

The zref-clever package^{*}

Code documentation

Gustavo Barros[†]

2023-06-19

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	8
4.1	Auxiliary	8
4.2	Messages	8
4.3	Data extraction	11
4.4	Option infra	12
4.5	Reference format	20
4.6	Languages	24
4.7	Language files	30
4.8	Options	43
5	Configuration	67
5.1	\zcsetup	67
5.2	\zcRefTypeSetup	68
5.3	\zcLanguageSetup	73
6	User interface	83
6.1	\zceref	83
6.2	\zcpageref	85
7	Sorting	85
8	Typesetting	92

^{*}This file describes v0.4.1, released 2023-06-19.

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

9	Compatibility	127
9.1	appendix	127
9.2	appendices	128
9.3	memoir	129
9.4	amsmath	131
9.5	mathtools	132
9.6	breqn	133
9.7	listings	134
9.8	enumitem	134
9.9	subcaption	135
9.10	subfig	136
10	Language files	136
10.1	Localization guidelines	136
10.2	English	139
10.3	German	143
10.4	French	151
10.5	Portuguese	156
10.6	Spanish	160
10.7	Dutch	164
10.8	Italian	169
	Index	173

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from l3candidates). We presume xparse (which made to the kernel in the 2020-10-01 release), and expl3 as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Also, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (ltxcmds), with implications to the hook we add to `\appendix` (by Phelype Oleinik at <https://tex.stackexchange.com/q/617905> and <https://github.com/latex3/latex2e/pull/699>). Second, the support for `\@currentcounter` has been improved, including `\footnote` and `amsmath` (by Frank Mittelbach and Ulrike Fischer at <https://github.com/latex3/latex2e/issues/687>). Finally, and critically, the new `label` hook introduced in the 2023-06-01 release, alongside the corresponding new hooks with arguments, just simplifies and improves label setting so much, by allowing `\zlabel` to be set with `\label`, that it is definitely a must for `zref-clever`, so we require that too. Hence we make the cut at this latter kernel release.

```
3 \newcommand{\zc@required@kernel@version}{2023-06-01}
```

```
4 \NeedsTeXFormat{LaTeX2e}
```

```

5 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
6 \IfFormatAtLeastTF{\zc@required@kernel@version}
7 {}
8 {%
9   \PackageError{zref-clever}{LaTeX kernel too old}
10  {%
11    'zref-clever' requires a LaTeX kernel \zc@required@kernel@version\space or newer.%
12    \MessageBreak Loading will abort!%
13  }%
14  \endinput
15 }%

Identify the package.

16 \ProvidesExplPackage {zref-clever} {2023-06-19} {0.4.1}
17 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

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

```

18 \RequirePackage { zref-base }
19 \RequirePackage { zref-user }
20 \RequirePackage { zref-abspage }
21 \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`.

```

22 \zref@newprop { zc@counter } { \l__zrefclever_current_counter_tl }
23 \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.

```

24 \zref@newprop { thecounter }
25 {
26   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
27   { \use:c { the \l__zrefclever_current_counter_tl } }
28   {
29     \cs_if_exist:cT { c@ \@currentcounter }
30     { \use:c { the \@currentcounter } }
31   }
32 }
33 \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`.

```

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

```

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

```

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

```

```

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

```

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

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

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

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

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

`__zrefclever_get_enclosing_counters_value:n`

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

```

\__zrefclever_get_enclosing_counters_value:n {\langle counter \rangle}

66 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
67 {
68   \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
69   {
70     { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
71     \__zrefclever_get_enclosing_counters_value:e
72     { \__zrefclever_counter_reset_by:n {#1} }
73   }
74 }

```

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).

```

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

```

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

`__zrefclever_counter_reset_by:n`

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

```

\__zrefclever_counter_reset_by:n {\langle counter \rangle}

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

```

```

87 {
88   \cs_if_exist:cT { c@ #2 }
89   {
90     \tl_if_empty:cF { c1@ #2 }
91     {
92       \tl_map_tokens:cn { c1@ #2 }
93       { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
94     }
95   }
96 }
97 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
98 {
99   \str_if_eq:nnT {#2} {#3}
100   { \tl_map_break:n { \seq_map_break:n {#1} } }
101 }

```

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

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

```

102 \zref@newprop { zc@enclval }
103 {
104   \__zrefclever_get_enclosing_counters_value:e
105   \l__zrefclever_current_counter_tl
106 }
107 \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

108 \int_new:N \g__zrefclever_page_format_int

```

```

109 \tl_new:N \g__zrefclever_prev_page_format_tl
110 \AddToHook { shipout / before }
111 {
112   \tl_if_eq:NNF \g__zrefclever_prev_page_format_tl \thepage
113   {
114     \int_gincr:N \g__zrefclever_page_format_int
115     \tl_gset_eq:NN \g__zrefclever_prev_page_format_tl \thepage
116   }
117 }
118 \zref@newprop* { zc@pgfmt } { \int_use:N \g__zrefclever_page_format_int }
119 \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.

```

120 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
121 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
122 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
123 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

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

4.2 Messages

```

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

```



```

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

```

```

198 \msg_new:nnn { zref-clever } { gender-not-declared }
199 { Language~'#1'~has-no~'#2'~gender-declared~\msg_line_context:. }
200 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
201 {
202   Gender~mismatch~for~type~'#1'~\msg_line_context:~
203   You've-specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
204 }
205 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
206 {
207   You've-specified~'g=#1'~\msg_line_context:~
208   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
209 }
210 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
211 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:. }
212 \msg_new:nnn { zref-clever } { option-document-only }
213 { Option~'#1'~is-only-available-after~\iow_char:N\begin\{document\}. }
214 \msg_new:nnn { zref-clever } { langfile-loaded }
215 { Loaded~'#1'~language~file. }
216 \msg_new:nnn { zref-clever } { zref-property-undefined }
217 {
218   Option~'ref=#1'~requested~\msg_line_context:~
219   But~the~property~'#1'~is~not~declared,~falling-back-to~'default'.
220 }
221 \msg_new:nnn { zref-clever } { endrange-property-undefined }
222 {
223   Option~'endrange=#1'~requested~\msg_line_context:~
224   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
225 }
226 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
227 {
228   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:~
229   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
230   '\iow_char:N\zcref'.
231 }
232 \msg_new:nnn { zref-clever } { missing-hyperref }
233 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
234 \msg_new:nnn { zref-clever } { option-preamble-only }
235 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
236 \msg_new:nnn { zref-clever } { unknown-compat-module }
237 {
238   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
239   Nothing~to~do.
240 }
241 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
242 {
243   The~value~of~option~'#1'~must~be~a~comma~separated~list~
244   of~four~items.~We~received~'#2'~items~\msg_line_context:~
245   Option~not~set.
246 }
247 \msg_new:nnn { zref-clever } { missing-zref-check }
248 {
249   Option~'check'~requested~\msg_line_context:~
250   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
251 }

```

```

252 \msg_new:nnn { zref-clever } { zref-check-too-old }
253 {
254   Option~'check'~requested~\msg_line_context:..~
255   But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
256 }
257 \msg_new:nnn { zref-clever } { missing-type }
258 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
259 \msg_new:nnn { zref-clever } { missing-property }
260 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
261 \msg_new:nnn { zref-clever } { missing-name }
262 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
263 \msg_new:nnn { zref-clever } { single-element-range }
264 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
265 \msg_new:nnn { zref-clever } { compat-package }
266 { Loaded~support~for~'#1'~package. }
267 \msg_new:nnn { zref-clever } { compat-class }
268 { Loaded~support~for~'#1'~documentclass. }
269 \msg_new:nnn { zref-clever } { option-deprecated }
270 {
271   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
272   Use~'#2'~instead.
273 }
274 \msg_new:nnn { zref-clever } { load-time-options }
275 {
276   'zref-clever'~does~not~accept~load~time~options.~
277   To~configure~package~options,~use~'\iow_char:N\zcsetup'.
278 }

```

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 }
{ \langle label \rangle } { \langle prop \rangle } { \langle default \rangle }

279 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
280 {
281   \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
282   { \zref@extractdefault {#2} {#3} {#4} }
283 }
284 \cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn , Nnvn }

```

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

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

```

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

285 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
286 {

```

```

287 \exp_args:NNo \exp_args:No
288 \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
289 }
290 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End of definition for __zrefclever_extract_unexp:nnn.)

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

```

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

291 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
292 { \zref@extractdefault {#1} {#2} {#3} }

```

(End of definition for __zrefclever_extract:nnn.)

4.4 Option infra

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

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

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

```

\__zrefclever_opt_varname_general:nn {<option>} {<data type>}

293 \cs_new:Npn \__zrefclever_opt_varname_general:nn #1#2
294 { l__zrefclever_opt_general_ #1 _ #2 }

```

(End of definition for __zrefclever_opt_varname_general:nn.)

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

```

\__zrefclever_opt_varname_type:nnn {<ref type>} {<option>} {<data type>}

```

```

295 \cs_new:Npn \__zrefclever_opt_varname_type:nnn #1#2#3
296 { l__zrefclever_opt_type_ #1 _ #2 _ #3 }
297 \cs_generate_variant:Nn \__zrefclever_opt_varname_type:nnn { enn , een }

```

(End of definition for __zrefclever_opt_varname_type:nnn.)

__zrefclever_opt_varname_language:nnn

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

```

\__zrefclever_opt_varname_language:nnn {\langle lang \rangle} {\langle option \rangle} {\langle data type \rangle}

298 \cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
299 {
300   \__zrefclever_language_if_declared:nTF {#1}
301   {
302     g__zrefclever_opt_language_
303     \tl_use:c { \__zrefclever_language_varname:n {#1} }
304     _ #2 _ #3
305   }
306   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
307 }
308 \cs_generate_variant:Nn \__zrefclever_opt_varname_language:nnn { enn }

```

(End of definition for __zrefclever_opt_varname_language:nnn.)

__zrefclever_opt_varname_lang_default:nnn

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

```

\__zrefclever_opt_varname_lang_default:nnn {\langle lang \rangle} {\langle option \rangle} {\langle data type \rangle}

309 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
310 {
311   \__zrefclever_language_if_declared:nTF {#1}
312   {
313     g__zrefclever_opt_lang_
314     \tl_use:c { \__zrefclever_language_varname:n {#1} }
315     _default_ #2 _ #3
316   }
317   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
318 }
319 \cs_generate_variant:Nn \__zrefclever_opt_varname_lang_default:nnn { enn }

```

(End of definition for __zrefclever_opt_varname_lang_default:nnn.)

__zrefclever_opt_varname_lang_type:nnnn

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

```

\__zrefclever_opt_varname_lang_type:nnnn {\langle lang \rangle} {\langle ref type \rangle}
{\langle option \rangle} {\langle data type \rangle}

```

```

320 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
321 {
322   \__zrefclever_language_if_declared:nTF {#1}
323   {
324     g__zrefclever_opt_lang_
325     \tl_use:c { \__zrefclever_language_varname:n {#1} }
326     _type_ #2 _ #3 _ #4
327   }
328   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
329 }
330 \cs_generate_variant:Nn
331 \__zrefclever_opt_varname_lang_type:nnnn { eenn , eenen }

```

(End of definition for __zrefclever_opt_varname_lang_type:nnnn.)

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

```

\__zrefclever_opt_varname_fallback:nn {\<option>} {\<data type>}
332 \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
333 { c__zrefclever_opt_fallback_ #1 _ #2 }

```

(End of definition for __zrefclever_opt_varname_fallback:nn.)

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

```

\__zrefclever_opt_var_set_bool:n {\<option var>}
334 \cs_new:Npn \__zrefclever_opt_var_set_bool:n #1
335 { \cs_to_str:N #1 _is_set_bool }

```

(End of definition for __zrefclever_opt_var_set_bool:n.)

```

\__zrefclever_opt_tl_set:Nn
\__zrefclever_opt_tl_clear:N
\__zrefclever_opt_tl_gset:N
\__zrefclever_opt_tl_gclear:N
\__zrefclever_opt_tl_gclear:N
336 \cs_new_protected:Npn \__zrefclever_opt_tl_set:Nn #1#2
337 {
338   \tl_if_exist:NF #1
339   { \tl_new:N #1 }
340   \tl_set:Nn #1 {#2}
341   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
342   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
343   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }

```

```

344 }
345 \cs_generate_variant:Nn \__zrefclever_opt_tl_set:Nn { cn }
346 \cs_new_protected:Npn \__zrefclever_opt_tl_clear:N #1
347 {
348   \tl_if_exist:NF #1
349     { \tl_new:N #1 }
350   \tl_clear:N #1
351   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
352     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
353   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
354 }
355 \cs_generate_variant:Nn \__zrefclever_opt_tl_clear:N { c }
356 \cs_new_protected:Npn \__zrefclever_opt_tl_gset:Nn #1#2
357 {
358   \tl_if_exist:NF #1
359     { \tl_new:N #1 }
360   \tl_gset:Nn #1 {#2}
361 }
362 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset:Nn { cn }
363 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear:N #1
364 {
365   \tl_if_exist:NF #1
366     { \tl_new:N #1 }
367   \tl_gclear:N #1
368 }
369 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear:N { c }

```

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

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

```

\__zrefclever_opt_tl_unset:N { $\langle option\ tl \rangle$ }
370 \cs_new_protected:Npn \__zrefclever_opt_tl_unset:N #1
371 {
372   \tl_if_exist:NT #1
373   {
374     \tl_clear:N #1 % ?
375     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
376       { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
377       { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
378   }
379 }
380 \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End of definition for __zrefclever_opt_tl_unset:N.)

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

```

\__zrefclever_opt_tl_if_set:N(TF) { $\langle option\ tl \rangle$ } { $\langle true \rangle$ } { $\langle false \rangle$ }

```

```

381 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
382 {
383   \tl_if_exist:NTF #1
384   {
385     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
386     {
387       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
388       { \prg_return_true: }
389       { \prg_return_false: }
390     }
391     { \prg_return_true: }
392   }
393   { \prg_return_false: }
394 }

```

(End of definition for __zrefclever_opt_tl_if_set:NTF.)

__zrefclever_opt_tl_gset_if_new:Nn
__zrefclever_opt_tl_gclear_if_new:N

```

\__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
395 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
396 {
397   \__zrefclever_opt_tl_if_set:NF #1
398   {
399     \tl_if_exist:NF #1
400     { \tl_new:N #1 }
401     \tl_gset:Nn #1 {#2}
402   }
403 }
404 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
405 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
406 {
407   \__zrefclever_opt_tl_if_set:NF #1
408   {
409     \tl_if_exist:NF #1
410     { \tl_new:N #1 }
411     \tl_gclear:N #1
412   }
413 }
414 \cs_generate_variant:Nn \__zrefclever_opt_tl_gclear_if_new:N { c }

```

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

__zrefclever_opt_tl_get:NNTF

```

\__zrefclever_opt_tl_get:NN(TF) {<option tl to get>} {<tl var to set>}
{<true>} {<false>}
415 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
416 {
417   \__zrefclever_opt_tl_if_set:NTF #1
418   {
419     \tl_set_eq:NN #2 #1
420     \prg_return_true:
421   }
422   { \prg_return_false: }
423 }

```



```

424 \prg_generate_conditional_variant:Nnn
425 \__zrefclever_opt_tl_get:NN { cN } { F }

```

(End of definition for __zrefclever_opt_tl_get:NNTF.)

```

\__zrefclever_opt_seq_set_clist_split:Nn
\__zrefclever_opt_seq_gset_clist_split:Nn
\__zrefclever_opt_seq_set_eq:NN
\__zrefclever_opt_seq_gset_eq:NN
426 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
427 { \seq_set_split:Nnn #1 { , } {#2} }
428 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
429 { \seq_gset_split:Nnn #1 { , } {#2} }
430 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
431 {
432   \seq_if_exist:NF #1
433   { \seq_new:N #1 }
434   \seq_set_eq:NN #1 #2
435   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
436   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
437   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
438 }
439 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
440 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
441 {
442   \seq_if_exist:NF #1
443   { \seq_new:N #1 }
444   \seq_gset_eq:NN #1 #2
445 }
446 \cs_generate_variant:Nn \__zrefclever_opt_seq_gset_eq:NN { cN }

```

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

__zrefclever_opt_seq_unset:N Unset $\langle option\ seq \rangle$.

```

\__zrefclever_opt_seq_unset:N { \langle option\ seq \rangle }
447 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
448 {
449   \seq_if_exist:NT #1
450   {
451     \seq_clear:N #1 % ?
452     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
453     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
454     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
455   }
456 }
457 \cs_generate_variant:Nn \__zrefclever_opt_seq_unset:N { c }

```

(End of 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) { \langle option\ seq \rangle } { \langle true \rangle } { \langle false \rangle }

```

```

458 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
459 {
460   \seq_if_exist:NTF #1
461   {
462     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
463     {
464       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
465       { \prg_return_true: }
466       { \prg_return_false: }
467     }
468     { \prg_return_true: }
469   }
470   { \prg_return_false: }
471 }
472 \prg_generate_conditional_variant:Nnn
473 \__zrefclever_opt_seq_if_set:N { c } { F , TF }

```

(End of definition for __zrefclever_opt_seq_if_set:NTF.)

```

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

```

(End of definition for __zrefclever_opt_seq_get:NNTF.)

__zrefclever_opt_bool_unset:N Unset <option bool>.

```

\__zrefclever_opt_bool_unset:N {<option bool>}
485 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
486 {
487   \bool_if_exist:NT #1
488   {
489     % \bool_set_false:N #1 % ?
490     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
491     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
492     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
493   }
494 }
495 \cs_generate_variant:Nn \__zrefclever_opt_bool_unset:N { c }

```

(End of definition for __zrefclever_opt_bool_unset:N.)

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

```

\__zrefclever_opt_bool_if_set:N(TF) {<option bool>} {<true>} {<false>}

```

```

496 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
497 {
498   \bool_if_exist:NTF #1
499   {
500     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
501     {
502       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
503       { \prg_return_true: }
504       { \prg_return_false: }
505     }
506     { \prg_return_true: }
507   }
508   { \prg_return_false: }
509 }
510 \prg_generate_conditional_variant:Nnn
511 \__zrefclever_opt_bool_if_set:N { c } { F , TF }

```

(End of definition for __zrefclever_opt_bool_if_set:NTF.)

```

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

```

```

543     \bool_gset_false:N #1
544   }
545   \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_false:N { c }

```

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

```

\__zrefclever_opt_bool_get:NNTF      \__zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
                                     {<true>} {<false>}}
546 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
547 {
548   \__zrefclever_opt_bool_if_set:NTF #1
549   {
550     \bool_set_eq:NN #2 #1
551     \prg_return_true:
552   }
553   { \prg_return_false: }
554 }
555 \prg_generate_conditional_variant:Nnn
556   \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End of definition for __zrefclever_opt_bool_get:NNTF.)

