tmpa_tl</code>	921, 945, 1841,
<code>\tl_if_empty_p:n</code>	1368, 1401,		1855, 1857, 1886, 1897, 1906, 1908,
	2511, 2775, 3274, 3299, 3530, 3531,		1928, 1939, 1948, 1954, 3402, 3403
	3540, 3541, 3566, 3567, 3582, 3597	<code>\l_tmpb_tl</code>	1803, 1810, 1813, 1817,
<code>\tl_if_eq:NNTF</code>	3501, 3524, 3831		1846, 1855, 1858, 1859, 1866, 1891,
<code>\tl_if_eq:NnTF</code>			1899, 1906, 1909, 1910, 1917, 1933,
	1764, 3444, 3476, 3651, 3654, 3679,		1941, 1948, 1951, 1952, 1955, 1963
	3682, 3799, 3847, 4884, 5134, 5463		
<code>\tl_if_eq:nnTF</code>	1754, 1766, 1778,		
	1788, 1854, 1905, 1947, 3643, 5130,		
	5136, 5158, 5162, 5177, 5557, 5565		
<code>\tl_if_exist:NnTF</code>	325, 335,		
	345, 352, 359, 370, 386, 396, 715, 5455		

			U
<code>\upshape</code>			5540
use commands:			
<code>\use:N</code>	25, 28, 794, 4193, 4268, 4642, 5291		
<code>\UseHook</code>	1840, 1885, 1927		

V	
<code>\value</code>	5339, 5358
<code>\verbfootnote</code>	129, 130
Z	
<code>\Z</code>	1874
<code>\zcDeclareLanguage</code>	12, 25, 723, 5711, 5916, 6370, 6588, 6817, 7031
<code>\zcDeclareLanguageAlias</code>	25, 740, 5712, 5713, 5714, 5715, 5716, 5717, 5718, 5919, 5920, 5921, 5922, 5923, 5924, 6371, 6372, 6373, 6374, 6589, 6590, 6591
<code>\zcLanguageSetup</code> 20, 29, 30, 67, 72, 73, 2832	
<code>\zcpageref</code>	84, 3423
<code>\zceref</code>	20, 64, 66, 82, 84–86, 92, 94, 134, 3375, 3428
<code>\zcRefTypeSetup</code>	20, 67, 2570, 5539
<code>\zcsetup</code>	20, 54, 64, 66, 2565
<code>\zlabel</code> 128, 132, 133, 135, 136, 5401, 5634	
zrefcheck commands:	
<code>\zrefcheck_zceref_beg_label:</code> ..	3388
<code>\zrefcheck_zceref_end_label_-</code> maybe:	3410
<code>\zrefcheck_zceref_run_checks_on_-</code> labels:n	3411
zrefclever commands:	
<code>\zrefclever_language_if_declared:nTF</code>	721
<code>\zrefclever_language_varname:n</code> ..	711
<code>\l_zrefclever_ref_language_tl</code> ..	707
zrefclever internal commands:	
<code>\l_zrefclever_abbrev_bool</code>	3723, 3910, 4906
<code>\l_zrefclever_amsmath_subequations_-</code> bool	5497, 5511, 5535
<code>\l_zrefclever_breqn_dgroup_bool</code>	5584, 5598, 5613
<code>\l_zrefclever_cap_bool</code>	3723, 3906, 4894
<code>\l_zrefclever_capfirst_bool</code> ...	1975, 1978, 4896
<code>_zrefclever_compat_module:nn</code> ..	63, 2436, 5298, 5316, 5377, 5449, 5476, 5545, 5580, 5620, 5640, 5667, 5690
<code>_zrefclever_counter_reset_by:n</code> ..	6, 61, 62, 57, 59, 61, 65, 5505, 5592
<code>_zrefclever_counter_reset_by_-</code> aux:nn	72, 75
<code>_zrefclever_counter_reset_by_-</code> auxi:nnn	82, 86
<code>\l_zrefclever_counter_resetby_-</code> prop	5, 61, 68, 69, 2366, 2378
<code>\l_zrefclever_counter_resettters_-</code> seq ..	5, 61, 62, 71, 2340, 2347, 2350
<code>\l_zrefclever_counter_type_prop</code>	4, 60, 34, 37, 2311, 2323