```

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

```

(End of definition for __zrefclever_opt_bool_if:NNTF.)

4.5 Reference format

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

The fact that we have multiple scopes to set reference format options has some implications for how we handle these options, and for the resulting UI. Since there is a clear precedence rule between the different levels, setting an option at a high priority level shadows everything below it. Hence, it may be relevant to be able to “unset” these options too, so as to be able go back to the lower precedence level of the language-specific options at any given point. However, since many of these options are token lists, or clists, for which “empty” is a legitimate value, we cannot rely on emptiness to distinguish that particular intention. How to deal with it, depends on the kind of option (its data type, to be precise). For token lists and clists/sequences, we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must

be made explicit in `\keys_define:nn` by means of the `.default:o` property of the key. For the technique, by Jonathan P. Spratte, aka ‘Skillmon’, and some discussion about it, including further insights by Phelype Oleinik, see <https://tex.stackexchange.com/q/614690> and <https://github.com/latex3/latex3/pull/988>. However, Joseph Wright seems to particularly dislike this use and the general idea of a “key with no value” being somehow meaningful for `l3keys` (e.g. his comments on the previous question, and https://tex.stackexchange.com/q/632157/#comment1576404_632157), which does make it somewhat risky to rely on this. For booleans, the situation is different, since they cannot meaningfully receive an empty value and the “key with no value” is a handy and expected shorthand for `key=true`. Therefore, for reference format option booleans, we use a third value “`unset`” for this purpose. And similarly for “choice” options.

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

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

```

565 \tl_new:N \l__zrefclever_setup_type_tl
566 \tl_new:N \l__zrefclever_setup_language_tl
567 \tl_new:N \l__zrefclever_lang_decl_case_tl
568 \seq_new:N \l__zrefclever_lang_declension_seq
569 \seq_new:N \l__zrefclever_lang_gender_seq

```

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

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

```

570 \seq_new:N \g__zrefclever_rf_opts_tl_not_type_specific_seq
571 \seq_gset_from_clist:Nn
572   \g__zrefclever_rf_opts_tl_not_type_specific_seq
573   {
574     tpairsep ,
575     tlistsep ,
576     tlastsep ,
577     notesep ,
578   }
579 \seq_new:N \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
580 \seq_gset_from_clist:Nn
581   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
582   {

```

```

583     namesep ,
584     pairsep ,
585     listsep ,
586     lastsep ,
587     rangesep ,
588     namefont ,
589     reffont ,
590 }
591 \seq_new:N \g__zrefclever_rf_opts_seq_refbounds_seq
592 \seq_gset_from_clist:Nn
593   \g__zrefclever_rf_opts_seq_refbounds_seq
594   {
595     refbounds-first ,
596     refbounds-first-sg ,
597     refbounds-first-pb ,
598     refbounds-first-rb ,
599     refbounds-mid ,
600     refbounds-mid-rb ,
601     refbounds-mid-re ,
602     refbounds-last ,
603     refbounds-last-pe ,
604     refbounds-last-re ,
605   }
606 \seq_new:N \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
607 \seq_gset_from_clist:Nn
608   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
609   {
610     cap ,
611     abbrev ,
612     rangetopair ,
613   }

```

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:`.

```

614 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
615 \seq_gset_from_clist:Nn
616   \g__zrefclever_rf_opts_tl_type_names_seq
617   {
618     Name-sg ,
619     name-sg ,
620     Name-pl ,
621     name-pl ,
622     Name-sg-ab ,
623     name-sg-ab ,
624     Name-pl-ab ,
625     name-pl-ab ,
626   }

```

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

```

627 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
628 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
629   \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
630   \g__zrefclever_rf_opts_tl_type_names_seq
631 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq

```

```

632 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
633 \g__zrefclever_rf_opts_tl_not_type_specific_seq
634 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq

```

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

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

```

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

```

```

682 }
683 \clist_map_inline:nn
684 {
685   reference ,
686   typesetup ,
687 }
688 {
689   \keys_define:nn { zref-clever/ #1 }
690   {
691     +refbounds-first .default:o = \c_novalue_tl ,
692     +refbounds-mid .default:o = \c_novalue_tl ,
693     +refbounds-last .default:o = \c_novalue_tl ,
694     +refbounds-rb .default:o = \c_novalue_tl ,
695     +refbounds-re .default:o = \c_novalue_tl ,
696     +refbounds .default:o = \c_novalue_tl ,
697     refbounds .default:o = \c_novalue_tl ,
698   }
699 }
700 \clist_map_inline:nn
701 {
702   langsetup ,
703   langfile ,
704 }
705 {
706   \keys_define:nn { zref-clever/ #1 }
707   {
708     +refbounds-first .value_required:n = true ,
709     +refbounds-mid .value_required:n = true ,
710     +refbounds-last .value_required:n = true ,
711     +refbounds-rb .value_required:n = true ,
712     +refbounds-re .value_required:n = true ,
713     +refbounds .value_required:n = true ,
714     refbounds .value_required:n = true ,
715   }
716 }

```

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.

```

717 \tl_new:N \l__zrefclever_ref_language_tl
718 \tl_new:N \l__zrefclever_current_language_tl
719 \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.

```

720 \tl_new:N \l_zrefclever_ref_language_tl
721 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```


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

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

```

\__zrefclever_language_varname:n {\langle language \rangle}

```

```

722 \cs_new:Npn \__zrefclever_language_varname:n #1
723 { g__zrefclever_declared_language_ #1 _tl }

```

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

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

```

724 \cs_set_eq:NN \zrefclever_language_varname:n
725 \__zrefclever_language_varname:n

```

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

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

```

\__zrefclever_language_if_declared:n(TF) {\langle language \rangle}

```

```

726 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
727 {
728   \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
729   { \prg_return_true: }
730   { \prg_return_false: }
731 }
732 \prg_generate_conditional_variant:Nnn
733 \__zrefclever_language_if_declared:n { x } { T , F , TF }

```

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

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

```

734 \prg_set_eq_conditional:NnN \zrefclever_language_if_declared:n
735 \__zrefclever_language_if_declared:n { TF }

```

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

`\zcDeclareLanguage` Declare a new language for use with `zref-clever`. $\langle language \rangle$ 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 [\options] {\language}
736 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
737 {
738   \group_begin:
739   \tl_if_empty:nF {#2}
740   {
741     \__zrefclever_language_if_declared:nTF {#2}
742     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
743     {
744       \tl_new:c { \__zrefclever_language_varname:n {#2} }
745       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
746       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
747       \keys_set:nn { zref-clever/declarelang } {#1}
748     }
749   }
750   \group_end:
751 }
752 \@onlypreamble \zcDeclareLanguage

```

(End of definition for \zcDeclareLanguage.)

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

```

\zcDeclareLanguageAlias {\language alias} {\aliased language}
753 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
754 {
755   \tl_if_empty:nF {#1}
756   {
757     \__zrefclever_language_if_declared:nTF {#2}
758     {
759       \tl_new:c { \__zrefclever_language_varname:n {#1} }
760       \tl_gset:cx { \__zrefclever_language_varname:n {#1} }
761       { \tl_use:c { \__zrefclever_language_varname:n {#2} } }
762     }
763     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
764   }
765 }
766 \@onlypreamble \zcDeclareLanguageAlias

```

(End of definition for \zcDeclareLanguageAlias.)

```

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

```

```

778         \_zrefclever_opt_varname_language:enn
779         { \l_zrefclever_setup_language_tl } { declension } { seq }
780     }
781     {#1}
782 },
783 declension .value_required:n = true ,
784 gender .code:n =
785 {
786     \seq_new:c
787     {
788         \_zrefclever_opt_varname_language:enn
789         { \l_zrefclever_setup_language_tl } { gender } { seq }
790     }
791     \seq_gset_from_clist:cn
792     {
793         \_zrefclever_opt_varname_language:enn
794         { \l_zrefclever_setup_language_tl } { gender } { seq }
795     }
796     {#1}
797 },
798 gender .value_required:n = true ,
799 allcaps .choices:nn =
800 { true , false }
801 {
802     \bool_new:c
803     {
804         \_zrefclever_opt_varname_language:enn
805         { \l_zrefclever_setup_language_tl } { allcaps } { bool }
806     }
807     \use:c { bool_gset_ \l_keys_choice_tl :c }
808     {
809         \_zrefclever_opt_varname_language:enn
810         { \l_zrefclever_setup_language_tl } { allcaps } { bool }
811     }
812 },
813 allcaps .default:n = true ,
814 }

```

`_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.

```

815 \cs_new_protected:Npn \_zrefclever_process_language_settings:
816 {
817     \_zrefclever_language_if_declared:xTF
818     { \l_zrefclever_ref_language_tl }
819     {

```

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.

```

820     \__zrefclever_opt_seq_get:cNF
821     {
822         \__zrefclever_opt_varname_language:enn
823         { \l__zrefclever_ref_language_tl } { declension } { seq }
824     }
825     \l__zrefclever_lang_declension_seq
826     { \seq_clear:N \l__zrefclever_lang_declension_seq }
827     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
828     {
829         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
830         {
831             \msg_warning:nxxx { zref-clever }
832             { language-no-decl-ref }
833             { \l__zrefclever_ref_language_tl }
834             { \l__zrefclever_ref_decl_case_tl }
835             \tl_clear:N \l__zrefclever_ref_decl_case_tl
836         }
837     }
838     {
839         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
840         {
841             \seq_get_left:NN \l__zrefclever_lang_declension_seq
842             \l__zrefclever_ref_decl_case_tl
843         }
844         {
845             \seq_if_in:NVF \l__zrefclever_lang_declension_seq
846             \l__zrefclever_ref_decl_case_tl
847             {
848                 \msg_warning:nxxx { zref-clever }
849                 { unknown-decl-case }
850                 { \l__zrefclever_ref_decl_case_tl }
851                 { \l__zrefclever_ref_language_tl }
852                 \seq_get_left:NN \l__zrefclever_lang_declension_seq
853                 \l__zrefclever_ref_decl_case_tl
854             }
855         }
856     }

```

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.

```

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

```

```

865     {
866       \tl_if_empty:NF \l__zrefclever_ref_gender_tl
867       {
868         \msg_warning:nnxxx { zref-clever }
869         { language-no-gender }
870         { \l__zrefclever_ref_language_tl }
871         { g }
872         { \l__zrefclever_ref_gender_tl }
873         \tl_clear:N \l__zrefclever_ref_gender_tl
874       }
875     }
876     {
877       \tl_if_empty:NF \l__zrefclever_ref_gender_tl
878       {
879         \seq_if_in:NVF \l__zrefclever_lang_gender_seq
880         \l__zrefclever_ref_gender_tl
881         {
882           \msg_warning:nnxx { zref-clever }
883           { gender-not-declared }
884           { \l__zrefclever_ref_language_tl }
885           { \l__zrefclever_ref_gender_tl }
886           \tl_clear:N \l__zrefclever_ref_gender_tl
887         }
888       }
889     }

```

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

```

890     \__zrefclever_opt_bool_if:cT
891     {
892       \__zrefclever_opt_varname_language:enn
893       { \l__zrefclever_ref_language_tl } { allcaps } { bool }
894     }
895     { \keys_set:nn { zref-clever/reference } { cap = true } }
896   }
897 {

```

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

```

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

```

```

914     }
915 }

```

(End of definition for `__zrefclever_process_language_settings:`)

4.7 Language files

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

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

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

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

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

```

916 \seq_new:N \g__zrefclever_loaded_langfiles_seq

```

(End of definition for `\g__zrefclever_loaded_langfiles_seq`.)

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

```

    \__zrefclever_provide_langfile:n {\language\}

917 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
918 {
919   \group_begin:
920   \@bsphack
921   \__zrefclever_language_if_declared:nT {#1}
922   {
923     \seq_if_in:NxF
924     \g__zrefclever_loaded_langfiles_seq
925     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
926     {
927       \exp_args:Nx \file_get:nnNTF
928       {
929         zref-clever-
930         \tl_use:c { \__zrefclever_language_varname:n {#1} }
931         .lang
932       }
933       { \ExplSyntaxOn }
934       \l_tmpa_tl
935       {
936         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
937         \tl_clear:N \l__zrefclever_setup_type_tl
938         \__zrefclever_opt_seq_get:cNF
939         {
940           \__zrefclever_opt_varname_language:nnn
941           {#1} { declension } { seq }
942         }
943         \l__zrefclever_lang_declension_seq
944         { \seq_clear:N \l__zrefclever_lang_declension_seq }
945         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
946         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
947         {
948           \seq_get_left:NN \l__zrefclever_lang_declension_seq
949           \l__zrefclever_lang_decl_case_tl
950         }
951         \__zrefclever_opt_seq_get:cNF
952         {
953           \__zrefclever_opt_varname_language:nnn
954           {#1} { gender } { seq }
955         }
956         \l__zrefclever_lang_gender_seq
957         { \seq_clear:N \l__zrefclever_lang_gender_seq }
958         \keys_set:nV { zref-clever/langfile } \l_tmpa_tl
959         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
960         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
961         \msg_info:nnx { zref-clever } { langfile-loaded }
962         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
963       }
964     }

```

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.

```

965         \seq_gput_right:Nx \g__zrefclever_loaded_langfiles_seq
966         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
967     }
968 }
969 }
970 \@esphack
971 \group_end:
972 }
973 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }

```

(End of definition for __zrefclever_provide_langfile:n.)

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

```

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

```



```

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

```

```

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

```

```

1120 {
1121   \str_case:nnF {#1}
1122   {
1123     { ref }
1124     {
1125       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1126       {
1127         \__zrefclever_opt_tl_gclear_if_new:c
1128         {
1129           \__zrefclever_opt_varname_lang_default:enn
1130           { \l__zrefclever_setup_language_tl }
1131           { endrangefunc } { tl }
1132         }
1133         \__zrefclever_opt_tl_gclear_if_new:c
1134         {
1135           \__zrefclever_opt_varname_lang_default:enn
1136           { \l__zrefclever_setup_language_tl }
1137           { endrangeprop } { tl }
1138         }
1139       }
1140     }
1141     \__zrefclever_opt_tl_gclear_if_new:c
1142     {
1143       \__zrefclever_opt_varname_lang_type:enn
1144       { \l__zrefclever_setup_language_tl }
1145       { \l__zrefclever_setup_type_tl }
1146       { endrangefunc } { tl }
1147     }
1148     \__zrefclever_opt_tl_gclear_if_new:c
1149     {
1150       \__zrefclever_opt_varname_lang_type:enn
1151       { \l__zrefclever_setup_language_tl }
1152       { \l__zrefclever_setup_type_tl }
1153       { endrangeprop } { tl }
1154     }
1155   }
1156 }
1157
1158 { stripprefix }
1159 {
1160   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1161   {
1162     \__zrefclever_opt_tl_gset_if_new:cn
1163     {
1164       \__zrefclever_opt_varname_lang_default:enn
1165       { \l__zrefclever_setup_language_tl }
1166       { endrangefunc } { tl }
1167     }
1168     { __zrefclever_get_endrange_stripprefix }
1169     \__zrefclever_opt_tl_gclear_if_new:c
1170     {
1171       \__zrefclever_opt_varname_lang_default:enn
1172       { \l__zrefclever_setup_language_tl }
1173       { endrangeprop } { tl }

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

1498             {#1} { bool }
1499         }
1500     }
1501 }
1502 {
1503     \__zrefclever_opt_bool_if_set:cF
1504     {
1505         \__zrefclever_opt_varname_lang_type:eenn
1506         { \l__zrefclever_setup_language_tl }
1507         { \l__zrefclever_setup_type_tl }
1508         {#1} { bool }
1509     }
1510     {
1511         \__zrefclever_opt_bool_gset_false:c
1512         {
1513             \__zrefclever_opt_varname_lang_type:eenn
1514             { \l__zrefclever_setup_language_tl }
1515             { \l__zrefclever_setup_type_tl }
1516             {#1} { bool }
1517         }
1518     }
1519 }
1520 },
1521 #1 .default:n = true ,
1522 no #1 .meta:n = { #1 = false } ,
1523 no #1 .value_forbidden:n = true ,
1524 }
1525 }

```

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.

```

1526 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1527 {
1528     \tl_const:cn
1529     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1530 }
1531 \keyval_parse:nnn
1532 { }
1533 { \__zrefclever_opt_tl_cset_fallback:nn }
1534 {
1535     tpairsep = {,~} ,
1536     tlistsep = {,~} ,
1537     tlastsep = {,~} ,
1538     notesep = {~} ,
1539     namesep = {\nobreakspace} ,
1540     pairsep = {,~} ,
1541     listsep = {,~} ,
1542     lastsep = {,~} ,
1543     rangesep = {\textendash} ,
1544 }

```

4.8 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}
1545 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1546 {
1547   \tl_if_empty:nTF {#3}
1548     { \prop_remove:Nn #1 {#2} }
1549     { \prop_put:Nnn #1 {#2} {#3} }
1550 }
```

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

ref option

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

```

1551 \tl_new:N \l__zrefclever_ref_property_tl
1552 \keys_define:nn { zref-clever/reference }
1553 {
1554   ref .code:n =
1555   {
1556     \tl_if_empty:nTF {#1}
1557     {
1558       \msg_warning:nnn { zref-clever }
1559       { zref-property-undefined } {#1}
1560       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1561     }
1562     {
1563       \zref@ifpropundefined {#1}
1564       {
1565         \msg_warning:nnn { zref-clever }
1566         { zref-property-undefined } {#1}
1567         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1568       }
1569       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1570     }
1571   } ,
1572   ref .initial:n = default ,
1573   ref .value_required:n = true ,
1574   page .meta:n = { ref = page },
1575   page .value_forbidden:n = true ,
1576 }
```

typeset option

```
1577 \bool_new:N \l__zrefclever_typeset_ref_bool
1578 \bool_new:N \l__zrefclever_typeset_name_bool
1579 \keys_define:nn { zref-clever/reference }
1580 {
1581   typeset .choice: ,
1582   typeset / both .code:n =
1583   {
1584     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1585     \bool_set_true:N \l__zrefclever_typeset_name_bool
1586   } ,
1587   typeset / ref .code:n =
1588   {
1589     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1590     \bool_set_false:N \l__zrefclever_typeset_name_bool
1591   } ,
1592   typeset / name .code:n =
1593   {
1594     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1595     \bool_set_true:N \l__zrefclever_typeset_name_bool
1596   } ,
1597   typeset .initial:n = both ,
1598   typeset .value_required:n = true ,
1599
1600   noname .meta:n = { typeset = ref } ,
1601   noname .value_forbidden:n = true ,
1602   noref .meta:n = { typeset = name } ,
1603   noref .value_forbidden:n = true ,
1604 }
```

sort option

```
1605 \bool_new:N \l__zrefclever_typeset_sort_bool
1606 \keys_define:nn { zref-clever/reference }
1607 {
1608   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1609   sort .initial:n = true ,
1610   sort .default:n = true ,
1611   nosort .meta:n = { sort = false } ,
1612   nosort .value_forbidden:n = true ,
1613 }
```

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.

```
1614 \seq_new:N \l__zrefclever_typesort_seq
1615 \keys_define:nn { zref-clever/reference }
1616 {
1617   typesort .code:n =
1618   {
1619     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1620     \seq_reverse:N \l__zrefclever_typesort_seq
1621   }
```

```

1621     } ,
1622     typesort .initial:n =
1623     { part , chapter , section , paragraph },
1624     typesort .value_required:n = true ,
1625     notypesort .code:n =
1626     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1627     notypesort .value_forbidden:n = true ,
1628 }

```

comp option

```

1629 \bool_new:N \l__zrefclever_typeset_compress_bool
1630 \keys_define:nn { zref-clever/reference }
1631 {
1632   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1633   comp .initial:n = true ,
1634   comp .default:n = true ,
1635   nocomp .meta:n = { comp = false },
1636   nocomp .value_forbidden:n = true ,
1637 }

```

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`.