<code>\l_zrefclever_current_counter_-</code> tl	3, 5, 62, 20, 24, 25, 35, 38, 40, 45, 46, 94, 2390, 2393
<code>\l_zrefclever_current_language_-</code> tl	24, 54, 705, 2068, 2074, 2078, 2094, 2131
<code>\l_zrefclever_endrangefunc_tl</code> ..	3723, 3898, 4190, 4191, 4193, 4261, 4262, 4268, 4639, 4640, 4642
<code>\l_zrefclever_endrangeprop_tl</code> ..	47, 1738, 1748, 3723, 3902
<code>_zrefclever_extract:nnn</code>	11, 278, 1843, 1848, 1888, 1893, 1930, 1935, 3572, 3574, 3587, 3604, 3692, 3694, 5141, 5143, 5147, 5149, 5167, 5169, 5173, 5175
<code>_zrefclever_extract_default:Nnnn</code>	11, 266, 1742, 1803, 1810, 1820, 1827, 3485, 3496, 3498, 3509, 3512, 3515, 3517, 3805, 3808
<code>_zrefclever_extract_unexp:nnn</code> ..	11, 122, 272, 1756, 1760, 1768, 1772, 1780, 1784, 1790, 1794, 4340, 4696, 4699, 4711, 4737, 4777, 4790, 4843, 4849, 4864, 5120, 5123, 5124, 5131, 5132, 5137, 5138, 5159, 5160, 5163, 5164, 5179, 5183, 5558, 5566
<code>_zrefclever_extract_url_-</code> unexp:n	4336, 4695, 4736, 4773, 4839, 5117
<code>_zrefclever_get_enclosing_-</code> counters_value:n ..	5, 6, 55, 60, 93
<code>_zrefclever_get_endrange_-</code> pagecomp:nnN	1880, 1921
<code>_zrefclever_get_endrange_-</code> pagecomptwo:nnN	1922, 1967
<code>_zrefclever_get_endrange_-</code> property:nnN	45, 1736, 1834
<code>_zrefclever_get_endrange_-</code> stripprefix:nnN	1835, 1870
<code>_zrefclever_get_ref:nnN</code>	112, 113, 3998, 4021, 4029, 4047, 4060, 4064, 4099, 4108, 4130, 4138, 4145, 4169, 4183, 4210, 4252, 4285, 4320, 4508, 4531, 4543, 4577, 4599, 4608, 4628, 4659, 4681
<code>_zrefclever_get_ref_endrange:nnN</code>	45, 113, 4200, 4275, 4649, 4721
<code>_zrefclever_get_ref_first:</code> ...	112, 113, 117, 4312, 4362, 4759

```

\__zrefclever_get_rf_opt_bool:nN 125
\__zrefclever_get_rf_opt_-
  bool:nnnnN 20, 3903, 3907, 3911, 5265
\__zrefclever_get_rf_opt_-
  seq:nnnN ..... 20, 124,
3915, 3919, 3923, 3927, 3931, 3935,
3939, 3943, 3947, 3951, 4951, 5232
\__zrefclever_get_rf_opt_tl:nnnN
  ..... 20, 22, 45, 124, 3399, 3770,
3774, 3778, 3867, 3871, 3875, 3879,
3883, 3887, 3891, 3895, 3899, 5199
\__zrefclever_hyperlink:nnn .....
121, 4334, 4694, 4735, 4771, 4837, 5111
\l__zrefclever_hyperlink_bool ...
1995, 2002, 2007, 2012, 2024, 2030,
2037, 3427, 4687, 4728, 4829, 5086
\l__zrefclever_hyperref_warn_-
  bool ... 1996, 2003, 2008, 2013, 2028
\__zrefclever_if_class_loaded:n 107
\__zrefclever_if_class_loaded:nTF
  ..... 5379
\__zrefclever_if_package_-
  loaded:n ..... 107
\__zrefclever_if_package_-
  loaded:nTF ..... 2022,
2066, 2072, 2275, 5318, 5478, 5485,
5547, 5582, 5622, 5642, 5669, 5692
\__zrefclever_is_integer_rgx:n 1871
\__zrefclever_is_integer_rgx:nTF
  ..... 1897, 1899, 1939, 1941
\g__zrefclever_koma_captionofbeside_-
  capttype_tl ..... 5458
\g__zrefclever_koma_capttype_tl ...
  ..... 5460, 5465, 5468, 5471
\l__zrefclever_label_a_tl .....