```

1638 \NewHook { zref-clever/endrange-setup }
1639 \keys_define:nn { zref-clever/reference }
1640 {
1641   endrange .code:n =
1642   {
1643     \str_case:nnF {#1}
1644     {
1645       { ref }
1646       {
1647         \__zrefclever_opt_tl_clear:c
1648         {
1649           \__zrefclever_opt_varname_general:nn
1650           { endrangefunc } { t1 }
1651         }
1652         \__zrefclever_opt_tl_clear:c
1653         {
1654           \__zrefclever_opt_varname_general:nn
1655           { endrangeprop } { t1 }
1656         }
1657       }
1658     }
1659     { stripprefix }
1660     {
1661       \__zrefclever_opt_tl_set:cn
1662       {
1663         \__zrefclever_opt_varname_general:nn
1664         { endrangefunc } { t1 }
1665       }
1666       { __zrefclever_get_endrange_stripprefix }
1667       \__zrefclever_opt_tl_clear:c
1668       {
1669         \__zrefclever_opt_varname_general:nn
1670         { endrangeprop } { t1 }
1671       }
1672     }
1673     { pagecomp }
1674     {
1675       \__zrefclever_opt_tl_set:cn
1676       {
1677         \__zrefclever_opt_varname_general:nn
1678         { endrangefunc } { t1 }
1679       }
1680       { __zrefclever_get_endrange_pagecomp }
1681       \__zrefclever_opt_tl_clear:c
1682       {
1683         \__zrefclever_opt_varname_general:nn
1684         { endrangeprop } { t1 }
1685       }
1686     }
1687   }
1688 
```

```

1689 { pagecomp2 }
1690 {
1691   \__zrefclever_opt_tl_set:cn
1692   {
1693     \__zrefclever_opt_varname_general:nn
1694     { endrangefunc } { t1 }
1695   }
1696   { __zrefclever_get_endrange_pagecomptwo }
1697   \__zrefclever_opt_tl_clear:c
1698   {
1699     \__zrefclever_opt_varname_general:nn
1700     { endrangeprop } { t1 }
1701   }
1702 }
1703
1704 { unset }
1705 {
1706   \__zrefclever_opt_tl_unset:c
1707   {
1708     \__zrefclever_opt_varname_general:nn
1709     { endrangefunc } { t1 }
1710   }
1711   \__zrefclever_opt_tl_unset:c
1712   {
1713     \__zrefclever_opt_varname_general:nn
1714     { endrangeprop } { t1 }
1715   }
1716 }
1717 }
1718 {
1719   \tl_if_empty:nTF {#1}
1720   {
1721     \msg_warning:nnn { zref-clever }
1722     { endrange-property-undefined } {#1}
1723   }
1724   {
1725     \zref@ifpropundefined {#1}
1726     {
1727       \msg_warning:nnn { zref-clever }
1728       { endrange-property-undefined } {#1}
1729     }
1730     {
1731       \__zrefclever_opt_tl_set:cn
1732       {
1733         \__zrefclever_opt_varname_general:nn
1734         { endrangefunc } { t1 }
1735       }
1736       { __zrefclever_get_endrange_property }
1737       \__zrefclever_opt_tl_set:cn
1738       {
1739         \__zrefclever_opt_varname_general:nn
1740         { endrangeprop } { t1 }
1741       }
1742       {#1}

```

```

1743     }
1744   }
1745 }
1746 },
1747   endrange .value_required:n = true ,
1748 }
1749 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1750 {
1751   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1752   {
1753     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1754     {
1755       \__zrefclever_extract_default:Nnvn #3
1756       {#2} { \l__zrefclever_ref_property_tl } { }
1757     }
1758     { \tl_set:Nn #3 { zc@missingproperty } }
1759   }
1760   {
1761     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1762     {

```

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.

```

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

```



```

1791         \exp_args:Nxx \tl_if_eq:nnT
1792         {
1793             \__zrefclever_extract_unexp:nnn
1794             {#1} { zc@counter } { }
1795         }
1796         {
1797             \__zrefclever_extract_unexp:nnn
1798             {#2} { zc@counter } { }
1799         }
1800         {
1801             \exp_args:Nxx \tl_if_eq:nnT
1802             {
1803                 \__zrefclever_extract_unexp:nnn
1804                 {#1} { zc@enclval } { }
1805             }
1806             {
1807                 \__zrefclever_extract_unexp:nnn
1808                 {#2} { zc@enclval } { }
1809             }
1810             { \bool_set_true:N \l_tmpa_bool }
1811         }
1812     }
1813 }
1814 \bool_if:NTF \l_tmpa_bool
1815 {
1816     \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1817     {#2} { l__zrefclever_endrangeprop_tl } { }
1818 }
1819 {
1820     \zref@ifrefcontainsprop
1821     {#2} { \l__zrefclever_ref_property_tl }
1822     {
1823         \__zrefclever_extract_default:Nnvn \l_tmpb_tl
1824         {#2} { l__zrefclever_ref_property_tl } { }
1825     }
1826     { \tl_set:Nn \l_tmpb_tl { zc@missingproperty } }
1827 }
1828 \exp_args:NNNV
1829 \group_end:
1830 \tl_set:Nn #3 \l_tmpb_tl
1831 }
1832 {
1833     \__zrefclever_extract_default:Nnvn #3
1834     {#2} { l__zrefclever_endrangeprop_tl } { }
1835 }
1836 }
1837 {
1838     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1839     {
1840         \__zrefclever_extract_default:Nnvn #3
1841         {#2} { l__zrefclever_ref_property_tl } { }
1842     }
1843     { \tl_set:Nn #3 { zc@missingproperty } }
1844 }

```

```

1845     }
1846   }
1847   \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>.

```

1848   \cs_new_protected:Npn \_zrefclever_get_endrange_stripprefix:nnN #1#2#3
1849   {
1850     \zref@ifrefcontainsprop {#2} { \l_zrefclever_ref_property_tl }
1851     {
1852       \group_begin:
1853       \UseHook { zref-clever/endrange-setup }
1854       \tl_set:Nx \l_tmpa_tl
1855       {
1856         \_zrefclever_extract:nnn
1857         {#1} { \l_zrefclever_ref_property_tl } { }
1858       }
1859       \tl_set:Nx \l_tmpb_tl
1860       {
1861         \_zrefclever_extract:nnn
1862         {#2} { \l_zrefclever_ref_property_tl } { }
1863       }
1864       \bool_set_false:N \l_tmpa_bool
1865       \bool_until_do:Nn \l_tmpa_bool
1866       {
1867         \exp_args:Nxx \tl_if_eq:nnTF
1868         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1869         {
1870           \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1871           \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1872           \tl_if_empty:NT \l_tmpb_tl
1873           { \bool_set_true:N \l_tmpa_bool }
1874         }
1875         { \bool_set_true:N \l_tmpa_bool }
1876       }
1877       \exp_args:NNNV
1878       \group_end:
1879       \tl_set:Nn #3 \l_tmpb_tl
1880     }
1881     { \tl_set:Nn #3 { zc@missingproperty } }
1882   }
1883   \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>).

```

1884   \prg_new_protected_conditional:Npnn
1885   \_zrefclever_is_integer_rgx:n #1 { F , TF }
1886   {
1887     \regex_match:nnTF { \A\d+\Z } {#1}
1888     { \prg_return_true: }
1889     { \prg_return_false: }
1890   }
1891   \prg_generate_conditional_variant:Nnn
1892   \_zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End of definition for _zrefclever_is_integer_rgx:n.)

```

1893 \cs_new_protected:Npn \_zrefclever_get_endrange_pagecomp:nnN #1#2#3
1894 {
1895   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1896   {
1897     \group_begin:
1898     \UseHook { zref-clever/endrange-setup }
1899     \tl_set:Nx \l_tmpa_tl
1900     {
1901       \_zrefclever_extract:nnn
1902       {#1} { \l__zrefclever_ref_property_tl } { }
1903     }
1904     \tl_set:Nx \l_tmpb_tl
1905     {
1906       \_zrefclever_extract:nnn
1907       {#2} { \l__zrefclever_ref_property_tl } { }
1908     }
1909     \bool_set_false:N \l_tmpa_bool
1910     \_zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1911     {
1912       \_zrefclever_is_integer_rgx:VF \l_tmpb_tl
1913       { \bool_set_true:N \l_tmpa_bool }
1914     }
1915     { \bool_set_true:N \l_tmpa_bool }
1916     \bool_until_do:Nn \l_tmpa_bool
1917     {
1918       \exp_args:Nxx \tl_if_eq:nnTF
1919       { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1920       {
1921         \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1922         \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1923         \tl_if_empty:NT \l_tmpb_tl
1924         { \bool_set_true:N \l_tmpa_bool }
1925       }
1926       { \bool_set_true:N \l_tmpa_bool }
1927     }
1928     \exp_args:NNNV
1929     \group_end:
1930     \tl_set:Nn #3 \l_tmpb_tl
1931   }
1932   { \tl_set:Nn #3 { zc@missingproperty } }
1933 }
1934 \cs_generate_variant:Nn \_zrefclever_get_endrange_pagecomp:nnN { VVN }
1935 \cs_new_protected:Npn \_zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1936 {
1937   \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1938   {
1939     \group_begin:
1940     \UseHook { zref-clever/endrange-setup }
1941     \tl_set:Nx \l_tmpa_tl
1942     {
1943       \_zrefclever_extract:nnn
1944       {#1} { \l__zrefclever_ref_property_tl } { }

```

```

1945     }
1946     \tl_set:Nx \l_tmpb_tl
1947     {
1948         \__zrefclever_extract:nnn
1949         {#2} { \l__zrefclever_ref_property_tl } { }
1950     }
1951     \bool_set_false:N \l_tmpa_bool
1952     \__zrefclever_is_integer_rgx:VTF \l_tmpa_tl
1953     {
1954         \__zrefclever_is_integer_rgx:VF \l_tmpb_tl
1955         { \bool_set_true:N \l_tmpa_bool }
1956     }
1957     { \bool_set_true:N \l_tmpa_bool }
1958     \bool_until_do:Nn \l_tmpa_bool
1959     {
1960         \exp_args:Nxx \tl_if_eq:nnTF
1961         { \tl_head:V \l_tmpa_tl } { \tl_head:V \l_tmpb_tl }
1962         {
1963             \bool_lazy_or:nnTF
1964             { \int_compare_p:nNn { \l_tmpb_tl } > { 99 } }
1965             { \int_compare_p:nNn { \tl_head:V \l_tmpb_tl } = { 0 } }
1966             {
1967                 \tl_set:Nx \l_tmpa_tl { \tl_tail:V \l_tmpa_tl }
1968                 \tl_set:Nx \l_tmpb_tl { \tl_tail:V \l_tmpb_tl }
1969             }
1970             { \bool_set_true:N \l_tmpa_bool }
1971         }
1972         { \bool_set_true:N \l_tmpa_bool }
1973     }
1974     \exp_args:NNNV
1975     \group_end:
1976     \tl_set:Nn #3 \l_tmpb_tl
1977 }
1978 { \tl_set:Nn #3 { zc@missingproperty } }
1979 }
1980 \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`.

```

1981 \bool_new:N \l__zrefclever_typeset_range_bool
1982 \keys_define:nn { zref-clever/reference }
1983 {
1984     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
1985     range .initial:n = false ,
1986     range .default:n = true ,
1987 }

```

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`.

```

1988 \bool_new:N \l__zrefclever_capfirst_bool
1989 \keys_define:nn { zref-clever/reference }
1990 {
1991     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
1992     capfirst .initial:n = false ,
1993     capfirst .default:n = true ,
1994 }

```

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`.

```

1995 \bool_new:N \l__zrefclever_noabbrev_first_bool
1996 \keys_define:nn { zref-clever/reference }
1997 {
1998     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
1999     noabbrevfirst .initial:n = false ,
2000     noabbrevfirst .default:n = true ,
2001 }

```

S option

```

2002 \keys_define:nn { zref-clever/reference }
2003 {
2004     S .meta:n =
2005         { capfirst = {#1} , noabbrevfirst = {#1} },
2006     S .default:n = true ,
2007 }

```

hyperref option

```

2008 \bool_new:N \l__zrefclever_hyperlink_bool
2009 \bool_new:N \l__zrefclever_hyperref_warn_bool
2010 \keys_define:nn { zref-clever/reference }
2011 {
2012     hyperref .choice: ,
2013     hyperref / auto .code:n =
2014         {
2015             \bool_set_true:N \l__zrefclever_hyperlink_bool
2016             \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2017         } ,
2018     hyperref / true .code:n =
2019         {
2020             \bool_set_true:N \l__zrefclever_hyperlink_bool
2021             \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2022         } ,
2023     hyperref / false .code:n =
2024         {
2025             \bool_set_false:N \l__zrefclever_hyperlink_bool
2026             \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2027         } ,
2028     hyperref .initial:n = auto ,
2029     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.

```

2030     nohyperref .meta:n = { hyperref = false } ,
2031     nohyperref .value_forbidden:n = true ,
2032   }
2033   \AddToHook { begindocument }
2034   {
2035     \__zrefclever_if_package_loaded:nTF { hyperref }
2036     {
2037       \bool_if:NT \l__zrefclever_hyperlink_bool
2038       { \RequirePackage { zref-hyperref } }
2039     }
2040     {
2041       \bool_if:NT \l__zrefclever_hyperref_warn_bool
2042       { \msg_warning:nn { zref-clever } { missing-hyperref } }
2043       \bool_set_false:N \l__zrefclever_hyperlink_bool
2044     }
2045     \keys_define:nn { zref-clever/reference }
2046     {
2047       hyperref .code:n =
2048       { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2049       nohyperref .code:n =
2050       { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2051     }
2052   }

```

nameinlink option

```

2053   \str_new:N \l__zrefclever_nameinlink_str
2054   \keys_define:nn { zref-clever/reference }
2055   {
2056     nameinlink .choice: ,
2057     nameinlink / true .code:n =
2058     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2059     nameinlink / false .code:n =
2060     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2061     nameinlink / single .code:n =
2062     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2063     nameinlink / tsingle .code:n =
2064     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2065     nameinlink .initial:n = tsingle ,
2066     nameinlink .default:n = true ,
2067   }

```

preposinlink option (deprecated)

```

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

```

```

2074         { preposinlink } { refbounds }
2075     } ,
2076 }

```

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`.

```

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

```

```

2106         \tl_set:Nn \l__zrefclever_ref_language_tl
2107         { \l__zrefclever_current_language_tl }
2108     }
2109
2110     { main }
2111     {
2112         \tl_set:Nn \l__zrefclever_ref_language_tl
2113         { \l__zrefclever_main_language_tl }
2114     }
2115 }
2116 {
2117     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2118     \__zrefclever_language_if_declared:nF {#1}
2119     {
2120         \msg_warning:nnn { zref-clever }
2121         { unknown-language-opt } {#1}
2122     }
2123 }
2124 \__zrefclever_provide_langfile:x
2125 { \l__zrefclever_ref_language_tl }
2126 }
2127 } ,
2128 lang .initial:n = current ,
2129 lang .value_required:n = true ,
2130 }
2131 \AddToHook { begindocument / before }
2132 {
2133     \AddToHook { begindocument }
2134     {

```

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.

```

2135     \keys_define:nn { zref-clever/reference }
2136     {
2137         lang .code:n =
2138         {
2139             \str_case:nnF {#1}
2140             {
2141                 { current }
2142                 {
2143                     \tl_set:Nn \l__zrefclever_ref_language_tl
2144                     { \l__zrefclever_current_language_tl }
2145                 }
2146
2147                 { main }
2148                 {
2149                     \tl_set:Nn \l__zrefclever_ref_language_tl
2150                     { \l__zrefclever_main_language_tl }
2151                 }
2152             }
2153         }
2154         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2155         \__zrefclever_language_if_declared:nF {#1}

```



```

2156             {
2157                 \msg_warning:nnn { zref-clever }
2158                 { unknown-language-opt } {#1}
2159             }
2160         }
2161     } ,
2162 }
2163 }
2164 }

```

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.

```

2165 \tl_new:N \l__zrefclever_ref_decl_case_tl
2166 \keys_define:nn { zref-clever/reference }
2167 {
2168     d .code:n =
2169     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2170 }
2171 \AddToHook { begindocument }
2172 {
2173     \keys_define:nn { zref-clever/reference }
2174     {

```

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

```

2175         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2176         d .value_required:n = true ,
2177     }
2178 }

```

nudge & co. options

```

2179 \bool_new:N \l__zrefclever_nudge_enabled_bool
2180 \bool_new:N \l__zrefclever_nudge_multitype_bool
2181 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2182 \bool_new:N \l__zrefclever_nudge_singular_bool
2183 \bool_new:N \l__zrefclever_nudge_gender_bool
2184 \tl_new:N \l__zrefclever_ref_gender_tl
2185 \keys_define:nn { zref-clever/reference }
2186 {
2187     nudge .choice: ,
2188     nudge / true .code:n =
2189     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2190     nudge / false .code:n =
2191     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2192     nudge / ifdraft .code:n =
2193     {

```

```

2194     \ifdraft
2195     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2196     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2197   } ,
2198   nudge / iffial .code:n =
2199   {
2200     \ifoptionfinal
2201     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2202     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2203   } ,
2204   nudge .initial:n = false ,
2205   nudge .default:n = true ,
2206   nonudge .meta:n = { nudge = false } ,
2207   nonudge .value_forbidden:n = true ,
2208   nudgeif .code:n =
2209   {
2210     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
2211     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2212     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2213     \clist_map_inline:nn {#1}
2214     {
2215       \str_case:nnF {##1}
2216       {
2217         { multitype }
2218         { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2219         { comptosing }
2220         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2221         { gender }
2222         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2223         { all }
2224         {
2225           \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2226           \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2227           \bool_set_true:N \l__zrefclever_nudge_gender_bool
2228         }
2229       }
2230     {
2231       \msg_warning:nnn { zref-clever }
2232       { nudgeif-unknown-value } {##1}
2233     }
2234   }
2235   } ,
2236   nudgeif .value_required:n = true ,
2237   nudgeif .initial:n = all ,
2238   sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2239   sg .initial:n = false ,
2240   sg .default:n = true ,
2241   g .code:n =
2242   { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2243 }
2244 \AddToHook { begindocument }
2245 {
2246   \keys_define:nn { zref-clever/reference }
2247   {

```

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

```

2248         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2249         g .value_required:n = true ,
2250     }
2251 }

```

font option

```

2252 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2253 \keys_define:nn { zref-clever/reference }
2254 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

2255 \keys_define:nn { zref-clever/reference }
2256 {
2257     titleref .code:n =
2258     {
2259         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2260         \msg_warning:nxxx { zref-clever }{ option-deprecated } { titleref }
2261         { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2262     } ,
2263 }

```

vario option

```

2264 \keys_define:nn { zref-clever/reference }
2265 {
2266     vario .code:n =
2267     {
2268         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2269         \msg_warning:nxxx { zref-clever }{ option-deprecated } { vario }
2270         { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2271     } ,
2272 }

```

note option

```

2273 \tl_new:N \l__zrefclever_zcref_note_tl
2274 \keys_define:nn { zref-clever/reference }
2275 {
2276     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
2277     note .value_required:n = true ,
2278 }

```

check option

Integration with `zref-check`.

```

2279 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2280 \bool_new:N \l__zrefclever_zcref_with_check_bool
2281 \keys_define:nn { zref-clever/reference }
2282 {
2283     check .code:n =
2284     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2285 }
2286 \AddToHook { begindocument }
2287 {

```

```

2288 \__zrefclever_if_package_loaded:nTF { zref-check }
2289 {
2290   \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2291   {
2292     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2293     \keys_define:nn { zref-clever/reference }
2294     {
2295       check .code:n =
2296       {
2297         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
2298         \keys_set:nn { zref-check / zcheck } {#1}
2299       } ,
2300       check .value_required:n = true ,
2301     }
2302   }
2303   {
2304     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2305     \keys_define:nn { zref-clever/reference }
2306     {
2307       check .code:n =
2308       {
2309         \msg_warning:nnn { zref-clever }
2310         { zref-check-too-old } { 2021-09-16-v0.2.1 }
2311       } ,
2312     }
2313   }
2314 }
2315 {
2316   \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2317   \keys_define:nn { zref-clever/reference }
2318   {
2319     check .code:n =
2320     { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2321   }
2322 }
2323 }

```

reftype option

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

```

2324 \tl_new:N \l__zrefclever_reftype_override_tl
2325 \keys_define:nn { zref-clever/label }
2326 {
2327   reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2328   reftype .default:n = {} ,
2329   reftype .initial:n = {} ,
2330 }

```

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`.

```

2331 \prop_new:N \l__zrefclever_counter_type_prop
2332 \keys_define:nn { zref-clever/label }
2333 {
2334   countertype .code:n =
2335   {
2336     \keyval_parse:nnn
2337     {
2338       \msg_warning:nnnn { zref-clever }
2339       { key-requires-value } { countertype }
2340     }
2341     {
2342       \__zrefclever_prop_put_non_empty:Nnn
2343       \l__zrefclever_counter_type_prop
2344     }
2345     {#1}
2346   } ,
2347   countertype .value_required:n = true ,
2348   countertype .initial:n =
2349   {
2350     subsection      = section ,
2351     subsubsection   = section ,
2352     subparagraph    = paragraph ,
2353     enumi           = item ,
2354     enumii          = item ,
2355     enumiii         = item ,
2356     enumiv          = item ,
2357     mpfootnote      = footnote ,
2358   } ,
2359 }
```

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 `\subsubsubsection`”.

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclval` property, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing

removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resettters_seq` with the `counterresetby` option.

```

2360 \seq_new:N \l__zrefclever_counter_resettters_seq
2361 \keys_define:nn { zref-clever/label }
2362 {
2363   counterresetters .code:n =
2364   {
2365     \clist_map_inline:nn {#1}
2366     {
2367       \seq_if_in:NnF \l__zrefclever_counter_resettters_seq {##1}
2368       {
2369         \seq_put_right:Nn
2370         \l__zrefclever_counter_resettters_seq {##1}
2371       }
2372     }
2373   } ,
2374   counterresetters .initial:n =
2375   {
2376     part ,
2377     chapter ,
2378     section ,
2379     subsection ,
2380     subsubsection ,
2381     paragraph ,
2382     subparagraph ,
2383   },
2384   counterresetters .value_required:n = true ,
2385 }

```

counterresetby option

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

```

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

```

```

2402     counterresetby .value_required:n = true ,
2403     counterresetby .initial:n =
2404     {

```

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.

```

2405         enumii = enumi ,
2406         enumiii = enumii ,
2407         enumiv = enumiii ,
2408     } ,
2409 }

```

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.

```

2410 \tl_new:N \l__zrefclever_current_counter_tl
2411 \keys_define:nn { zref-clever/label }
2412 {
2413     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2414     currentcounter .default:n = \@currentcounter ,
2415     currentcounter .initial:n = \@currentcounter ,
2416 }

```

labelhook option

```

2417 \bool_new:N \l__zrefclever_labelhook_bool
2418 \keys_define:nn { zref-clever/label }
2419 {
2420     labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2421     labelhook .initial:n = true ,
2422     labelhook .default:n = true ,
2423 }

```

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

```

2424 \AddToHookWithArguments { label }
2425 {
2426     \bool_if:NT \l__zrefclever_labelhook_bool
2427     { \zref@wrapper@babel \zref@label {#1} }
2428 }

```

nocompat option

```

2429 \bool_new:N \g__zrefclever_nocompat_bool
2430 \seq_new:N \g__zrefclever_nocompat_modules_seq
2431 \keys_define:nn { zref-clever/reference }
2432 {

```

```

2433     nocompat .code:n =
2434     {
2435         \tl_if_empty:nTF {#1}
2436         { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2437         {
2438             \clist_map_inline:nn {#1}
2439             {
2440                 \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2441                 {
2442                     \seq_gput_right:Nn
2443                     \g__zrefclever_nocompat_modules_seq {##1}
2444                 }
2445             }
2446         } ,
2447     }
2448 }
2449 \AddToHook { begindocument }
2450 {
2451     \keys_define:nn { zref-clever/reference }
2452     {
2453         nocompat .code:n =
2454         {
2455             \msg_warning:nnn { zref-clever }
2456             { option-preamble-only } { nocompat }
2457         }
2458     }
2459 }
2460 \AtEndOfPackage
2461 {
2462     \AddToHook { begindocument }
2463     {
2464         \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2465         { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2466     }
2467 }

```

`__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>}

2468 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2469 {
2470     \AddToHook { begindocument }
2471     {
2472         \bool_if:NF \g__zrefclever_nocompat_bool

```



```

2473         { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2474         \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2475     }
2476 }

```

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

Reference options

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

```

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

```

```

2519     {
2520         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2521         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2522         { postref } { refbounds }
2523     } ,
2524 }
2525 \seq_map_inline:Nn
2526   \g__zrefclever_rf_opts_seq_refbounds_seq
2527   {
2528     \keys_define:nn { zref-clever/reference }
2529     {
2530       #1 .default:o = \c_novalue_tl ,
2531       #1 .code:n =
2532       {
2533         \tl_if_novalue:nTF {##1}
2534         {
2535           \__zrefclever_opt_seq_unset:c
2536           { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2537         }
2538         {
2539           \seq_clear:N \l_tmpa_seq
2540           \__zrefclever_opt_seq_set_clist_split:Nn
2541             \l_tmpa_seq {##1}
2542           \bool_lazy_or:nnTF
2543             { \tl_if_empty_p:n {##1} }
2544             { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2545             {
2546               \__zrefclever_opt_seq_set_eq:cN
2547               { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2548               \l_tmpa_seq
2549             }
2550             {
2551               \msg_warning:nxxx { zref-clever }
2552               { refbounds-must-be-four }
2553               {#1} { \seq_count:N \l_tmpa_seq }
2554             }
2555           }
2556         } ,
2557     }
2558 }
2559 \seq_map_inline:Nn
2560   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2561   {
2562     \keys_define:nn { zref-clever/reference }
2563     {
2564       #1 .choice: ,
2565       #1 / true .code:n =
2566       {
2567         \__zrefclever_opt_bool_set_true:c
2568         { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2569       } ,
2570       #1 / false .code:n =
2571       {
2572         \__zrefclever_opt_bool_set_false:c

```

```

2573         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2574     } ,
2575     #1 / unset .code:n =
2576     {
2577         \_zrefclever_opt_bool_unset:c
2578         { \_zrefclever_opt_varname_general:nn {#1} { bool } }
2579     } ,
2580     #1 .default:n = true ,
2581     no #1 .meta:n = { #1 = false } ,
2582     no #1 .value_forbidden:n = true ,
2583 }
2584 }

```

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.

```

2585 \keys_define:nn { }
2586 {
2587     zref-clever/zcsetup .inherit:n =
2588     {
2589         zref-clever/label ,
2590         zref-clever/reference ,
2591     }
2592 }

```

`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.

```

2593 \bool_lazy_and:nnT
2594 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2595 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2596 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

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

```

2597 \NewDocumentCommand \zcsetup { m }
2598 { \_zrefclever_zcsetup:n {#1} }

```

(End of definition for `\zcsetup`.)

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

```

    \__zrefclever_zcsetup:n{<options>}

2599 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
2600 { \keys_set:nn { zref-clever/zcsetup } {#1} }
2601 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End of definition for __zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

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

```

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

2602 \NewDocumentCommand \zcRefTypeSetup { m m }
2603 {
2604   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
2605   \keys_set:nn { zref-clever/typesetup } {#2}
2606   \tl_clear:N \l__zrefclever_setup_type_tl
2607 }

(End of definition for \zcRefTypeSetup.)

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

```

```

2635         }
2636     {
2637         \__zrefclever_opt_tl_set:cn
2638         {
2639             \__zrefclever_opt_varname_type:enn
2640             { \l__zrefclever_setup_type_tl } {#1} { t1 }
2641         }
2642         {##1}
2643     }
2644 } ,
2645 }
2646 }
2647 \keys_define:nn { zref-clever/typesetup }
2648 {
2649     endrange .code:n =
2650     {
2651         \str_case:nnF {#1}
2652         {
2653             { ref }
2654             {
2655                 \__zrefclever_opt_tl_clear:c
2656                 {
2657                     \__zrefclever_opt_varname_type:enn
2658                     { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2659                 }
2660                 \__zrefclever_opt_tl_clear:c
2661                 {
2662                     \__zrefclever_opt_varname_type:enn
2663                     { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2664                 }
2665             }
2666         }
2667     { stripprefix }
2668     {
2669         \__zrefclever_opt_tl_set:cn
2670         {
2671             \__zrefclever_opt_varname_type:enn
2672             { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2673         }
2674         { __zrefclever_get_endrange_stripprefix }
2675         \__zrefclever_opt_tl_clear:c
2676         {
2677             \__zrefclever_opt_varname_type:enn
2678             { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2679         }
2680     }
2681     { pagecomp }
2682     {
2683         \__zrefclever_opt_tl_set:cn
2684         {
2685             \__zrefclever_opt_varname_type:enn
2686             { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2687         }
2688     }

```

```

2689         { __zrefclever_get_endrange_pagecomp }
2690     \__zrefclever_opt_tl_clear:c
2691     {
2692         \__zrefclever_opt_varname_type:enn
2693         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2694     }
2695 }
2696
2697 { pagecomp2 }
2698 {
2699     \__zrefclever_opt_tl_set:cn
2700     {
2701         \__zrefclever_opt_varname_type:enn
2702         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2703     }
2704     { __zrefclever_get_endrange_pagecomptwo }
2705     \__zrefclever_opt_tl_clear:c
2706     {
2707         \__zrefclever_opt_varname_type:enn
2708         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2709     }
2710 }
2711
2712 { unset }
2713 {
2714     \__zrefclever_opt_tl_unset:c
2715     {
2716         \__zrefclever_opt_varname_type:enn
2717         { \l__zrefclever_setup_type_tl } { endrangefunc } { tl }
2718     }
2719     \__zrefclever_opt_tl_unset:c
2720     {
2721         \__zrefclever_opt_varname_type:enn
2722         { \l__zrefclever_setup_type_tl } { endrangeprop } { tl }
2723     }
2724 }
2725 }
2726 {
2727     \tl_if_empty:nTF {#1}
2728     {
2729         \msg_warning:nnn { zref-clever }
2730         { endrange-property-undefined } {#1}
2731     }
2732     {
2733         \zref@ifpropundefined {#1}
2734         {
2735             \msg_warning:nnn { zref-clever }
2736             { endrange-property-undefined } {#1}
2737         }
2738         {
2739             \__zrefclever_opt_tl_set:cn
2740             {
2741                 \__zrefclever_opt_varname_type:enn
2742                 { \l__zrefclever_setup_type_tl }

```

```

2743         { endrangefunc } { t1 }
2744     }
2745     { __zrefclever_get_endrange_property }
2746     \__zrefclever_opt_t1_set:cn
2747     {
2748         \__zrefclever_opt_varname_type:enn
2749         { \l__zrefclever_setup_type_t1 }
2750         { endrangeprop } { t1 }
2751     }
2752     {#1}
2753 }
2754 }
2755 }
2756 },
2757 endrange .value_required:n = true ,
2758 }
2759 \keys_define:nn { zref-clever/typesetup }
2760 {
2761     refpre .code:n =
2762     {
2763         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2764         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2765         { refpre } { refbounds }
2766     } ,
2767     refpos .code:n =
2768     {
2769         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2770         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2771         { refpos } { refbounds }
2772     } ,
2773     preref .code:n =
2774     {
2775         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2776         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2777         { preref } { refbounds }
2778     } ,
2779     postref .code:n =
2780     {
2781         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2782         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2783         { postref } { refbounds }
2784     } ,
2785 }
2786 \seq_map_inline:Nn
2787 \g__zrefclever_rf_opts_seq_refbounds_seq
2788 {
2789     \keys_define:nn { zref-clever/typesetup }
2790     {
2791         #1 .default:o = \c_novalue_t1 ,
2792         #1 .code:n =
2793         {
2794             \tl_if_novalue:nTF {##1}
2795             {
2796                 \__zrefclever_opt_seq_unset:c

```

```

2797         {
2798             \__zrefclever_opt_varname_type:enn
2799             { \l__zrefclever_setup_type_tl } {#1} { seq }
2800         }
2801     }
2802     {
2803         \seq_clear:N \l_tmpa_seq
2804         \__zrefclever_opt_seq_set_clist_split:Nn
2805         \l_tmpa_seq {##1}
2806         \bool_lazy_or:nnTF
2807         { \tl_if_empty_p:n {##1} }
2808         { \int_compare_p:nNn { \seq_count:N \l_tmpa_seq } = { 4 } }
2809         {
2810             \__zrefclever_opt_seq_set_eq:cN
2811             {
2812                 \__zrefclever_opt_varname_type:enn
2813                 { \l__zrefclever_setup_type_tl } {#1} { seq }
2814             }
2815             \l_tmpa_seq
2816         }
2817         {
2818             \msg_warning:nxxx { zref-clever }
2819             { refbounds-must-be-four }
2820             {#1} { \seq_count:N \l_tmpa_seq }
2821         }
2822     }
2823 } ,
2824 }
2825 }
2826 \seq_map_inline:Nn
2827 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2828 {
2829     \keys_define:nn { zref-clever/typesetup }
2830     {
2831         #1 .choice: ,
2832         #1 / true .code:n =
2833         {
2834             \__zrefclever_opt_bool_set_true:c
2835             {
2836                 \__zrefclever_opt_varname_type:enn
2837                 { \l__zrefclever_setup_type_tl }
2838                 {#1} { bool }
2839             }
2840         } ,
2841         #1 / false .code:n =
2842         {
2843             \__zrefclever_opt_bool_set_false:c
2844             {
2845                 \__zrefclever_opt_varname_type:enn
2846                 { \l__zrefclever_setup_type_tl }
2847                 {#1} { bool }
2848             }
2849         } ,
2850         #1 / unset .code:n =

```



```

2851     {
2852         \__zrefclever_opt_bool_unset:c
2853         {
2854             \__zrefclever_opt_varname_type:enn
2855             { \l__zrefclever_setup_type_tl }
2856             {#1} { bool }
2857         }
2858     } ,
2859     #1 .default:n = true ,
2860     no #1 .meta:n = { #1 = false } ,
2861     no #1 .value_forbidden:n = true ,
2862 }
2863 }

```

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

```

```

2892     }
2893     { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
2894     \group_end:
2895   }
2896   \@onlypreamble \zcLanguageSetup

```

(End of definition for \zcLanguageSetup.)

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

```

2897 \keys_define:nn { zref-clever/langsetup }
2898 {
2899   type .code:n =
2900   {
2901     \tl_if_empty:NTF {#1}
2902     { \tl_clear:N \l__zrefclever_setup_type_tl }
2903     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2904   } ,
2905
2906   case .code:n =
2907   {
2908     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2909     {
2910       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
2911       { \l__zrefclever_setup_language_tl } {#1}
2912     }
2913     {
2914       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2915       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2916       {
2917         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
2918         {#1} { \l__zrefclever_setup_language_tl }
2919         \seq_get_left:NN \l__zrefclever_lang_declension_seq
2920         \l__zrefclever_lang_decl_case_tl
2921       }
2922     }
2923   } ,
2924   case .value_required:n = true ,
2925
2926   gender .value_required:n = true ,
2927   gender .code:n =
2928   {
2929     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2930     {
2931       \msg_warning:nxxxx { zref-clever } { language-no-gender }
2932       { \l__zrefclever_setup_language_tl } { gender } {#1}
2933     }
2934     {
2935       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2936       {
2937         \msg_warning:nnn { zref-clever }
2938         { option-only-type-specific } { gender }
2939       }
2940       {
2941         \seq_clear:N \l_tmpa_seq

```

```

2942         \clist_map_inline:nn {#1}
2943         {
2944             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2945             { \seq_put_right:Nn \l_tmpa_seq {##1} }
2946             {
2947                 \msg_warning:nxxx { zref-clever }
2948                 { gender-not-declared }
2949                 { \l__zrefclever_setup_language_tl } {##1}
2950             }
2951         }
2952     \__zrefclever_opt_seq_gset_eq:cN
2953     {
2954         \__zrefclever_opt_varname_lang_type:eenn
2955         { \l__zrefclever_setup_language_tl }
2956         { \l__zrefclever_setup_type_tl }
2957         { gender }
2958         { seq }
2959     }
2960     \l_tmpa_seq
2961 }
2962 }
2963 },
2964 }
2965 \seq_map_inline:Nn
2966 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2967 {
2968     \keys_define:nn { zref-clever/langsetup }
2969     {
2970         #1 .value_required:n = true ,
2971         #1 .code:n =
2972         {
2973             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2974             {
2975                 \__zrefclever_opt_tl_gset:cn
2976                 {
2977                     \__zrefclever_opt_varname_lang_default:enn
2978                     { \l__zrefclever_setup_language_tl } {#1} { tl }
2979                 }
2980                 {##1}
2981             }
2982             {
2983                 \msg_warning:nnn { zref-clever }
2984                 { option-not-type-specific } {#1}
2985             }
2986         },
2987     }
2988 }
2989 \seq_map_inline:Nn
2990 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
2991 {
2992     \keys_define:nn { zref-clever/langsetup }
2993     {
2994         #1 .value_required:n = true ,
2995         #1 .code:n =

```

```

2996 {
2997   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2998   {
2999     \__zrefclever_opt_tl_gset:cn
3000     {
3001       \__zrefclever_opt_varname_lang_default:enn
3002       { \l__zrefclever_setup_language_tl } {#1} { t1 }
3003     }
3004     {##1}
3005   }
3006   {
3007     \__zrefclever_opt_tl_gset:cn
3008     {
3009       \__zrefclever_opt_varname_lang_type:eenn
3010       { \l__zrefclever_setup_language_tl }
3011       { \l__zrefclever_setup_type_tl }
3012       {#1} { t1 }
3013     }
3014     {##1}
3015   }
3016   } ,
3017 }
3018 }
3019 \keys_define:nn { zref-clever/langsetup }
3020 {
3021   endrange .value_required:n = true ,
3022   endrange .code:n =
3023   {
3024     \str_case:nnF {#1}
3025     {
3026       { ref }
3027       {
3028         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3029         {
3030           \__zrefclever_opt_tl_gclear:c
3031           {
3032             \__zrefclever_opt_varname_lang_default:enn
3033             { \l__zrefclever_setup_language_tl }
3034             { endrangefunc } { t1 }
3035           }
3036           \__zrefclever_opt_tl_gclear:c
3037           {
3038             \__zrefclever_opt_varname_lang_default:enn
3039             { \l__zrefclever_setup_language_tl }
3040             { endrangeprop } { t1 }
3041           }
3042         }
3043         {
3044           \__zrefclever_opt_tl_gclear:c
3045           {
3046             \__zrefclever_opt_varname_lang_type:eenn
3047             { \l__zrefclever_setup_language_tl }
3048             { \l__zrefclever_setup_type_tl }
3049             { endrangefunc } { t1 }

```

```

3050     }
3051     \__zrefclever_opt_tl_gclear:c
3052     {
3053         \__zrefclever_opt_varname_lang_type:enn
3054         { \l__zrefclever_setup_language_tl }
3055         { \l__zrefclever_setup_type_tl }
3056         { endrangeprop } { tl }
3057     }
3058 }
3059 }
3060
3061 { stripprefix }
3062 {
3063     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3064     {
3065         \__zrefclever_opt_tl_gset:cn
3066         {
3067             \__zrefclever_opt_varname_lang_default:enn
3068             { \l__zrefclever_setup_language_tl }
3069             { endrangefunc } { tl }
3070         }
3071         { __zrefclever_get_endrange_stripprefix }
3072         \__zrefclever_opt_tl_gclear:c
3073         {
3074             \__zrefclever_opt_varname_lang_default:enn
3075             { \l__zrefclever_setup_language_tl }
3076             { endrangeprop } { tl }
3077         }
3078     }
3079     {
3080         \__zrefclever_opt_tl_gset:cn
3081         {
3082             \__zrefclever_opt_varname_lang_type:enn
3083             { \l__zrefclever_setup_language_tl }
3084             { \l__zrefclever_setup_type_tl }
3085             { endrangefunc } { tl }
3086         }
3087         { __zrefclever_get_endrange_stripprefix }
3088         \__zrefclever_opt_tl_gclear:c
3089         {
3090             \__zrefclever_opt_varname_lang_type:enn
3091             { \l__zrefclever_setup_language_tl }
3092             { \l__zrefclever_setup_type_tl }
3093             { endrangeprop } { tl }
3094         }
3095     }
3096 }
3097
3098 { pagecomp }
3099 {
3100     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3101     {
3102         \__zrefclever_opt_tl_gset:cn
3103         {