. 91, 3704, 3788, 3807, 3820, 3842,
3844, 3850, 3853, 3859, 3976, 3998,
4021, 4029, 4064, 4130, 4145, 4195,
4201, 4210, 4242, 4252, 4270, 4276,
4285, 4438, 4442, 4452, 4465, 4482,
4508, 4544, 4608, 4644, 4650, 4659
\l__zrefclever_label_b_tl .....
..... 91, 3704,
3791, 3796, 3810, 3822, 3827, 4442
\l__zrefclever_label_count_int ...
..... 92, 3701, 3761,
3865, 3968, 4421, 4447, 4675, 4938
\l__zrefclever_label_enclval_a_-
  tl ..... 3431, 3509, 3511, 3566,
3582, 3602, 3614, 3618, 3619, 3626
\l__zrefclever_label_enclval_b_-
  tl ..... 3431, 3512, 3514, 3567,
3589, 3597, 3616, 3620, 3621, 3628
\l__zrefclever_label_extdoc_a_tl
  .. 3431, 3515, 3525, 3530, 3540, 3553
\l__zrefclever_label_extdoc_b_tl
  .. 3431, 3517, 3526, 3531, 3541, 3552
\l__zrefclever_label_type_a_tl ..
..... 3400, 3431, 3486, 3488,
3491, 3497, 3502, 3651, 3679, 3771,
3775, 3779, 3801, 3806, 3832, 3847,
3868, 3872, 3876, 3880, 3884, 3888,
3892, 3896, 3900, 3904, 3908, 3912,
3916, 3920, 3924, 3928, 3932, 3936,
3940, 3944, 3948, 3952, 3978, 4454
\l__zrefclever_label_type_b_tl ..
..... 3431, 3499,
3503, 3654, 3682, 3802, 3809, 3833
\__zrefclever_label_type_put_-
  new_right:n .... 85, 86, 3447, 3483
\l__zrefclever_label_types_seq ...
..... 86, 3440, 3443, 3487, 3490, 3677
\__zrefclever_labels_in_sequence:nn
  ..... 47, 92, 122, 4240, 4441, 5128
\l__zrefclever_lang_decl_case_tl
  ..... 552, 933, 936, 979, 984, 1323, 1340,
2847, 2850, 2883, 2888, 3234, 3251
\l__zrefclever_lang_declension_-
  seq ..... 552,
812, 813, 814, 828, 832, 839, 930,
931, 932, 935, 972, 978, 983, 2844,
2845, 2846, 2849, 2876, 2882, 2887
\l__zrefclever_lang_gender_seq ...
..... 552, 849, 850, 851, 866, 943,
944, 993, 1008, 2857, 2858, 2897, 2912
\__zrefclever_language_if_-
  declared:n ..... 25, 722
\__zrefclever_language_if_-
  declared:n(TF) ..... 24
\__zrefclever_language_if_-
  declared:nTF 287, 298, 309, 713,
728, 744, 804, 908, 2105, 2142, 2835
\__zrefclever_language_varname:n
  ..... 24, 290,
301, 312, 709, 712, 715, 731, 732,
746, 747, 748, 912, 917, 947, 949, 953
\l__zrefclever_last_of_type_bool
  ..... 92, 3698, 3818,
3823, 3824, 3828, 3834, 3835, 3958
\l__zrefclever_lastsep_tl . 3723,
3882, 4028, 4063, 4107, 4144, 4182
\l__zrefclever_link_star_bool ...
.. 3382, 3421, 4688, 4729, 4830, 5087
\l__zrefclever_listsep_tl .....
... 3723, 3878, 4059, 4137, 4507,
4530, 4542, 4576, 4598, 4607, 4627

```



```

\g_zrefclever_loaded_langfiles_-
  seq ..... 903, 911, 946, 952
\__zrefclever_ltxlabel:n .....
  ..... 132, 5480, 5490, 5495
\l_zrefclever_main_language_tl .
  ..... 24,
  54, 706, 2069, 2075, 2079, 2100, 2137
\__zrefclever_mathtools_showonlyrefs:n
  ..... 3416, 5552
\l_zrefclever_mathtools_-
  showonlyrefs_bool 3414, 5544, 5551
\__zrefclever_memoir_both_-
  labels: .....