```

```

3104         \_zrefclever_opt_varname_lang_default:enn
3105         { \l_zrefclever_setup_language_tl }
3106         { endrangefunc } { tl }
3107     }
3108     { __zrefclever_get_endrange_pagecomp }
3109 \_zrefclever_opt_tl_gclear:c
3110 {
3111     \_zrefclever_opt_varname_lang_default:enn
3112     { \l_zrefclever_setup_language_tl }
3113     { endrangeprop } { tl }
3114 }
3115 }
3116 {
3117     \_zrefclever_opt_tl_gset:cn
3118     {
3119         \_zrefclever_opt_varname_lang_type:eenn
3120         { \l_zrefclever_setup_language_tl }
3121         { \l_zrefclever_setup_type_tl }
3122         { endrangefunc } { tl }
3123     }
3124     { __zrefclever_get_endrange_pagecomp }
3125 \_zrefclever_opt_tl_gclear:c
3126 {
3127     \_zrefclever_opt_varname_lang_type:eenn
3128     { \l_zrefclever_setup_language_tl }
3129     { \l_zrefclever_setup_type_tl }
3130     { endrangeprop } { tl }
3131 }
3132 }
3133 }
3134
3135 { pagecomp2 }
3136 {
3137     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
3138     {
3139         \_zrefclever_opt_tl_gset:cn
3140         {
3141             \_zrefclever_opt_varname_lang_default:enn
3142             { \l_zrefclever_setup_language_tl }
3143             { endrangefunc } { tl }
3144         }
3145         { __zrefclever_get_endrange_pagecomptwo }
3146 \_zrefclever_opt_tl_gclear:c
3147 {
3148     \_zrefclever_opt_varname_lang_default:enn
3149     { \l_zrefclever_setup_language_tl }
3150     { endrangeprop } { tl }
3151 }
3152 }
3153 {
3154     \_zrefclever_opt_tl_gset:cn
3155     {
3156         \_zrefclever_opt_varname_lang_type:eenn
3157         { \l_zrefclever_setup_language_tl }

```

```

3158         { \l__zrefclever_setup_type_tl }
3159         { endrangefunc } { tl }
3160     }
3161     { __zrefclever_get_endrange_pagecomptwo }
3162 \__zrefclever_opt_tl_gclear:c
3163 {
3164     \__zrefclever_opt_varname_lang_type:enn
3165     { \l__zrefclever_setup_language_tl }
3166     { \l__zrefclever_setup_type_tl }
3167     { endrangeprop } { tl }
3168 }
3169 }
3170 }
3171 }
3172 {
3173     \tl_if_empty:nTF {#1}
3174     {
3175         \msg_warning:nnn { zref-clever }
3176         { endrange-property-undefined } {#1}
3177     }
3178     {
3179         \zref@ifpropundefined {#1}
3180         {
3181             \msg_warning:nnn { zref-clever }
3182             { endrange-property-undefined } {#1}
3183         }
3184         {
3185             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3186             {
3187                 \__zrefclever_opt_tl_gset:cn
3188                 {
3189                     \__zrefclever_opt_varname_lang_default:enn
3190                     { \l__zrefclever_setup_language_tl }
3191                     { endrangefunc } { tl }
3192                 }
3193                 { __zrefclever_get_endrange_property }
3194                 \__zrefclever_opt_tl_gset:cn
3195                 {
3196                     \__zrefclever_opt_varname_lang_default:enn
3197                     { \l__zrefclever_setup_language_tl }
3198                     { endrangeprop } { tl }
3199                 }
3200                 {#1}
3201             }
3202         }
3203         {
3204             \__zrefclever_opt_tl_gset:cn
3205             {
3206                 \__zrefclever_opt_varname_lang_type:enn
3207                 { \l__zrefclever_setup_language_tl }
3208                 { \l__zrefclever_setup_type_tl }
3209                 { endrangefunc } { tl }
3210             }
3211             { __zrefclever_get_endrange_property }
3212             \__zrefclever_opt_tl_gset:cn

```

```

3212         {
3213             \__zrefclever_opt_varname_lang_type:eenm
3214             { \l__zrefclever_setup_language_tl }
3215             { \l__zrefclever_setup_type_tl }
3216             { endrangeprop } { tl }
3217         }
3218         {#1}
3219     }
3220 }
3221 }
3222 }
3223 } ,
3224 }
3225 \keys_define:nn { zref-clever/langsetup }
3226 {
3227     refpre .code:n =
3228     {
3229         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3230         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3231         { refpre } { refbounds }
3232     } ,
3233     refpos .code:n =
3234     {
3235         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3236         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3237         { refpos } { refbounds }
3238     } ,
3239     preref .code:n =
3240     {
3241         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3242         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3243         { preref } { refbounds }
3244     } ,
3245     postref .code:n =
3246     {
3247         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3248         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3249         { postref } { refbounds }
3250     } ,
3251 }
3252 \seq_map_inline:Nn
3253 \g__zrefclever_rf_opts_tl_type_names_seq
3254 {
3255     \keys_define:nn { zref-clever/langsetup }
3256     {
3257         #1 .value_required:n = true ,
3258         #1 .code:n =
3259         {
3260             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3261             {
3262                 \msg_warning:nnn { zref-clever }
3263                 { option-only-type-specific } {#1}
3264             }
3265             {

```



```

3266         \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3267         {
3268             \__zrefclever_opt_tl_gset:cn
3269             {
3270                 \__zrefclever_opt_varname_lang_type:een
3271                 { \l__zrefclever_setup_language_tl }
3272                 { \l__zrefclever_setup_type_tl }
3273                 {#1} { tl }
3274             }
3275             {##1}
3276         }
3277         {
3278             \__zrefclever_opt_tl_gset:cn
3279             {
3280                 \__zrefclever_opt_varname_lang_type:een
3281                 { \l__zrefclever_setup_language_tl }
3282                 { \l__zrefclever_setup_type_tl }
3283                 { \l__zrefclever_lang_decl_case_tl - #1 }
3284                 { tl }
3285             }
3286             {##1}
3287         }
3288     } ,
3289 }
3290 }
3291 }
3292 \seq_map_inline:Nn
3293   \g__zrefclever_rf_opts_seq_refbounds_seq
3294   {
3295     \keys_define:nn { zref-clever/langsetup }
3296     {
3297       #1 .value_required:n = true ,
3298       #1 .code:n =
3299       {
3300         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3301         {
3302           \seq_gclear:N \g_tmpa_seq
3303           \__zrefclever_opt_seq_gset_clist_split:Nn
3304             \g_tmpa_seq {##1}
3305           \bool_lazy_or:nnTF
3306             { \tl_if_empty_p:n {##1} }
3307             {
3308               \int_compare_p:nNn
3309                 { \seq_count:N \g_tmpa_seq } = { 4 }
3310             }
3311             {
3312               \__zrefclever_opt_seq_gset_eq:cn
3313               {
3314                 \__zrefclever_opt_varname_lang_default:enn
3315                 { \l__zrefclever_setup_language_tl }
3316                 {#1} { seq }
3317               }
3318               \g_tmpa_seq
3319             }

```

```

3320         {
3321             \msg_warning:nxxx { zref-clever }
3322             { rebounds-must-be-four }
3323             {#1} { \seq_count:N \g_tmpa_seq }
3324         }
3325     }
3326     {
3327         \seq_gclear:N \g_tmpa_seq
3328         \__zrefclever_opt_seq_gset_clist_split:Nn
3329         \g_tmpa_seq {##1}
3330         \bool_lazy_or:nnTF
3331         { \tl_if_empty_p:n {##1} }
3332         {
3333             \int_compare_p:nNn
3334             { \seq_count:N \g_tmpa_seq } = { 4 }
3335         }
3336         {
3337             \__zrefclever_opt_seq_gset_eq:cN
3338             {
3339                 \__zrefclever_opt_varname_lang_type:eenn
3340                 { \l__zrefclever_setup_language_tl }
3341                 { \l__zrefclever_setup_type_tl } {#1} { seq }
3342             }
3343             \g_tmpa_seq
3344         }
3345         {
3346             \msg_warning:nxxx { zref-clever }
3347             { rebounds-must-be-four }
3348             {#1} { \seq_count:N \g_tmpa_seq }
3349         }
3350     } ,
3351 } ,
3352 }
3353 }
3354 \seq_map_inline:Nn
3355 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3356 {
3357     \keys_define:nn { zref-clever/langsetup }
3358     {
3359         #1 .choice: ,
3360         #1 / true .code:n =
3361         {
3362             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3363             {
3364                 \__zrefclever_opt_bool_gset_true:c
3365                 {
3366                     \__zrefclever_opt_varname_lang_default:enn
3367                     { \l__zrefclever_setup_language_tl }
3368                     {#1} { bool }
3369                 }
3370             }
3371             {
3372                 \__zrefclever_opt_bool_gset_true:c
3373                 {

```

```

3374         \_zrefclever_opt_varname_lang_type:enn
3375         { \l__zrefclever_setup_language_tl }
3376         { \l__zrefclever_setup_type_tl }
3377         {#1} { bool }
3378     }
3379 }
3380 },
3381 #1 / false .code:n =
3382 {
3383     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3384     {
3385         \_zrefclever_opt_bool_gset_false:c
3386         {
3387             \_zrefclever_opt_varname_lang_default:enn
3388             { \l__zrefclever_setup_language_tl }
3389             {#1} { bool }
3390         }
3391     }
3392     {
3393         \_zrefclever_opt_bool_gset_false:c
3394         {
3395             \_zrefclever_opt_varname_lang_type:enn
3396             { \l__zrefclever_setup_language_tl }
3397             { \l__zrefclever_setup_type_tl }
3398             {#1} { bool }
3399         }
3400     }
3401 },
3402 #1 .default:n = true ,
3403 no #1 .meta:n = { #1 = false } ,
3404 no #1 .value_forbidden:n = true ,
3405 }
3406 }

```

6 User interface

6.1 \zcref

\zcref The main user command of the package.

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

```

3407 \NewDocumentCommand \zcref { s O { } m }
3408 { \zref@wrapper@babel \_zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End of definition for \zcref.)

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

_zrefclever_zcref:nnnn {\labels} {*} {\options}

```

3409 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3410 {
3411   \group_begin:

```

Set options.

```

3412   \keys_set:nn { zref-clever/reference } {#3}

```

Store arguments values.

```

3413   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3414   \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.

```

3415   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }

```

Process language settings.

```

3416   \__zrefclever_process_language_settings:

```

Integration with `zref-check`.

```

3417   \bool_lazy_and:nnT
3418     { \l__zrefclever_zrefcheck_available_bool }
3419     { \l__zrefclever_zcref_with_check_bool }
3420     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

3421   \bool_lazy_or:nnT
3422     { \l__zrefclever_typeset_sort_bool }
3423     { \l__zrefclever_typeset_range_bool }
3424     { \__zrefclever_sort_labels: }

```

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

```

3425   \group_begin:
3426   \l__zrefclever_ref_typeset_font_tl
3427   \__zrefclever_typeset_refs:
3428   \group_end:

```

Typeset note.

```

3429   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3430   {
3431     \__zrefclever_get_rf_opt_tl:nxxN { notesep }
3432     { \l__zrefclever_label_type_a_tl }
3433     { \l__zrefclever_ref_language_tl }
3434     \l_tmpa_tl
3435     \l_tmpa_tl
3436     \l__zrefclever_zcref_note_tl
3437   }

```

Integration with `zref-check`.

```

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

```

```

3444         { \l__zrefclever_zcref_labels_seq }
3445     }

```

Integration with mathtools.

```

3446     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
3447     {
3448         \__zrefclever_mathtools_showonlyrefs:n
3449         { \l__zrefclever_zcref_labels_seq }
3450     }
3451     \group_end:
3452 }

```

(End of definition for __zrefclever_zcref:nmmn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

3453 \seq_new:N \l__zrefclever_zcref_labels_seq
3454 \bool_new:N \l__zrefclever_link_star_bool

```

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

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```

\zcpageref*[\options]{\labels}

3455 \NewDocumentCommand \zcpageref { s O { } m }
3456 {
3457     \group_begin:
3458     \IfBooleanT {#1}
3459     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3460     \zcref [#2, ref = page] {#3}
3461     \group_end:
3462 }

```

(End of definition for \zcpageref.)

7 Sorting

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

$\backslash l_zrefclever_label_type_a_tl$ $\backslash l_zrefclever_label_type_b_tl$ $\backslash l_zrefclever_label_enclval_a_tl$ $\backslash l_zrefclever_label_enclval_b_tl$ $\backslash l_zrefclever_label_extdoc_a_tl$ $\backslash l_zrefclever_label_extdoc_b_tl$	<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> 3463 \tl_new:N \l__zrefclever_label_type_a_tl 3464 \tl_new:N \l__zrefclever_label_type_b_tl 3465 \tl_new:N \l__zrefclever_label_enclval_a_tl 3466 \tl_new:N \l__zrefclever_label_enclval_b_tl 3467 \tl_new:N \l__zrefclever_label_extdoc_a_tl 3468 \tl_new:N \l__zrefclever_label_extdoc_b_tl </pre> <p><i>(End of 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> 3469 \bool_new:N \l__zrefclever_sort_decided_bool </pre> <p><i>(End of 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> 3470 \int_new:N \l__zrefclever_sort_prior_a_int 3471 \int_new:N \l__zrefclever_sort_prior_b_int </pre> <p><i>(End of 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> 3472 \seq_new:N \l__zrefclever_label_types_seq </pre> <p><i>(End of 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> 3473 \cs_new_protected:Npn _zrefclever_sort_labels: 3474 { </pre> <p>Store label types sequence.</p> <pre> 3475 \seq_clear:N \l__zrefclever_label_types_seq 3476 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page } 3477 { 3478 \seq_map_function:NN \l__zrefclever_zcref_labels_seq 3479 _zrefclever_label_type_put_new_right:n 3480 } </pre>

Sort.

```

3481 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3482 {
3483   \zref@ifrefundefined {##1}
3484   {
3485     \zref@ifrefundefined {##2}
3486     {
3487       % Neither label is defined.
3488       \sort_return_same:
3489     }
3490     {
3491       % The second label is defined, but the first isn't, leave the
3492       % undefined first (to be more visible).
3493       \sort_return_same:
3494     }
3495   }
3496   {
3497     \zref@ifrefundefined {##2}
3498     {
3499       % The first label is defined, but the second isn't, bring the
3500       % second forward.
3501       \sort_return_swapped:
3502     }
3503     {
3504       % The interesting case: both labels are defined. References
3505       % to the "default" property or to the "page" are quite
3506       % different with regard to sorting, so we branch them here to
3507       % specialized functions.
3508       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3509       { \__zrefclever_sort_page:nn {##1} {##2} }
3510       { \__zrefclever_sort_default:nn {##1} {##2} }
3511     }
3512   }
3513 }
3514 }
```

(End of definition for __zrefclever_sort_labels:.)

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

3515 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3516 {
3517   \__zrefclever_extract_default:Nnnn
3518   \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3519   \seq_if_in:NVF \l__zrefclever_label_types_seq
```

```

3520 \l__zrefclever_label_type_a_tl
3521 {
3522   \seq_put_right:NV \l__zrefclever_label_types_seq
3523   \l__zrefclever_label_type_a_tl
3524 }
3525 }

```

(End of definition for __zrefclever_label_type_put_new_right:n.)

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

```

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

3526 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3527 {
3528   \__zrefclever_extract_default:Nnnn
3529   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
3530   \__zrefclever_extract_default:Nnnn
3531   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
3532
3533   \tl_if_eq:NNTF
3534   \l__zrefclever_label_type_a_tl
3535   \l__zrefclever_label_type_b_tl
3536   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3537   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3538 }

```

(End of definition for __zrefclever_sort_default:nn.)

```

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

3539 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3540 {
3541   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3542   {#1} {zc@enclval} { }
3543   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3544   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3545   {#2} {zc@enclval} { }
3546   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3547   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3548   {#1} {externaldocument} { }
3549   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3550   {#2} {externaldocument} { }
3551
3552   \bool_set_false:N \l__zrefclever_sort_decided_bool
3553
3554   % First we check if there's any "external document" difference (coming
3555   % from 'zref-xr') and, if so, sort based on that.
3556   \tl_if_eq:NNTF
3557   \l__zrefclever_label_extdoc_a_tl
3558   \l__zrefclever_label_extdoc_b_tl
3559   {

```



```

3560 \bool_if:nTF
3561 {
3562   \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3563   ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3564 }
3565 {
3566   \bool_set_true:N \l__zrefclever_sort_decided_bool
3567   \sort_return_same:
3568 }
3569 {
3570   \bool_if:nTF
3571   {
3572     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3573     \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3574   }
3575   {
3576     \bool_set_true:N \l__zrefclever_sort_decided_bool
3577     \sort_return_swapped:
3578   }
3579   {
3580     \bool_set_true:N \l__zrefclever_sort_decided_bool
3581     % Two different "external documents": last resort, sort by the
3582     % document name itself.
3583     \str_compare:eNeTF
3584     { \l__zrefclever_label_extdoc_b_tl } <
3585     { \l__zrefclever_label_extdoc_a_tl }
3586     { \sort_return_swapped: }
3587     { \sort_return_same: }
3588   }
3589 }
3590 }
3591
3592 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3593 {
3594   \bool_if:nTF
3595   {
3596     % Both are empty: neither label has any (further) "enclosing
3597     % counters" (left).
3598     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3599     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3600   }
3601   {
3602     \bool_set_true:N \l__zrefclever_sort_decided_bool
3603     \int_compare:nNnTF
3604     { \__zrefclever_extract:nnn {#1} {zc@cntval} { -1 } }
3605     >
3606     { \__zrefclever_extract:nnn {#2} {zc@cntval} { -1 } }
3607     { \sort_return_swapped: }
3608     { \sort_return_same: }
3609   }
3610   {
3611     \bool_if:nTF
3612     {
3613       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.

```

```

3614     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3615   }
3616   {
3617     \bool_set_true:N \l__zrefclever_sort_decided_bool
3618     \int_compare:nNnTF
3619       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3620       >
3621       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3622       { \sort_return_swapped: }
3623       { \sort_return_same: }
3624   }
3625   {
3626     \bool_if:nTF
3627     {
3628       % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
3629       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3630     }
3631     {
3632       \bool_set_true:N \l__zrefclever_sort_decided_bool
3633       \int_compare:nNnTF
3634         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3635         <
3636         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3637         { \sort_return_same: }
3638         { \sort_return_swapped: }
3639     }
3640     {
3641       % Neither is empty: we can compare the values of the
3642       % current enclosing counter in the loop, if they are
3643       % equal, we are still in the loop, if they are not, a
3644       % sorting decision can be made directly.
3645       \int_compare:nNnTF
3646         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3647         =
3648         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3649         {
3650           \tl_set:Nx \l__zrefclever_label_enclval_a_tl
3651             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3652           \tl_set:Nx \l__zrefclever_label_enclval_b_tl
3653             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3654         }
3655         {
3656           \bool_set_true:N \l__zrefclever_sort_decided_bool
3657           \int_compare:nNnTF
3658             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3659             >
3660             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3661             { \sort_return_swapped: }
3662             { \sort_return_same: }
3663         }
3664     }
3665   }
3666 }
3667

```

```
3668 }
```

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

```
_zrefclever_sort_default_different_types:nn
```

```
\_zrefclever_sort_default_different_types:nn {\label a} {\label b}
```

```
3669 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
```

```
3670 {
```

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”.

```
3671 \int_zero:N \l_zrefclever_sort_prior_a_int
```

```
3672 \int_zero:N \l_zrefclever_sort_prior_b_int
```

```
3673 \seq_map_indexed_inline:Nn \l_zrefclever_typesort_seq
```

```
3674 {
```

```
\tl_if_eq:nnTF {##2} {{othertypes}}
```

```
{
```

```
\int_compare:nNnT { \l_zrefclever_sort_prior_a_int } = { 0 }
```

```
{ \int_set:Nn \l_zrefclever_sort_prior_a_int { - ##1 } }
```

```
\int_compare:nNnT { \l_zrefclever_sort_prior_b_int } = { 0 }
```

```
{ \int_set:Nn \l_zrefclever_sort_prior_b_int { - ##1 } }
```

```
}
```

```
{
```

```
\tl_if_eq:NnTF \l_zrefclever_label_type_a_tl {##2}
```

```
{ \int_set:Nn \l_zrefclever_sort_prior_a_int { - ##1 } }
```

```
{
```

```
\tl_if_eq:NnT \l_zrefclever_label_type_b_tl {##2}
```

```
{ \int_set:Nn \l_zrefclever_sort_prior_b_int { - ##1 } }
```

```
}
```

```
}
```

```
3690 }
```

Then do the actual sorting.

```
3691 \bool_if:nTF
```

```
{
```

```
\int_compare_p:nNn
```

```
{ \l_zrefclever_sort_prior_a_int } <
```

```
{ \l_zrefclever_sort_prior_b_int }
```

```
}
```

```
{ \sort_return_same: }
```

```
{
```

```
\bool_if:nTF
```

```
{
```

```
\int_compare_p:nNn
```

```
{ \l_zrefclever_sort_prior_a_int } >
```

```
{ \l_zrefclever_sort_prior_b_int }
```

```
}
```

```
{ \sort_return_swapped: }
```

```
{
```

```
% Sort priorities are equal: the type that occurs first in
```

```
% ‘labels’, as given by the user, is kept (or brought) forward.
```

```
\seq_map_inline:Nn \l_zrefclever_label_types_seq
```

```
{
```

```
\tl_if_eq:NnTF \l_zrefclever_label_type_a_tl {##1}
```

```

3712         { \seq_map_break:n { \sort_return_same: } }
3713     {
3714         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3715         { \seq_map_break:n { \sort_return_swapped: } }
3716     }
3717 }
3718 }
3719 }
3720 }

```

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

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

```

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

3721 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3722 {
3723     \int_compare:nNnTF
3724     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3725     >
3726     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3727     { \sort_return_swapped: }
3728     { \sort_return_same: }
3729 }

```

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

8 Typesetting

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

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

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

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

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

Range compression, in particular, relies heavily on counting to be able to distinguish relevant cases. `\l__zrefclever_range_count_int` counts the number of elements in the current sequential “streak”, and `\l__zrefclever_range_same_count_int` counts the number of *equal* elements in that same “streak”. The difference between the two allows us to distinguish the cases in which a range actually “skips” a number in the sequence, in which case we should use a range separator, from when they are after all just contiguous, in which case a pair separator is called for. Since, as usual, we can only know this when 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.

3730 `\seq_new:N \l__zrefclever_typeset_labels_seq`
3731 `\bool_new:N \l__zrefclever_typeset_last_bool`
3732 `\bool_new:N \l__zrefclever_last_of_type_bool`

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

`\l_zrefclever_type_count_int`
`\l_zrefclever_label_count_int`
`\l__zrefclever_ref_count_int`

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

3733 `\int_new:N \l_zrefclever_type_count_int`
3734 `\int_new:N \l_zrefclever_label_count_int`
3735 `\int_new:N \l__zrefclever_ref_count_int`

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

`\l__zrefclever_label_a_tl`
`\l__zrefclever_label_b_tl`
`\l_zrefclever_typeset_queue_prev_tl`
`\l_zrefclever_typeset_queue_curr_tl`
`\l_zrefclever_type_first_label_tl`
`\l_zrefclever_type_first_label_type_tl`

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

3736 `\tl_new:N \l__zrefclever_label_a_tl`
3737 `\tl_new:N \l__zrefclever_label_b_tl`
3738 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
3739 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
3740 `\tl_new:N \l_zrefclever_type_first_label_tl`
3741 `\tl_new:N \l_zrefclever_type_first_label_type_tl`

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

`\l__zrefclever_type_name_tl`
`\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.

3742 `\tl_new:N \l__zrefclever_type_name_tl`
3743 `\bool_new:N \l_zrefclever_name_in_link_bool`
3744 `\bool_new:N \l__zrefclever_type_name_missing_bool`
3745 `\tl_new:N \l_zrefclever_name_format_tl`
3746 `\tl_new:N \l_zrefclever_name_format_fallback_tl`
3747 `\seq_new:N \l__zrefclever_type_name_gender_seq`

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

`\l_zrefclever_range_count_int`
`\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.

3748 `\int_new:N \l_zrefclever_range_count_int`
3749 `\int_new:N \l_zrefclever_range_same_count_int`
3750 `\tl_new:N \l__zrefclever_range_beg_label_tl`
3751 `\bool_new:N \l__zrefclever_range_beg_is_first_bool`
3752 `\tl_new:N \l_zrefclever_range_end_ref_tl`
3753 `\bool_new:N \l_zrefclever_next_maybe_range_bool`
3754 `\bool_new:N \l_zrefclever_next_is_same_bool`

(End of 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.

```

3755 \tl_new:N \l__zrefclever_tpairsep_tl
3756 \tl_new:N \l__zrefclever_tlistsep_tl
3757 \tl_new:N \l__zrefclever_tlastsep_tl
3758 \tl_new:N \l__zrefclever_namesep_tl
3759 \tl_new:N \l__zrefclever_pairsep_tl
3760 \tl_new:N \l__zrefclever_listsep_tl
3761 \tl_new:N \l__zrefclever_lastsep_tl
3762 \tl_new:N \l__zrefclever_rangeseq_tl
3763 \tl_new:N \l__zrefclever_namefont_tl
3764 \tl_new:N \l__zrefclever_reffont_tl
3765 \tl_new:N \l__zrefclever_endrangefunc_tl
3766 \tl_new:N \l__zrefclever_endrangeprop_tl
3767 \bool_new:N \l__zrefclever_cap_bool
3768 \bool_new:N \l__zrefclever_abbrev_bool
3769 \bool_new:N \l__zrefclever_rangetopair_bool

```

(End of 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.

```

3770 \seq_new:N \l__zrefclever_refbounds_first_seq
3771 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
3772 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
3773 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
3774 \seq_new:N \l__zrefclever_refbounds_mid_seq
3775 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
3776 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
3777 \seq_new:N \l__zrefclever_refbounds_last_seq
3778 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
3779 \seq_new:N \l__zrefclever_refbounds_last_re_seq
3780 \seq_new:N \l__zrefclever_type_first_refbounds_seq
3781 \bool_new:N \l__zrefclever_type_first_refbounds_set_bool

```

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

```

\l__zrefclever_verbose_testing_bool

```

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

```

3782 \bool_new:N \l__zrefclever_verbose_testing_bool

```

(End of definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```

3783 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3784 {
3785   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3786   \l__zrefclever_zceref_labels_seq
3787   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3788   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3789   \tl_clear:N \l__zrefclever_type_first_label_tl

```

```

3790 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3791 \tl_clear:N \l__zrefclever_range_beg_label_tl
3792 \tl_clear:N \l__zrefclever_range_end_ref_tl
3793 \int_zero:N \l__zrefclever_label_count_int
3794 \int_zero:N \l__zrefclever_type_count_int
3795 \int_zero:N \l__zrefclever_ref_count_int
3796 \int_zero:N \l__zrefclever_range_count_int
3797 \int_zero:N \l__zrefclever_range_same_count_int
3798 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3799 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3800
3801 % Get type block options (not type-specific).
3802 \__zrefclever_get_rf_opt_tl:nxxN { tpairsep }
3803   { \l__zrefclever_label_type_a_tl }
3804   { \l__zrefclever_ref_language_tl }
3805   \l__zrefclever_tpairsep_tl
3806 \__zrefclever_get_rf_opt_tl:nxxN { tlistsep }
3807   { \l__zrefclever_label_type_a_tl }
3808   { \l__zrefclever_ref_language_tl }
3809   \l__zrefclever_tlistsep_tl
3810 \__zrefclever_get_rf_opt_tl:nxxN { tlastsep }
3811   { \l__zrefclever_label_type_a_tl }
3812   { \l__zrefclever_ref_language_tl }
3813   \l__zrefclever_tlastsep_tl
3814
3815 % Process label stack.
3816 \bool_set_false:N \l__zrefclever_typeset_last_bool
3817 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3818 {
3819   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3820   \l__zrefclever_label_a_tl
3821   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3822     {
3823       \tl_clear:N \l__zrefclever_label_b_tl
3824       \bool_set_true:N \l__zrefclever_typeset_last_bool
3825     }
3826     {
3827       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3828       \l__zrefclever_label_b_tl
3829     }
3830
3831   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3832     {
3833       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3834       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3835     }
3836     {
3837       \__zrefclever_extract_default:NVnn
3838       \l__zrefclever_label_type_a_tl
3839       \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3840       \__zrefclever_extract_default:NVnn
3841       \l__zrefclever_label_type_b_tl
3842       \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3843     }

```



```

3844
3845 % First, we establish whether the "current label" (i.e. 'a') is the
3846 % last one of its type. This can happen because the "next label"
3847 % (i.e. 'b') is of a different type (or different definition status),
3848 % or because we are at the end of the list.
3849 \bool_if:NTF \l__zrefclever_typeset_last_bool
3850 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3851 {
3852   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3853   {
3854     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3855     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3856     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3857   }
3858   {
3859     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3860     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3861     {
3862       % Neither is undefined, we must check the types.
3863       \tl_if_eq:NNTF
3864         \l__zrefclever_label_type_a_tl
3865         \l__zrefclever_label_type_b_tl
3866         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3867         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3868     }
3869   }
3870 }
3871
3872 % Handle warnings in case of reference or type undefined.
3873 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3874 \zref@refused { \l__zrefclever_label_a_tl }
3875 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
3876 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3877 {}
3878 {
3879   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3880   {
3881     \msg_warning:nxx { zref-clever } { missing-type }
3882     { \l__zrefclever_label_a_tl }
3883   }
3884   \zref@ifrefcontainsprop
3885     { \l__zrefclever_label_a_tl }
3886     { \l__zrefclever_ref_property_tl }
3887     { }
3888     {
3889       \msg_warning:nxxx { zref-clever } { missing-property }
3890       { \l__zrefclever_ref_property_tl }
3891       { \l__zrefclever_label_a_tl }
3892     }
3893 }
3894
3895 % Get possibly type-specific separators, refbounds, font and other
3896 % options, once per type.
3897 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```

```

3898 {
3899   \__zrefclever_get_rf_opt_tl:nxxN { namesep }
3900   { \l__zrefclever_label_type_a_tl }
3901   { \l__zrefclever_ref_language_tl }
3902   \l__zrefclever_namesep_tl
3903   \__zrefclever_get_rf_opt_tl:nxxN { pairsep }
3904   { \l__zrefclever_label_type_a_tl }
3905   { \l__zrefclever_ref_language_tl }
3906   \l__zrefclever_pairsep_tl
3907   \__zrefclever_get_rf_opt_tl:nxxN { listsep }
3908   { \l__zrefclever_label_type_a_tl }
3909   { \l__zrefclever_ref_language_tl }
3910   \l__zrefclever_listsep_tl
3911   \__zrefclever_get_rf_opt_tl:nxxN { lastsep }
3912   { \l__zrefclever_label_type_a_tl }
3913   { \l__zrefclever_ref_language_tl }
3914   \l__zrefclever_lastsep_tl
3915   \__zrefclever_get_rf_opt_tl:nxxN { rangesep }
3916   { \l__zrefclever_label_type_a_tl }
3917   { \l__zrefclever_ref_language_tl }
3918   \l__zrefclever_rangesep_tl
3919   \__zrefclever_get_rf_opt_tl:nxxN { namefont }
3920   { \l__zrefclever_label_type_a_tl }
3921   { \l__zrefclever_ref_language_tl }
3922   \l__zrefclever_namefont_tl
3923   \__zrefclever_get_rf_opt_tl:nxxN { reffont }
3924   { \l__zrefclever_label_type_a_tl }
3925   { \l__zrefclever_ref_language_tl }
3926   \l__zrefclever_reffont_tl
3927   \__zrefclever_get_rf_opt_tl:nxxN { endrangefunc }
3928   { \l__zrefclever_label_type_a_tl }
3929   { \l__zrefclever_ref_language_tl }
3930   \l__zrefclever_endrangefunc_tl
3931   \__zrefclever_get_rf_opt_tl:nxxN { endrangeprop }
3932   { \l__zrefclever_label_type_a_tl }
3933   { \l__zrefclever_ref_language_tl }
3934   \l__zrefclever_endrangeprop_tl
3935   \__zrefclever_get_rf_opt_bool:nxxxN { cap } { false }
3936   { \l__zrefclever_label_type_a_tl }
3937   { \l__zrefclever_ref_language_tl }
3938   \l__zrefclever_cap_bool
3939   \__zrefclever_get_rf_opt_bool:nxxxN { abbrev } { false }
3940   { \l__zrefclever_label_type_a_tl }
3941   { \l__zrefclever_ref_language_tl }
3942   \l__zrefclever_abbrev_bool
3943   \__zrefclever_get_rf_opt_bool:nxxxN { rangetopair } { true }
3944   { \l__zrefclever_label_type_a_tl }
3945   { \l__zrefclever_ref_language_tl }
3946   \l__zrefclever_rangetopair_bool
3947   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first }
3948   { \l__zrefclever_label_type_a_tl }
3949   { \l__zrefclever_ref_language_tl }
3950   \l__zrefclever_refbounds_first_seq
3951   \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-sg }

```

```

3952         { \l__zrefclever_label_type_a_tl }
3953         { \l__zrefclever_ref_language_tl }
3954         \l__zrefclever_refbounds_first_sg_seq
3955     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-pb }
3956         { \l__zrefclever_label_type_a_tl }
3957         { \l__zrefclever_ref_language_tl }
3958         \l__zrefclever_refbounds_first_pb_seq
3959     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-first-rb }
3960         { \l__zrefclever_label_type_a_tl }
3961         { \l__zrefclever_ref_language_tl }
3962         \l__zrefclever_refbounds_first_rb_seq
3963     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid }
3964         { \l__zrefclever_label_type_a_tl }
3965         { \l__zrefclever_ref_language_tl }
3966         \l__zrefclever_refbounds_mid_seq
3967     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-rb }
3968         { \l__zrefclever_label_type_a_tl }
3969         { \l__zrefclever_ref_language_tl }
3970         \l__zrefclever_refbounds_mid_rb_seq
3971     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-mid-re }
3972         { \l__zrefclever_label_type_a_tl }
3973         { \l__zrefclever_ref_language_tl }
3974         \l__zrefclever_refbounds_mid_re_seq
3975     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last }
3976         { \l__zrefclever_label_type_a_tl }
3977         { \l__zrefclever_ref_language_tl }
3978         \l__zrefclever_refbounds_last_seq
3979     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-pe }
3980         { \l__zrefclever_label_type_a_tl }
3981         { \l__zrefclever_ref_language_tl }
3982         \l__zrefclever_refbounds_last_pe_seq
3983     \__zrefclever_get_rf_opt_seq:nxxN { refbounds-last-re }
3984         { \l__zrefclever_label_type_a_tl }
3985         { \l__zrefclever_ref_language_tl }
3986         \l__zrefclever_refbounds_last_re_seq
3987 }
3988
3989 % Here we send this to a couple of auxiliary functions.
3990 \bool_if:NTF \l__zrefclever_last_of_type_bool
3991 % There exists no next label of the same type as the current.
3992 { \__zrefclever_typeset_refs_last_of_type: }
3993 % There exists a next label of the same type as the current.
3994 { \__zrefclever_typeset_refs_not_last_of_type: }
3995 }
3996 }

```

(End of definition for __zrefclever_typeset_refs:.)

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

the one which does the actual typesetting, while `__zrefclever_typeset_refs_not_last_of_type:` is more of an “accumulation” function.

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

```

3997 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
3998 {
3999   % Process the current label to the current queue.
4000   \int_case:nnF { \l__zrefclever_label_count_int }
4001   {
4002     % It is the last label of its type, but also the first one, and that's
4003     % what matters here: just store it.
4004     % Test: 'zc-typeset01.lvt': "Last of type: single"
4005     { 0 }
4006     {
4007       \tl_set:NV \l__zrefclever_type_first_label_tl
4008       \l__zrefclever_label_a_tl
4009       \tl_set:NV \l__zrefclever_type_first_label_type_tl
4010       \l__zrefclever_label_type_a_tl
4011       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4012       \l__zrefclever_refbounds_first_sg_seq
4013       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4014     }
4015
4016     % The last is the second: we have a pair (if not repeated).
4017     % Test: 'zc-typeset01.lvt': "Last of type: pair"
4018     { 1 }
4019     {
4020       \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4021       {
4022         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4023         \l__zrefclever_refbounds_first_sg_seq
4024         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4025       }
4026       {
4027         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4028         {
4029           \exp_not:V \l__zrefclever_pairsep_tl
4030           \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4031           \l__zrefclever_refbounds_last_pe_seq
4032         }
4033         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4034         \l__zrefclever_refbounds_first_pb_seq
4035         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4036       }
4037     }
4038   }
4039   % Last is third or more of its type: without repetition, we'd have the
4040   % last element on a list, but control for possible repetition.
4041   {
4042     \int_case:nnF { \l__zrefclever_range_count_int }
4043     {
4044       % There was no range going on.
4045       % Test: 'zc-typeset01.lvt': "Last of type: not range"
4046       { 0 }

```

```

4047 {
4048   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4049   {
4050     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4051     {
4052       \exp_not:V \l__zrefclever_pairsep_tl
4053       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4054       \l__zrefclever_refbounds_last_pe_seq
4055     }
4056   }
4057   {
4058     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4059     {
4060       \exp_not:V \l__zrefclever_lastsep_tl
4061       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4062       \l__zrefclever_refbounds_last_seq
4063     }
4064   }
4065 }
4066 % Last in the range is also the second in it.
4067 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
4068 { 1 }
4069 {
4070   \int_compare:nNnTF
4071   { \l__zrefclever_range_same_count_int } = { 1 }
4072   {
4073     % We know 'range_beg_is_first_bool' is false, since this is
4074     % the second element in the range, but the third or more in
4075     % the type list.
4076     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4077     {
4078       \exp_not:V \l__zrefclever_pairsep_tl
4079       \__zrefclever_get_ref:VN
4080       \l__zrefclever_range_beg_label_tl
4081       \l__zrefclever_refbounds_last_pe_seq
4082     }
4083     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4084     \l__zrefclever_refbounds_first_pb_seq
4085     \bool_set_true:N
4086     \l__zrefclever_type_first_refbounds_set_bool
4087   }
4088   {
4089     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4090     {
4091       \exp_not:V \l__zrefclever_listsep_tl
4092       \__zrefclever_get_ref:VN
4093       \l__zrefclever_range_beg_label_tl
4094       \l__zrefclever_refbounds_mid_seq
4095       \exp_not:V \l__zrefclever_lastsep_tl
4096       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4097       \l__zrefclever_refbounds_last_seq
4098     }
4099   }
4100 }

```

```

4101 }
4102 % Last in the range is third or more in it.
4103 {
4104   \int_case:nnF
4105   {
4106     \l__zrefclever_range_count_int -
4107     \l__zrefclever_range_same_count_int
4108   }
4109   {
4110     % Repetition, not a range.
4111     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
4112     { 0 }
4113     {
4114       % If 'range_beg_is_first_bool' is true, it means it was also
4115       % the first of the type, and hence its typesetting was
4116       % already handled, and we just have to set rebounds.
4117       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4118       {
4119         \seq_set_eq:NN \l__zrefclever_type_first_rebounds_seq
4120         \l__zrefclever_rebounds_first_sg_seq
4121         \bool_set_true:N
4122         \l__zrefclever_type_first_rebounds_set_bool
4123       }
4124       {
4125         \int_compare:nNnTF
4126         { \l__zrefclever_ref_count_int } < { 2 }
4127         {
4128           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4129           {
4130             \exp_not:V \l__zrefclever_pairsep_tl
4131             \__zrefclever_get_ref:VN
4132             \l__zrefclever_range_beg_label_tl
4133             \l__zrefclever_rebounds_last_pe_seq
4134           }
4135         }
4136         {
4137           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4138           {
4139             \exp_not:V \l__zrefclever_lastsep_tl
4140             \__zrefclever_get_ref:VN
4141             \l__zrefclever_range_beg_label_tl
4142             \l__zrefclever_rebounds_last_seq
4143           }
4144         }
4145       }
4146     }
4147     % A 'range', but with no skipped value, treat as pair if range
4148     % started with first of type, otherwise as list.
4149     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
4150     { 1 }
4151     {
4152       % Ditto.
4153       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4154       {

```