  .. 5395, 5406, 5408, 5410, 5414, 5418
\l_zrefclever_memoir_footnote_-
  type_tl .... 5425, 5426, 5428, 5432
\__zrefclever_memoir_label_and_-
  zlabel:n ..... 5398, 5403
\__zrefclever_memoir_orig_-
  label:n ..... 5397, 5400
\__zrefclever_name_default: ....
  ..... 4677, 4812
\l_zrefclever_name_format_-
  fallback_tl ..... 3710, 4913,
  4917, 4982, 5031, 5043, 5045, 5063
\l_zrefclever_name_format_tl ...
  ... 3710, 4899, 4900, 4903, 4904,
  4914, 4915, 4988, 4995, 4996, 5004,
  5012, 5022, 5039, 5040, 5053, 5073
\l_zrefclever_name_in_link_bool
  ..... 114,
  117, 3710, 4330, 4764, 5091, 5107, 5108
\l_zrefclever_namefont_tl 3723,
  3890, 4333, 4349, 4782, 4802, 4817
\l_zrefclever_nameinlink_str ...
  ..... 2040, 2045, 2047,
  2049, 2051, 5089, 5095, 5097, 5101
\l_zrefclever_namesep_tl .....
  .. 3723, 3870, 4785, 4805, 4813, 4821
\l_zrefclever_next_is_same_bool
  ..... 92, 122, 3716,
  4435, 4468, 4485, 4491, 5152, 5190
\l_zrefclever_next_maybe_range_-
  bool .....
  . 92, 122, 3716, 4234, 4247, 4434,
  4461, 4474, 5144, 5151, 5170, 5188
\l_zrefclever_noabbrev_first_-
  bool ..... 1982, 1985, 4910
\g_zrefclever_nocompat_bool ...
  ..... 2397, 2404, 2440
\l_zrefclever_nocompat_bool ... 63
\g_zrefclever_nocompat_modules_-
  seq 2398, 2408, 2411, 2432, 2441, 2442
\l_zrefclever_nocompat_modules_-
  seq ..... 63
\l_zrefclever_nudge_comptosing_-
  bool ... 2168, 2198, 2207, 2213, 4934
\l_zrefclever_nudge_enabled_-
  bool ..... 2166, 2176, 2178,
  2182, 2183, 2188, 2189, 4406, 4920
\l_zrefclever_nudge_gender_bool
  ..... 2170, 2199, 2209, 2214, 4948
\l_zrefclever_nudge_multitype_-
  bool ... 2167, 2197, 2205, 2212, 4407
\l_zrefclever_nudge_singular_-
  bool ..... 2169, 2225, 4922
\__zrefclever_opt_bool_get:NN(TF)
  ..... 19
\__zrefclever_opt_bool_get:NNTF .
  ... 533, 5268, 5273, 5278, 5283, 5288
\__zrefclever_opt_bool_gset_-
  false:N .....
  .... 18, 499, 1481, 1498, 3353, 3361
\__zrefclever_opt_bool_gset_-
  true:N 18, 499, 1443, 1460, 3332, 3340
\__zrefclever_opt_bool_if:N(TF) . 19
\__zrefclever_opt_bool_if:N(TF) ...
  ..... 544, 877
\__zrefclever_opt_bool_if_-
  set:N(TF) ..... 18
\__zrefclever_opt_bool_if_-
  set:N(TF) ..... 483,
  535, 546, 1436, 1452, 1474, 1490
\__zrefclever_opt_bool_set_-
  false:N ..... 18, 499, 2540, 2811
\__zrefclever_opt_bool_set_-
  true:N ..... 18, 499, 2535, 2802
\__zrefclever_opt_bool_unset:N .
  ..... 17, 472, 2545, 2820
\__zrefclever_opt_seq_get:NN(TF) 17
\__zrefclever_opt_seq_get:NNTF .
  ... 461, 807, 844, 925, 938, 2839,
  2852, 5235, 5240, 5245, 5250, 5255
\__zrefclever_opt_seq_gset_-
  clist_split:Nn .....
  .... 16, 413, 1365, 1398, 3271, 3296
\__zrefclever_opt_seq_gset_eq:NN
  16, 413, 1374, 1407, 2920, 3280, 3305
\__zrefclever_opt_seq_if_-
  set:N(TF) ..... 17
\__zrefclever_opt_seq_if_set:N(TF)
  ..... 445, 463, 1016, 1358, 1390
\__zrefclever_opt_seq_set_clist_-
  split:Nn ..... 16, 413, 2508, 2772
\__zrefclever_opt_seq_set_eq:NN .