```

4155         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4156         \l__zrefclever_refbounds_first_pb_seq
4157     \bool_set_true:N
4158         \l__zrefclever_type_first_refbounds_set_bool
4159     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4160     {
4161         \exp_not:V \l__zrefclever_pairsep_tl
4162         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4163         \l__zrefclever_refbounds_last_pe_seq
4164     }
4165 }
4166 {
4167     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4168     {
4169         \exp_not:V \l__zrefclever_listsep_tl
4170         \__zrefclever_get_ref:VN
4171         \l__zrefclever_range_beg_label_tl
4172         \l__zrefclever_refbounds_mid_seq
4173     }
4174     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4175     {
4176         \exp_not:V \l__zrefclever_lastsep_tl
4177         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4178         \l__zrefclever_refbounds_last_seq
4179     }
4180 }
4181 }
4182 }
4183 {
4184     % An actual range.
4185     % Test: 'zc-typeset01.lvt': "Last of type: range"
4186     % Ditto.
4187     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4188     {
4189         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4190         \l__zrefclever_refbounds_first_rb_seq
4191         \bool_set_true:N
4192         \l__zrefclever_type_first_refbounds_set_bool
4193     }
4194     {
4195         \int_compare:nNnTF
4196         { \l__zrefclever_ref_count_int } < { 2 }
4197         {
4198             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4199             {
4200                 \exp_not:V \l__zrefclever_pairsep_tl
4201                 \__zrefclever_get_ref:VN
4202                 \l__zrefclever_range_beg_label_tl
4203                 \l__zrefclever_refbounds_mid_rb_seq
4204             }
4205             \seq_set_eq:NN
4206             \l__zrefclever_type_first_refbounds_seq
4207             \l__zrefclever_refbounds_first_pb_seq
4208             \bool_set_true:N

```

```

4209         \l__zrefclever_type_first_refbounds_set_bool
4210     }
4211     {
4212         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4213         {
4214             \exp_not:V \l__zrefclever_lastsep_tl
4215             \__zrefclever_get_ref:VN
4216             \l__zrefclever_range_beg_label_tl
4217             \l__zrefclever_refbounds_mid_rb_seq
4218         }
4219     }
4220 }
4221 \bool_lazy_and:nnTF
4222 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4223 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4224 {
4225     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4226     \l__zrefclever_range_beg_label_tl
4227     \l__zrefclever_label_a_tl
4228     \l__zrefclever_range_end_ref_tl
4229     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4230     {
4231         \exp_not:V \l__zrefclever_rangesep_tl
4232         \__zrefclever_get_ref_endrange:VVN
4233         \l__zrefclever_label_a_tl
4234         \l__zrefclever_range_end_ref_tl
4235         \l__zrefclever_refbounds_last_re_seq
4236     }
4237 }
4238 {
4239     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4240     {
4241         \exp_not:V \l__zrefclever_rangesep_tl
4242         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4243         \l__zrefclever_refbounds_last_re_seq
4244     }
4245 }
4246 }
4247 }
4248 }
4249
4250 % Handle "range" option. The idea is simple: if the queue is not empty,
4251 % we replace it with the end of the range (or pair). We can still
4252 % retrieve the end of the range from 'label_a' since we know to be
4253 % processing the last label of its type at this point.
4254 \bool_if:NT \l__zrefclever_typeset_range_bool
4255 {
4256     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4257     {
4258         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4259         { }
4260         {
4261             \msg_warning:nmx { zref-clever } { single-element-range }
4262             { \l__zrefclever_type_first_label_type_tl }

```



```

4263     }
4264 }
4265 {
4266   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4267   \bool_if:NT \l__zrefclever_rangetopair_bool
4268   {
4269     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4270     { }
4271     {
4272       \__zrefclever_labels_in_sequence:nn
4273       { \l__zrefclever_type_first_label_tl }
4274       { \l__zrefclever_label_a_tl }
4275     }
4276   }
4277   % Test: 'zc-typeset01.lvt': "Last of type: option range"
4278   % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
4279   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4280   {
4281     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4282     {
4283       \exp_not:V \l__zrefclever_pairsep_tl
4284       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4285       \l__zrefclever_refbounds_last_pe_seq
4286     }
4287     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4288     \l__zrefclever_refbounds_first_pb_seq
4289     \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4290   }
4291   {
4292     \bool_lazy_and:nnTF
4293     { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4294     { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4295     {
4296       % We must get 'type_first_label_tl' instead of
4297       % 'range_beg_label_tl' here, since it is not necessary
4298       % that the first of type was actually starting a range for
4299       % the 'range' option to be used.
4300       \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4301       \l__zrefclever_type_first_label_tl
4302       \l__zrefclever_label_a_tl
4303       \l__zrefclever_range_end_ref_tl
4304       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4305       {
4306         \exp_not:V \l__zrefclever_rangesep_tl
4307         \__zrefclever_get_ref_endrange:VVN
4308         \l__zrefclever_label_a_tl
4309         \l__zrefclever_range_end_ref_tl
4310         \l__zrefclever_refbounds_last_re_seq
4311       }
4312     }
4313   }
4314   \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4315   {
4316     \exp_not:V \l__zrefclever_rangesep_tl

```

```

4317         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4318         \l__zrefclever_refbounds_last_re_seq
4319     }
4320 }
4321 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4322 \l__zrefclever_refbounds_first_rb_seq
4323 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4324 }
4325 }
4326 }
4327
4328 % If none of the special cases for the first of type refbounds have been
4329 % set, do it.
4330 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4331 {
4332     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4333     \l__zrefclever_refbounds_first_seq
4334 }
4335
4336 % Now that the type block is finished, we can add the name and the first
4337 % ref to the queue. Also, if "typeset" option is not "both", handle it
4338 % here as well.
4339 \_zrefclever_type_name_setup:
4340 \bool_if:nTF
4341 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4342 {
4343     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4344     { \_zrefclever_get_ref_first: }
4345 }
4346 {
4347     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4348     {
4349         % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
4350         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4351         {
4352             \_zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4353             \l__zrefclever_type_first_refbounds_seq
4354         }
4355     }
4356     {
4357         \bool_if:NTF \l__zrefclever_typeset_name_bool
4358         {
4359             % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
4360             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
4361             {
4362                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4363                 {
4364                     \exp_not:N \group_begin:
4365                     \exp_not:V \l__zrefclever_namefont_tl
4366                     \_zrefclever_hyperlink:nnn
4367                     {
4368                         \_zrefclever_extract_url_unexp:V
4369                         \l__zrefclever_type_first_label_tl
4370                     }

```

```

4371         {
4372             \__zrefclever_extract_unexp:Vnn
4373             \l__zrefclever_type_first_label_tl
4374             { anchor } { }
4375         }
4376         { \exp_not:V \l__zrefclever_type_name_tl }
4377     \exp_not:N \group_end:
4378 }
4379 {
4380     \exp_not:N \group_begin:
4381     \exp_not:V \l__zrefclever_namefont_tl
4382     \exp_not:V \l__zrefclever_type_name_tl
4383     \exp_not:N \group_end:
4384 }
4385 }
4386 }
4387 {
4388     % Logically, this case would correspond to "typeset=none", but
4389     % it should not occur, given that the options are set up to
4390     % typeset either "ref" or "name". Still, leave here a
4391     % sensible fallback, equal to the behavior of "both".
4392     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
4393     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
4394         { \__zrefclever_get_ref_first: }
4395 }
4396 }
4397 }
4398
4399 % Typeset the previous type block, if there is one.
4400 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4401 {
4402     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4403     { \l__zrefclever_tlistsep_tl }
4404     \l__zrefclever_typeset_queue_prev_tl
4405 }
4406
4407 % Extra log for testing.
4408 \bool_if:NT \l__zrefclever_verbose_testing_bool
4409 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4410
4411 % Wrap up loop, or prepare for next iteration.
4412 \bool_if:NTF \l__zrefclever_typeset_last_bool
4413 {
4414     % We are finishing, typeset the current queue.
4415     \int_case:nnF { \l__zrefclever_type_count_int }
4416     {
4417         % Single type.
4418         % Test: 'zc-typeset01.lvt': "Last of type: single type"
4419         { 0 }
4420         { \l__zrefclever_typeset_queue_curr_tl }
4421         % Pair of types.
4422         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
4423         { 1 }
4424         {

```

```

4425         \l__zrefclever_tpairsep_tl
4426         \l__zrefclever_typeset_queue_curr_tl
4427     }
4428 }
4429 {
4430     % Last in list of types.
4431     % Test: 'zc-typeset01.lvt': "Last of type: list of types"
4432     \l__zrefclever_tlastsep_tl
4433     \l__zrefclever_typeset_queue_curr_tl
4434 }
4435 % And nudge in case of multitype reference.
4436 \bool_lazy_all:nT
4437 {
4438     { \l__zrefclever_nudge_enabled_bool }
4439     { \l__zrefclever_nudge_multitype_bool }
4440     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4441 }
4442 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4443 }
4444 {
4445     % There are further labels, set variables for next iteration.
4446     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4447         \l__zrefclever_typeset_queue_curr_tl
4448     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4449     \tl_clear:N \l__zrefclever_type_first_label_tl
4450     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4451     \tl_clear:N \l__zrefclever_range_beg_label_tl
4452     \tl_clear:N \l__zrefclever_range_end_ref_tl
4453     \int_zero:N \l__zrefclever_label_count_int
4454     \int_zero:N \l__zrefclever_ref_count_int
4455     \int_incr:N \l__zrefclever_type_count_int
4456     \int_zero:N \l__zrefclever_range_count_int
4457     \int_zero:N \l__zrefclever_range_same_count_int
4458     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4459     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4460 }
4461 }

```

(End of definition for __zrefclever_typeset_refs_last_of_type:.)

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

```

4462 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4463 {
4464     % Signal if next label may form a range with the current one (only
4465     % considered if compression is enabled in the first place).
4466     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4467     \bool_set_false:N \l__zrefclever_next_is_same_bool
4468     \bool_if:NT \l__zrefclever_typeset_compress_bool
4469     {
4470         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4471         { }
4472         {
4473             \__zrefclever_labels_in_sequence:nn
4474             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4475     }
4476 }
4477
4478 % Process the current label to the current queue.
4479 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4480 {
4481     % Current label is the first of its type (also not the last, but it
4482     % doesn't matter here): just store the label.
4483     \tl_set:NV \l__zrefclever_type_first_label_tl
4484         \l__zrefclever_label_a_tl
4485     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4486         \l__zrefclever_label_type_a_tl
4487     \int_incr:N \l__zrefclever_ref_count_int
4488
4489     % If the next label may be part of a range, signal it (we deal with it
4490     % as the "first", and must do it there, to handle hyperlinking), but
4491     % also step the range counters.
4492     % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
4493     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4494     {
4495         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4496         \tl_set:NV \l__zrefclever_range_beg_label_tl
4497             \l__zrefclever_label_a_tl
4498         \tl_clear:N \l__zrefclever_range_end_ref_tl
4499         \int_incr:N \l__zrefclever_range_count_int
4500         \bool_if:NT \l__zrefclever_next_is_same_bool
4501             { \int_incr:N \l__zrefclever_range_same_count_int }
4502     }
4503 }
4504 {
4505     % Current label is neither the first (nor the last) of its type.
4506     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4507     {
4508         % Starting, or continuing a range.
4509         \int_compare:nNnTF
4510             { \l__zrefclever_range_count_int } = { 0 }
4511         {
4512             % There was no range going, we are starting one.
4513             \tl_set:NV \l__zrefclever_range_beg_label_tl
4514                 \l__zrefclever_label_a_tl
4515             \tl_clear:N \l__zrefclever_range_end_ref_tl
4516             \int_incr:N \l__zrefclever_range_count_int
4517             \bool_if:NT \l__zrefclever_next_is_same_bool
4518                 { \int_incr:N \l__zrefclever_range_same_count_int }
4519         }
4520         {
4521             % Second or more in the range, but not the last.
4522             \int_incr:N \l__zrefclever_range_count_int
4523             \bool_if:NT \l__zrefclever_next_is_same_bool
4524                 { \int_incr:N \l__zrefclever_range_same_count_int }
4525         }
4526     }
4527 }
4528 % Next element is not in sequence: there was no range, or we are

```

```

4529 % closing one.
4530 \int_case:nnF { \l__zrefclever_range_count_int }
4531 {
4532   % There was no range going on.
4533   % Test: 'zc-typeset01.lvt': "Not last of type: no range"
4534   { 0 }
4535   {
4536     \int_incr:N \l__zrefclever_ref_count_int
4537     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4538     {
4539       \exp_not:V \l__zrefclever_listsep_tl
4540       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4541       \l__zrefclever_refbounds_mid_seq
4542     }
4543   }
4544   % Last is second in the range: if 'range_same_count' is also
4545   % '1', it's a repetition (drop it), otherwise, it's a "pair
4546   % within a list", treat as list.
4547   % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
4548   % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
4549   { 1 }
4550   {
4551     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4552     {
4553       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4554       \l__zrefclever_refbounds_first_seq
4555       \bool_set_true:N
4556       \l__zrefclever_type_first_refbounds_set_bool
4557     }
4558     {
4559       \int_incr:N \l__zrefclever_ref_count_int
4560       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4561       {
4562         \exp_not:V \l__zrefclever_listsep_tl
4563         \__zrefclever_get_ref:VN
4564         \l__zrefclever_range_beg_label_tl
4565         \l__zrefclever_refbounds_mid_seq
4566       }
4567     }
4568     \int_compare:nNnF
4569     { \l__zrefclever_range_same_count_int } = { 1 }
4570     {
4571       \int_incr:N \l__zrefclever_ref_count_int
4572       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4573       {
4574         \exp_not:V \l__zrefclever_listsep_tl
4575         \__zrefclever_get_ref:VN
4576         \l__zrefclever_label_a_tl
4577         \l__zrefclever_refbounds_mid_seq
4578       }
4579     }
4580   }
4581 }
4582 {

```

```

4583 % Last is third or more in the range: if 'range_count' and
4584 % 'range_same_count' are the same, its a repetition (drop it),
4585 % if they differ by '1', its a list, if they differ by more,
4586 % it is a real range.
4587 \int_case:nnF
4588 {
4589   \l__zrefclever_range_count_int -
4590   \l__zrefclever_range_same_count_int
4591 }
4592 {
4593   % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
4594   { 0 }
4595   {
4596     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4597     {
4598       \seq_set_eq:NN
4599       \l__zrefclever_type_first_refbounds_seq
4600       \l__zrefclever_refbounds_first_seq
4601       \bool_set_true:N
4602       \l__zrefclever_type_first_refbounds_set_bool
4603     }
4604     {
4605       \int_incr:N \l__zrefclever_ref_count_int
4606       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4607       {
4608         \exp_not:V \l__zrefclever_listsep_tl
4609         \__zrefclever_get_ref:VN
4610         \l__zrefclever_range_beg_label_tl
4611         \l__zrefclever_refbounds_mid_seq
4612       }
4613     }
4614   }
4615   % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
4616   { 1 }
4617   {
4618     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4619     {
4620       \seq_set_eq:NN
4621       \l__zrefclever_type_first_refbounds_seq
4622       \l__zrefclever_refbounds_first_seq
4623       \bool_set_true:N
4624       \l__zrefclever_type_first_refbounds_set_bool
4625     }
4626     {
4627       \int_incr:N \l__zrefclever_ref_count_int
4628       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4629       {
4630         \exp_not:V \l__zrefclever_listsep_tl
4631         \__zrefclever_get_ref:VN
4632         \l__zrefclever_range_beg_label_tl
4633         \l__zrefclever_refbounds_mid_seq
4634       }
4635     }
4636     \int_incr:N \l__zrefclever_ref_count_int

```

```

4637 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4638 {
4639   \exp_not:V \l__zrefclever_listsep_tl
4640   \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4641   \l__zrefclever_refbounds_mid_seq
4642 }
4643 }
4644 }
4645 {
4646   % Test: 'zc-typeset01.lvt': "Not last of type: range"
4647   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4648   {
4649     \seq_set_eq:NN
4650     \l__zrefclever_type_first_refbounds_seq
4651     \l__zrefclever_refbounds_first_rb_seq
4652     \bool_set_true:N
4653     \l__zrefclever_type_first_refbounds_set_bool
4654   }
4655   {
4656     \int_incr:N \l__zrefclever_ref_count_int
4657     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4658     {
4659       \exp_not:V \l__zrefclever_listsep_tl
4660       \__zrefclever_get_ref:VN
4661       \l__zrefclever_range_beg_label_tl
4662       \l__zrefclever_refbounds_mid_rb_seq
4663     }
4664   }
4665   % For the purposes of the serial comma, and thus for the
4666   % distinction of 'lastsep' and 'pairsep', a "range" counts
4667   % as one. Since 'range_beg' has already been counted
4668   % (here or with the first of type), we refrain from
4669   % incrementing 'ref_count_int'.
4670   \bool_lazy_and:nnTF
4671   { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4672   { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4673   {
4674     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4675     \l__zrefclever_range_beg_label_tl
4676     \l__zrefclever_label_a_tl
4677     \l__zrefclever_range_end_ref_tl
4678     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4679     {
4680       \exp_not:V \l__zrefclever_rangesep_tl
4681       \__zrefclever_get_ref_endrange:VVN
4682       \l__zrefclever_label_a_tl
4683       \l__zrefclever_range_end_ref_tl
4684       \l__zrefclever_refbounds_mid_re_seq
4685     }
4686   }
4687   {
4688     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
4689     {
4690       \exp_not:V \l__zrefclever_rangesep_tl

```



```

4691         \_zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4692         \l__zrefclever_refbounds_mid_re_seq
4693     }
4694 }
4695 }
4696 }
4697 % We just closed a range, reset 'range_beg_is_first' in case a
4698 % second range for the same type occurs, in which case its
4699 % 'range_beg' will no longer be 'first'.
4700 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4701 % Reset counters.
4702 \int_zero:N \l__zrefclever_range_count_int
4703 \int_zero:N \l__zrefclever_range_same_count_int
4704 }
4705 }
4706 % Step label counter for next iteration.
4707 \int_incr:N \l__zrefclever_label_count_int
4708 }

```

(End of definition for `_zrefclever_typeset_refs_not_last_of_type:`)

Auxiliary functions

`_zrefclever_get_ref:nN` and `_zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `_zrefclever_get_ref:nN` handles all references but the first of its type, and `_zrefclever_get_ref_first:` deals with the first reference of a type. Saying they do “typesetting” is imprecise though, they actually prepare material to be accumulated in `\l__zrefclever_typeset_queue_curr_tl` inside `_zrefclever_typeset_refs_last_of_type:` and `_zrefclever_typeset_refs_not_last_of_type:`. And this difference results quite crucial for the \TeX nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `_zrefclever_get_ref:nN` and `_zrefclever_get_ref_first:` get called, as they must, in the context of `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.

```

4709 \cs_new_protected:Npn \_zrefclever_ref_default:
4710 { \zref@default }
4711 \cs_new_protected:Npn \_zrefclever_name_default:
4712 { \zref@default }

```

(End of definition for `_zrefclever_ref_default:` and `_zrefclever_name_default:`)

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

```

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

4713 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4714 {
4715   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4716   {
4717     \bool_if:nTF
4718     {
4719       \l__zrefclever_hyperlink_bool &&
4720       ! \l__zrefclever_link_star_bool
4721     }
4722     {
4723       \seq_item:Nn #2 { 1 }
4724       \__zrefclever_hyperlink:nnn
4725       { \__zrefclever_extract_url_unexp:n {#1} }
4726       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4727       {
4728         \seq_item:Nn #2 { 2 }
4729         \exp_not:N \group_begin:
4730         \exp_not:V \l__zrefclever_reffont_tl
4731         \__zrefclever_extract_unexp:nvn {#1}
4732         { \l__zrefclever_ref_property_tl } { }
4733         \exp_not:N \group_end:
4734         \seq_item:Nn #2 { 3 }
4735       }
4736       \seq_item:Nn #2 { 4 }
4737     }
4738     {
4739       \seq_item:Nn #2 { 1 }
4740       \seq_item:Nn #2 { 2 }
4741       \exp_not:N \group_begin:
4742       \exp_not:V \l__zrefclever_reffont_tl
4743       \__zrefclever_extract_unexp:nvn {#1}
4744       { \l__zrefclever_ref_property_tl } { }
4745       \exp_not:N \group_end:
4746       \seq_item:Nn #2 { 3 }
4747       \seq_item:Nn #2 { 4 }
4748     }
4749   }
4750   { \__zrefclever_ref_default: }
4751 }
4752 \cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }

```

(End of definition for `_zrefclever_get_ref:nN`.)