  ..... 16, 413, 2514, 2778

```



```

\__zrefclever_opt_seq_unset:N ...
..... 16, 434, 2503, 2764
\__zrefclever_opt_tl_clear:N ...
.. 14, 323, 1634, 1639, 1654, 1669,
1684, 2623, 2628, 2643, 2658, 2673
\__zrefclever_opt_tl_cset_-
  fallback:nn ..... 1513, 1520
\__zrefclever_opt_tl_gclear:N ...
.. 14, 323, 2998, 3004, 3012, 3019,
3040, 3056, 3077, 3093, 3114, 3130
\__zrefclever_opt_tl_gclear_if_-
  new:N .....
.. 15, 382, 1114, 1120, 1128, 1135,
1156, 1172, 1193, 1209, 1230, 1246
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF ...
402, 4984, 4999, 5018, 5027, 5048,
5058, 5202, 5207, 5212, 5217, 5222
\__zrefclever_opt_tl_gset:N .... 14
\__zrefclever_opt_tl_gset:Nn ...
..... 323, 2943, 2967, 2975,
3033, 3048, 3070, 3085, 3107, 3122,
3155, 3162, 3171, 3179, 3236, 3246
\__zrefclever_opt_tl_gset_if_-
  new:Nn 15, 382, 1057, 1082, 1091,
1149, 1164, 1186, 1201, 1223, 1238,
1271, 1278, 1287, 1295, 1325, 1335
\__zrefclever_opt_tl_if_set:N(TF)
..... 15
\__zrefclever_opt_tl_if_set:NNTF .
..... 368, 384, 394, 404
\__zrefclever_opt_tl_set:N ..... 14
\__zrefclever_opt_tl_set:Nn 323,
1648, 1663, 1678, 1718, 1724, 2459,
2605, 2637, 2652, 2667, 2707, 2714
\__zrefclever_opt_tl_unset:N 14,
357, 1693, 1698, 2455, 2598, 2682, 2687
\__zrefclever_opt_var_set_bool:n
..... 13, 14, 321, 328,
329, 330, 338, 339, 340, 362, 363,
364, 372, 374, 422, 423, 424, 439,
440, 441, 449, 451, 477, 478, 479,
487, 489, 504, 505, 506, 514, 515, 516
\__zrefclever_opt_varname_-
  fallback:nn .....
..... 13, 319, 1516, 5223, 5256, 5289
\__zrefclever_opt_varname_-
  general:nn ..... 12,
280, 1636, 1641, 1650, 1656, 1665,
1671, 1680, 1686, 1695, 1700, 1720,
1726, 2456, 2460, 2504, 2515,
2536, 2541, 2546, 5203, 5236, 5269
\__zrefclever_opt_varname_lang_-
  default:nnn ..... 12, 296,
1059, 1084, 1116, 1122, 1151, 1158,
1188, 1195, 1225, 1232, 1273, 1280,
1360, 1376, 1438, 1445, 1476, 1483,
2945, 2969, 3000, 3006, 3035, 3042,
3072, 3079, 3109, 3116, 3157, 3164,
3282, 3334, 3355, 5218, 5251, 5284
\__zrefclever_opt_varname_lang_-
  type:nnnn .....
..... 13, 307, 1018, 1027, 1035,
1093, 1130, 1137, 1166, 1174, 1203,
1211, 1240, 1248, 1289, 1297, 1327,
1337, 1392, 1409, 1454, 1462, 1492,
1500, 2922, 2977, 3014, 3021, 3050,
3058, 3087, 3095, 3124, 3132, 3173,
3181, 3238, 3248, 3307, 3342, 3363,
5001, 5050, 5060, 5213, 5246, 5279
\__zrefclever_opt_varname_-
  language:nnn ..... 12,
285, 760, 765, 775, 780, 791, 796,
809, 846, 879, 927, 940, 2841, 2854
\__zrefclever_opt_varname_-
  type:nnn ..... 12, 282, 2600,
2607, 2625, 2630, 2639, 2645, 2654,
2660, 2669, 2675, 2684, 2689, 2709,
2716, 2766, 2780, 2804, 2813, 2822,
4986, 5020, 5029, 5208, 5241, 5274
\__zrefclever_orig_ltxlabel:n ...
..... 5482, 5489, 5494
\g_zrefclever_page_format_tl ...
..... 7, 97, 102, 105
\l_zrefclever_pairsep_tl .....
..... 3723, 3874, 3997,
4020, 4046, 4098, 4129, 4168, 4251
\__zrefclever_process_language_-
  settings: ..... 57, 58, 802, 3384
\__zrefclever_prop_put_non_-
  empty:Nnn .... 42, 1532, 2322, 2377
\__zrefclever_provide_langfile:n
..... 20, 30, 31, 83, 904, 2111, 3383
\l_zrefclever_range_beg_is_-
  first_bool ..... 3716,
3766, 4085, 4121, 4155, 4426,
4463, 4519, 4564, 4586, 4615, 4668
\l_zrefclever_range_beg_label_-
  tl ..... 92,
3716, 3759, 4048, 4061, 4100, 4109,
4139, 4170, 4184, 4194, 4419, 4464,
4481, 4532, 4578, 4600, 4629, 4643
\l_zrefclever_range_count_int ..
..... 92,
3716, 3764, 4010, 4074, 4424, 4467,
4478, 4484, 4490, 4498, 4557, 4670
\l_zrefclever_range_end_ref_tl .
... 3716, 3760, 4196, 4202, 4271,

```

4277, 4420, 4466, 4483, 4645, 4651
 \l_zrefclever_range_same_count_-
 int 92,
 3716, 3765, 3988, 4039, 4075, 4425,
 4469, 4486, 4492, 4537, 4558, 4671
 \l_zrefclever_rangeseq_tl
 3723, 3886,
 4199, 4209, 4274, 4284, 4648, 4658
 \l_zrefclever_rangetopair_bool .
 3723, 3914, 4235
 \l_zrefclever_ref_count_int ...
 3701, 3763,
 4016, 4094, 4164, 4422, 4455, 4504,
 4527, 4539, 4573, 4595, 4604, 4624
 \l_zrefclever_ref_decl_case_tl .
 27, 816, 821, 822, 826, 829,
 833, 837, 840, 885, 888, 890, 2152,
 2162, 4993, 4997, 5036, 5041, 5046
 _zrefclever_ref_default:
 .. 4677, 4718, 4724, 4762, 4806, 4872
 \l_zrefclever_ref_gender_tl ...
 28, 853, 859,
 860, 864, 867, 872, 873, 892, 898,
 899, 2171, 2235, 4949, 4957, 4963, 4971
 \l_zrefclever_ref_language_tl ..
 24, 27, 54, 704, 708, 805,
 810, 820, 838, 847, 857, 871, 880,
 889, 896, 2093, 2099, 2104, 2112,
 2130, 2136, 2141, 3383, 3401, 3772,
 3776, 3780, 3869, 3873, 3877, 3881,
 3885, 3889, 3893, 3897, 3901, 3905,
 3909, 3913, 3917, 3921, 3925, 3929,
 3933, 3937, 3941, 3945, 3949, 3953,
 4953, 4965, 4976, 5002, 5051, 5061
 \l_zrefclever_ref_property_tl ..
 42, 47, 1538,
 1547, 1554, 1556, 1740, 1764, 1808,
 1825, 1837, 1844, 1849, 1882, 1889,
 1894, 1924, 1931, 1936, 3476, 3799,
 3854, 3858, 4683, 4768, 4825, 5134
 \l_zrefclever_ref_propserity_tl 3444
 \l_zrefclever_ref_typeset_font_-
 tl 2239, 2241, 3394
 \l_zrefclever_refbounds_first_-
 pb_seq 3738,
 3926, 4002, 4052, 4124, 4175, 4256
 \l_zrefclever_refbounds_first_-
 rb_seq . 3738, 3930, 4158, 4290, 4619
 \l_zrefclever_refbounds_first_-
 seq 3738, 3918, 4301, 4522, 4568, 4590
 \l_zrefclever_refbounds_first_-
 sg_seq . 3738, 3922, 3980, 3991, 4088
 \l_zrefclever_refbounds_last_-
 pe_seq 3738, 3950,
 3999, 4022, 4049, 4101, 4131, 4253
 \l_zrefclever_refbounds_last_-
 re_seq
 .. 3738, 3954, 4203, 4211, 4278, 4286
 \l_zrefclever_refbounds_last_-
 seq 3738, 3946, 4030, 4065, 4110, 4146
 \l_zrefclever_refbounds_mid_rb_-
 seq ... 3738, 3938, 4171, 4185, 4630
 \l_zrefclever_refbounds_mid_re_-
 seq 3738, 3942, 4652, 4660
 \l_zrefclever_refbounds_mid_seq
 3738, 3934, 4062, 4140,
 4509, 4533, 4545, 4579, 4601, 4609
 \l_zrefclever_reffont_tl
 3723, 3894, 4692, 4708,
 4733, 4748, 4787, 4796, 4834, 4861
 \g_zrefclever_rf_opts_bool_-
 maybe_type_specific_seq
 52, 557, 1427, 2528, 2795, 3323
 \g_zrefclever_rf_opts_seq_-
 refbounds_seq
 557, 1349, 2494, 2755, 3261
 \g_zrefclever_rf_opts_tl_maybe_-
 type_specific_seq 557, 1073, 2958
 \g_zrefclever_rf_opts_tl_not_-
 type_specific_seq
 557, 1048, 2577, 2934
 \g_zrefclever_rf_opts_tl_-
 reference_seq 557, 2446
 \g_zrefclever_rf_opts_tl_type_-
 names_seq 557, 1310, 3221
 \g_zrefclever_rf_opts_tl_-
 typesetup_seq 557, 2589
 \l_zrefclever_setup_language_tl
 552, 733, 761, 766, 776,
 781, 792, 797, 923, 975, 982, 996,
 1013, 1019, 1028, 1036, 1060, 1085,
 1094, 1117, 1123, 1131, 1138, 1152,
 1159, 1167, 1175, 1189, 1196, 1204,
 1212, 1226, 1233, 1241, 1249, 1274,
 1281, 1290, 1298, 1328, 1338, 1361,
 1377, 1393, 1410, 1439, 1446, 1455,
 1463, 1477, 1484, 1493, 1501, 2838,
 2879, 2886, 2900, 2917, 2923, 2946,
 2970, 2978, 3001, 3007, 3015, 3022,
 3036, 3043, 3051, 3059, 3073, 3080,
 3088, 3096, 3110, 3117, 3125, 3133,
 3158, 3165, 3174, 3182, 3239, 3249,
 3283, 3308, 3335, 3343, 3356, 3364
 \l_zrefclever_setup_type_tl ...
 552, 924, 966, 967, 999, 1020, 1029,
 1037, 1055, 1080, 1095, 1112, 1132,
 1139, 1147, 1168, 1176, 1184, 1205,
 1213, 1221, 1242, 1250, 1269, 1291,

1299, 1317, 1329, 1339, 1356, 1394,
1411, 1434, 1456, 1464, 1472, 1494,
1502, 2572, 2574, 2601, 2608, 2626,
2631, 2640, 2646, 2655, 2661, 2670,
2676, 2685, 2690, 2710, 2717, 2767,
2781, 2805, 2814, 2823, 2837, 2870,
2871, 2903, 2924, 2941, 2965, 2979,
2996, 3016, 3023, 3031, 3052, 3060,
3068, 3089, 3097, 3105, 3126, 3134,
3153, 3175, 3183, 3228, 3240, 3250,
3268, 3309, 3330, 3344, 3351, 3365
\l_zrefclever_sort_decided_bool
..... 3437, 3520, 3534, 3544,
3548, 3560, 3570, 3585, 3600, 3624
_zrefclever_sort_default:nn ...
..... 87, 3478, 3494
_zrefclever_sort_default_-
different_types:nn
..... 44, 85, 90, 3505, 3637
_zrefclever_sort_default_same_-
type:nn 85, 87, 3504, 3507
_zrefclever_sort_labels:
..... 85-87, 91, 3392, 3441
_zrefclever_sort_page:nn
..... 91, 3477, 3689
\l_zrefclever_sort_prior_a_int .
..... 3438,
3639, 3645, 3646, 3652, 3662, 3670
\l_zrefclever_sort_prior_b_int .
..... 3438,
3640, 3647, 3648, 3655, 3663, 3671
\l_zrefclever_tlastsep_tl
..... 3723, 3781, 4400
\l_zrefclever_tlistsep_tl
..... 3723, 3777, 4371
\l_zrefclever_tpairsep_tl
..... 3723, 3773, 4393
\l_zrefclever_type_count_int ...
. 92, 117, 3701, 3762, 4368, 4370,
4383, 4408, 4423, 4897, 4909, 5104
\l_zrefclever_type_first_label_-
tl 92,
114, 3704, 3757, 3975, 4226, 4237,
4241, 4269, 4320, 4337, 4341, 4417,
4451, 4761, 4767, 4774, 4778, 4791,
4824, 4840, 4844, 4850, 4865, 4878
\l_zrefclever_type_first_label_-
type_tl ... 92, 117, 3704, 3758,
3977, 4230, 4418, 4453, 4885, 4928,
4944, 4952, 4964, 4970, 4987, 5003,
5013, 5021, 5030, 5052, 5062, 5074
\l_zrefclever_type_first_-
refbounds_seq
... 3738, 3979, 3990, 4001, 4051,
4087, 4123, 4157, 4174, 4255, 4289,
4300, 4321, 4521, 4567, 4589, 4618,
4788, 4789, 4793, 4797, 4836, 4848,
4853, 4856, 4862, 4863, 4867, 4868
\l_zrefclever_type_first_-
refbounds_set_bool 3738,
3767, 3981, 3992, 4003, 4054, 4090,
4126, 4160, 4177, 4257, 4291,
4298, 4427, 4524, 4570, 4592, 4621
\l_zrefclever_type_name_gender_-
seq ... 3710, 4954, 4956, 4959, 4974
\l_zrefclever_type_name_-
missing_bool
.. 3710, 4810, 4881, 4888, 5010, 5070
_zrefclever_type_name_setup: ..
..... 20, 22, 114, 4307, 4876
\l_zrefclever_type_name_tl
..... 114, 117,
3710, 4344, 4350, 4783, 4803, 4818,
4820, 4880, 4887, 4991, 5007, 5009,
5025, 5034, 5056, 5066, 5068, 5088
\l_zrefclever_typeset_compress_-
bool 1616, 1619, 4436
\l_zrefclever_typeset_labels_-
seq 91, 3698, 3753, 3787, 3789, 3795
\l_zrefclever_typeset_last_bool
..... 92, 3698,
3784, 3785, 3792, 3817, 4380, 5103
\l_zrefclever_typeset_name_bool
.. 1565, 1572, 1577, 1582, 4309, 4325
\l_zrefclever_typeset_queue_-
curr_tl 92,
94, 112, 117, 3704, 3756, 3995,
4018, 4026, 4044, 4057, 4096,
4105, 4127, 4135, 4142, 4166, 4180,
4197, 4207, 4224, 4249, 4272, 4282,
4311, 4318, 4328, 4361, 4377, 4388,
4394, 4401, 4415, 4416, 4505, 4528,
4540, 4574, 4596, 4605, 4625, 4646,
4656, 4902, 4924, 4935, 5098, 5102
\l_zrefclever_typeset_queue_-
prev_tl . 92, 3704, 3755, 4372, 4414
\l_zrefclever_typeset_range_-
bool ... 1750, 1968, 1971, 3391, 4222
\l_zrefclever_typeset_ref_bool .
.. 1564, 1571, 1576, 1581, 4309, 4315
_zrefclever_typeset_refs:
..... 91, 93, 94, 3395, 3751
_zrefclever_typeset_refs_last_-
of_type: 98, 112, 114, 117, 3960, 3965
_zrefclever_typeset_refs_not_-
last_of_type:
..... 92, 99, 112, 122, 3962, 4430

\l_zrefclever_typeset_sort_bool	\l_zrefclever_zcref_note_tl ...
..... 1592, 1595, 3390 2260, 2263, 3397, 3404
\l_zrefclever_typesort_seq	\l_zrefclever_zcref_with_check_
. 44, 90, 1601, 1606, 1607, 1613, 3641	bool 2267, 2284, 3387, 3408
\l_zrefclever_verbose_testing_	_zrefclever_zcsetup:n 67, 2566, 2567, 5302,
bool 3750, 4376	5326, 5332, 5340, 5362, 5381, 5431,
_zrefclever_zcref:nnn 27, 56, 3376, 3377	5435, 5436, 5445, 5500, 5534, 5587,
_zrefclever_zcref:nnnn 82, 85, 3377	5612, 5624, 5636, 5651, 5671, 5694
\l_zrefclever_zcref_labels_seq .	\l_zrefclever_zrefcheck_
..... 85, 86, 3381,	available_bool 2266, 2279, 2291, 2303, 3386, 3407
3412, 3417, 3421, 3446, 3449, 3754	