```

\__zrefclever_get_ref_endrange:nnN
\__zrefclever_get_ref_endrange:nnN {<label>} {<reference>} {<refbounds>}
4753 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4754 {
4755   \str_if_eq:nnTF {#2} { zc@missingproperty }
4756   { \__zrefclever_ref_default: }
4757   {
4758     \bool_if:nTF
4759     {
4760       \l__zrefclever_hyperlink_bool &&
4761       ! \l__zrefclever_link_star_bool
4762     }
4763     {
4764       \seq_item:Nn #3 { 1 }
4765       \__zrefclever_hyperlink:nnn
4766       { \__zrefclever_extract_url_unexp:n {#1} }
4767       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4768       {
4769         \seq_item:Nn #3 { 2 }
4770         \exp_not:N \group_begin:
4771         \exp_not:V \l__zrefclever_reffont_tl
4772         \exp_not:n {#2}
4773         \exp_not:N \group_end:
4774         \seq_item:Nn #3 { 3 }
4775       }
4776       \seq_item:Nn #3 { 4 }
4777     }
4778     {
4779       \seq_item:Nn #3 { 1 }
4780       \seq_item:Nn #3 { 2 }
4781       \exp_not:N \group_begin:
4782       \exp_not:V \l__zrefclever_reffont_tl
4783       \exp_not:n {#2}
4784       \exp_not:N \group_end:
4785       \seq_item:Nn #3 { 3 }
4786       \seq_item:Nn #3 { 4 }
4787     }
4788   }
4789 }
4790 \cs_generate_variant:Nn \__zrefclever_get_ref_endrange:nnN { VVN }

```

(End of definition for __zrefclever_get_ref_endrange:nnN.)

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

```

4791 \cs_new:Npn \__zrefclever_get_ref_first:
4792 {
4793   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4794   { \__zrefclever_ref_default: }
4795   {
4796     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4797 {
4798   \zref@ifrefcontainsprop
4799   { \l__zrefclever_type_first_label_tl }
4800   { \l__zrefclever_ref_property_tl }
4801   {
4802     \__zrefclever_hyperlink:nnn
4803     {
4804       \__zrefclever_extract_url_unexp:V
4805       \l__zrefclever_type_first_label_tl
4806     }
4807     {
4808       \__zrefclever_extract_unexp:Vnn
4809       \l__zrefclever_type_first_label_tl { anchor } { }
4810     }
4811     {
4812       \exp_not:N \group_begin:
4813       \exp_not:V \l__zrefclever_namefont_tl
4814       \exp_not:V \l__zrefclever_type_name_tl
4815       \exp_not:N \group_end:
4816       \exp_not:V \l__zrefclever_namesep_tl
4817       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4818       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4819       \exp_not:N \group_begin:
4820       \exp_not:V \l__zrefclever_reffont_tl
4821       \__zrefclever_extract_unexp:Vnn
4822       \l__zrefclever_type_first_label_tl
4823       { \l__zrefclever_ref_property_tl } { }
4824       \exp_not:N \group_end:
4825       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4826     }
4827     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4828   }
4829   {
4830     \exp_not:N \group_begin:
4831     \exp_not:V \l__zrefclever_namefont_tl
4832     \exp_not:V \l__zrefclever_type_name_tl
4833     \exp_not:N \group_end:
4834     \exp_not:V \l__zrefclever_namesep_tl
4835     \__zrefclever_ref_default:
4836   }
4837 }
4838 {
4839   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4840   {
4841     \__zrefclever_name_default:
4842     \exp_not:V \l__zrefclever_namesep_tl
4843   }
4844   {
4845     \exp_not:N \group_begin:
4846     \exp_not:V \l__zrefclever_namefont_tl
4847     \exp_not:V \l__zrefclever_type_name_tl
4848     \exp_not:N \group_end:
4849     \tl_if_empty:NF \l__zrefclever_type_name_tl
4850     { \exp_not:V \l__zrefclever_namesep_tl }

```

```

4851     }
4852     \zref@ifrefcontainsprop
4853     { \l__zrefclever_type_first_label_tl }
4854     { \l__zrefclever_ref_property_tl }
4855     {
4856         \bool_if:nTF
4857         {
4858             \l__zrefclever_hyperlink_bool &&
4859             ! \l__zrefclever_link_star_bool
4860         }
4861         {
4862             \seq_item:Nn
4863             \l__zrefclever_type_first_refbounds_seq { 1 }
4864             \__zrefclever_hyperlink:nnn
4865             {
4866                 \__zrefclever_extract_url_unexp:V
4867                 \l__zrefclever_type_first_label_tl
4868             }
4869             {
4870                 \__zrefclever_extract_unexp:Vnn
4871                 \l__zrefclever_type_first_label_tl { anchor } { }
4872             }
4873             {
4874                 \seq_item:Nn
4875                 \l__zrefclever_type_first_refbounds_seq { 2 }
4876                 \exp_not:N \group_begin:
4877                 \exp_not:V \l__zrefclever_reffont_tl
4878                 \__zrefclever_extract_unexp:Vvn
4879                 \l__zrefclever_type_first_label_tl
4880                 { \l__zrefclever_ref_property_tl } { }
4881                 \exp_not:N \group_end:
4882                 \seq_item:Nn
4883                 \l__zrefclever_type_first_refbounds_seq { 3 }
4884             }
4885             \seq_item:Nn
4886             \l__zrefclever_type_first_refbounds_seq { 4 }
4887         }
4888         {
4889             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4890             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4891             \exp_not:N \group_begin:
4892             \exp_not:V \l__zrefclever_reffont_tl
4893             \__zrefclever_extract_unexp:Vvn
4894             \l__zrefclever_type_first_label_tl
4895             { \l__zrefclever_ref_property_tl } { }
4896             \exp_not:N \group_end:
4897             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4898             \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4899         }
4900     }
4901     { \__zrefclever_ref_default: }
4902 }
4903 }
4904 }

```

(End of definition for _zrefclever_get_ref_first:.)

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

```

4905 \cs_new_protected:Npn \_zrefclever_type_name_setup:
4906 {
4907   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
4908   {
4909     \tl_clear:N \l\_zrefclever_type_name_tl
4910     \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4911   }
4912   {
4913     \tl_if_eq:NnTF
4914       \l\_zrefclever_type_first_label_type_tl { zc@missingtype }
4915       {
4916         \tl_clear:N \l\_zrefclever_type_name_tl
4917         \bool_set_true:N \l\_zrefclever_type_name_missing_bool
4918       }
4919       {
4920         % Determine whether we should use capitalization, abbreviation,
4921         % and plural.
4922         \bool_lazy_or:nnTF
4923           { \l\_zrefclever_cap_bool }
4924           {
4925             \l\_zrefclever_capfirst_bool &&
4926             \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 }
4927           }
4928           { \tl_set:Nn \l\_zrefclever_name_format_tl {Name} }
4929           { \tl_set:Nn \l\_zrefclever_name_format_tl {name} }
4930         % If the queue is empty, we have a singular, otherwise, plural.
4931         \tl_if_empty:NTF \l\_zrefclever_typeset_queue_curr_tl
4932           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -sg } }
4933           { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -pl } }
4934         \bool_lazy_and:nnTF
4935           { \l\_zrefclever_abbrev_bool }
4936           {
4937             ! \int_compare_p:nNn
4938               { \l\_zrefclever_type_count_int } = { 0 } ||
4939             ! \l\_zrefclever_noabbrev_first_bool
4940           }
4941           {
4942             \tl_set:NV \l\_zrefclever_name_format_fallback_tl
4943               \l\_zrefclever_name_format_tl
4944             \tl_put_right:Nn \l\_zrefclever_name_format_tl { -ab }

```

```

4945     }
4946     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4947
4948 % Handle number and gender nudges.
4949 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4950 {
4951     \bool_if:NTF \l__zrefclever_nudge_singular_bool
4952     {
4953         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4954         {
4955             \msg_warning:nxx { zref-clever }
4956             { nudge-plural-when-sg }
4957             { \l__zrefclever_type_first_label_type_tl }
4958         }
4959     }
4960     {
4961         \bool_lazy_all:nT
4962         {
4963             { \l__zrefclever_nudge_comptosing_bool }
4964             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4965             {
4966                 \int_compare_p:nNn
4967                 { \l__zrefclever_label_count_int } > { 0 }
4968             }
4969         }
4970         {
4971             \msg_warning:nxx { zref-clever }
4972             { nudge-comptosing }
4973             { \l__zrefclever_type_first_label_type_tl }
4974         }
4975     }
4976     \bool_lazy_and:nnT
4977     { \l__zrefclever_nudge_gender_bool }
4978     { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
4979     {
4980         \__zrefclever_get_rf_opt_seq:nxxN { gender }
4981         { \l__zrefclever_type_first_label_type_tl }
4982         { \l__zrefclever_ref_language_tl }
4983         \l__zrefclever_type_name_gender_seq
4984         \seq_if_in:NVF
4985         \l__zrefclever_type_name_gender_seq
4986         \l__zrefclever_ref_gender_tl
4987         {
4988             \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
4989             {
4990                 \msg_warning:nxxxx { zref-clever }
4991                 { nudge-gender-not-declared-for-type }
4992                 { \l__zrefclever_ref_gender_tl }
4993                 { \l__zrefclever_type_first_label_type_tl }
4994                 { \l__zrefclever_ref_language_tl }
4995             }
4996             {
4997                 \msg_warning:nxxxxx { zref-clever }
4998                 { nudge-gender-mismatch }

```

```

4999         { \l__zrefclever_type_first_label_type_tl }
5000         { \l__zrefclever_ref_gender_tl }
5001         {
5002             \seq_use:Nn
5003             \l__zrefclever_type_name_gender_seq { , ~ }
5004         }
5005         { \l__zrefclever_ref_language_tl }
5006     }
5007 }
5008 }
5009 }
5010
5011 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5012 {
5013     \__zrefclever_opt_tl_get:cNF
5014     {
5015         \__zrefclever_opt_varname_type:een
5016         { \l__zrefclever_type_first_label_type_tl }
5017         { \l__zrefclever_name_format_tl }
5018         { tl }
5019     }
5020     \l__zrefclever_type_name_tl
5021     {
5022         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5023         {
5024             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5025             \tl_put_left:NV \l__zrefclever_name_format_tl
5026             \l__zrefclever_ref_decl_case_tl
5027         }
5028         \__zrefclever_opt_tl_get:cNF
5029         {
5030             \__zrefclever_opt_varname_lang_type:een
5031             { \l__zrefclever_ref_language_tl }
5032             { \l__zrefclever_type_first_label_type_tl }
5033             { \l__zrefclever_name_format_tl }
5034             { tl }
5035         }
5036         \l__zrefclever_type_name_tl
5037         {
5038             \tl_clear:N \l__zrefclever_type_name_tl
5039             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5040             \msg_warning:nxxx { zref-clever } { missing-name }
5041             { \l__zrefclever_name_format_tl }
5042             { \l__zrefclever_type_first_label_type_tl }
5043         }
5044     }
5045 }
5046 {
5047     \__zrefclever_opt_tl_get:cNF
5048     {
5049         \__zrefclever_opt_varname_type:een
5050         { \l__zrefclever_type_first_label_type_tl }
5051         { \l__zrefclever_name_format_tl }
5052         { tl }

```



```

5053 }
5054 \l__zrefclever_type_name_tl
5055 {
5056   \__zrefclever_opt_tl_get:cNF
5057   {
5058     \__zrefclever_opt_varname_type:een
5059     { \l__zrefclever_type_first_label_type_tl }
5060     { \l__zrefclever_name_format_fallback_tl }
5061     { tl }
5062   }
5063   \l__zrefclever_type_name_tl
5064   {
5065     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5066     {
5067       \tl_put_left:Nn
5068         \l__zrefclever_name_format_tl { - }
5069       \tl_put_left:NV \l__zrefclever_name_format_tl
5070         \l__zrefclever_ref_decl_case_tl
5071       \tl_put_left:Nn
5072         \l__zrefclever_name_format_fallback_tl { - }
5073       \tl_put_left:NV
5074         \l__zrefclever_name_format_fallback_tl
5075         \l__zrefclever_ref_decl_case_tl
5076     }
5077     \__zrefclever_opt_tl_get:cNF
5078     {
5079       \__zrefclever_opt_varname_lang_type:eeen
5080       { \l__zrefclever_ref_language_tl }
5081       { \l__zrefclever_type_first_label_type_tl }
5082       { \l__zrefclever_name_format_tl }
5083       { tl }
5084     }
5085     \l__zrefclever_type_name_tl
5086     {
5087       \__zrefclever_opt_tl_get:cNF
5088       {
5089         \__zrefclever_opt_varname_lang_type:eeen
5090         { \l__zrefclever_ref_language_tl }
5091         { \l__zrefclever_type_first_label_type_tl }
5092         { \l__zrefclever_name_format_fallback_tl }
5093         { tl }
5094       }
5095       \l__zrefclever_type_name_tl
5096       {
5097         \tl_clear:N \l__zrefclever_type_name_tl
5098         \bool_set_true:N
5099           \l__zrefclever_type_name_missing_bool
5100         \msg_warning:nxxx { zref-clever }
5101           { missing-name }
5102         { \l__zrefclever_name_format_tl }
5103         { \l__zrefclever_type_first_label_type_tl }
5104       }
5105     }
5106   }

```

```

5107     }
5108   }
5109 }
5110 }
5111
5112 % Signal whether the type name is to be included in the hyperlink or not.
5113 \bool_lazy_any:nTF
5114 {
5115   { ! \l__zrefclever_hyperlink_bool }
5116   { \l__zrefclever_link_star_bool }
5117   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5118   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5119 }
5120 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5121 {
5122   \bool_lazy_any:nTF
5123   {
5124     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5125     {
5126       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5127       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5128     }
5129     {
5130       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5131       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5132       \l__zrefclever_typeset_last_bool &&
5133       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5134     }
5135   }
5136   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5137   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5138 }
5139 }

```

(End of definition for __zrefclever_type_name_setup:.)

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

```

\__zrefclever_hyperlink:nnn {\url/file}\{\anchor}\{\text}
5140 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5141 {
5142   \tl_if_empty:nTF {#1}
5143   { \hyperlink {#2} {#3} }
5144   { \hyper@linkfile {#3} {#1} {#2} }
5145 }

```

(End of definition for __zrefclever_hyperlink:nnn.)

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

```

5146 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5147 {
5148   \zref@ifpropundefined { urluse }
5149   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5150   {
5151     \zref@ifrefcontainsprop {#1} { urluse }
5152     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5153     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5154   }
5155 }
5156 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End of definition for __zrefclever_extract_url_unexp:n.)

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

```

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

5157 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5158 {
5159   \exp_args:Nxx \tl_if_eq:nnT
5160   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5161   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5162   {
5163     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5164     {
5165       \exp_args:Nxx \tl_if_eq:nnT
5166       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5167       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5168       {
5169         \int_compare:nNnTF
5170         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5171         =
5172         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5173         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5174         {
5175           \int_compare:nNnT
5176           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5177           =
5178           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5179           {
5180             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5181             \bool_set_true:N \l__zrefclever_next_is_same_bool
5182           }
5183         }
5184       }
5185     }
5186   }
5187   \exp_args:Nxx \tl_if_eq:nnT

```

```

5188 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5189 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5190 {
5191   \exp_args:Nxx \tl_if_eq:nnT
5192   { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5193   { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5194   {
5195     \int_compare:nNnTF
5196     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5197     =
5198     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5199     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5200     {
5201       \int_compare:nNnT
5202       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5203       =
5204       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5205       {

```

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.

```

5206   \exp_args:Nxx \tl_if_eq:nnT
5207   {
5208     \_zrefclever_extract_unexp:nvn {#1}
5209     { \l__zrefclever_ref_property_tl } { }
5210   }
5211   {
5212     \_zrefclever_extract_unexp:nvn {#2}
5213     { \l__zrefclever_ref_property_tl } { }
5214   }
5215   {
5216     \bool_set_true:N
5217     \l__zrefclever_next_maybe_range_bool
5218     \bool_set_true:N
5219     \l__zrefclever_next_is_same_bool
5220   }
5221 }
5222 }
5223 }
5224 }
5225 }
5226 }
5227 }

```

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

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

```

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

```

(End of definition for __zrefclever_get_rf_opt_tl:nnnN.)

```

\__zrefclever_get_rf_opt_seq:nnnN {\option}
  {\ref type} {\language} {\seq variable}
5261 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5262 {
5263   % First attempt: general options.
5264   \__zrefclever_opt_seq_get:cNF
5265   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5266   #4
5267   {
5268     % If not found, try type specific options.
5269     \__zrefclever_opt_seq_get:cNF
5270     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5271     #4
5272     {
5273       % If not found, try type- and language-specific.
5274       \__zrefclever_opt_seq_get:cNF

```

```

5275         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5276         #4
5277         {
5278             % If not found, try language-specific default.
5279             \__zrefclever_opt_seq_get:cNF
5280             { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5281             #4
5282             {
5283                 % If not found, try fallback.
5284                 \__zrefclever_opt_seq_get:cNF
5285                 { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5286                 #4
5287                 { \seq_clear:N #4 }
5288             }
5289         }
5290     }
5291 }
5292 }
5293 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { nxxN }

```

(End of definition for __zrefclever_get_rf_opt_seq:nnnN.)

```

\__zrefclever_get_rf_opt_bool:nnnnN

\__zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
{<ref type>} {<language>} {<bool variable>}

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

```

```

5324     }
5325   }
5326   \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nxxxN }

```

(End of definition for `__zrefclever_get_rf_opt_bool:nnnnN`.)

9 Compatibility

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

9.1 appendix

One relevant case of different reference types sharing the same counter is the `\appendix` which in some document classes, including the standard ones, change the sectioning commands looks but, of course, keep using the same counter. `book.cls` and `report.cls` reset counters `chapter` and `section` to 0, change `\@chapapp` to use `\appendixname` and use `\@Alph` for `\thechapter`. `article.cls` resets counters `section` and `subsection` to 0, and uses `\@Alph` for `\thesection`. `memoir.cls`, `scrbook.cls` and `scrarticle.cls` do the same as their corresponding standard classes, and sometimes a little more, but what interests us here is pretty much the same. See also the `appendix` package.

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

```

5327 \__zrefclever_compat_module:nn { appendix }
5328 {
5329   \AddToHook { cmd / appendix / before }
5330   {
5331     \__zrefclever_zcsetup:n
5332     {
5333       countertype =
5334       {
5335         chapter      = appendix ,
5336         section      = appendix ,
5337         subsection   = appendix ,
5338         subsubsection = appendix ,
5339         paragraph    = appendix ,
5340         subparagraph = appendix ,
5341       }
5342     }
5343   }
5344 }

```

Depending on the definition of `\appendix`, using the hook may lead to trouble with the first released version of `ltxcmds` (the one released with the 2021-06-01 kernel). Particularly, if the definition of the command being hooked at contains a double hash mark (`##`) the patch to add the hook, if it needs to be done with the `\scantokens`

method, may fail noisily (see <https://tex.stackexchange.com/q/617905>, with a detailed explanation and possible workaround by Phelype Oleinik). The 2021-11-15 kernel release already handles this gracefully, thanks to fix by Phelype Oleinik at <https://github.com/latex3/latex2e/pull/699>.

9.2 appendices

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

```

5345 \__zrefclever_compat_module:nn { appendices }
5346 {
5347   \__zrefclever_if_package_loaded:nT { appendix }
5348   {
5349     \newcounter { zc@appendix }
5350     \newcounter { zc@save@appendix }
5351     \setcounter { zc@appendix } { 0 }
5352     \setcounter { zc@save@appendix } { 0 }
5353     \cs_if_exist:cTF { chapter }
5354     {
5355       \__zrefclever_zcsetup:n
5356       { counterresetby = { chapter = zc@appendix } }
5357     }
5358     {
5359       \cs_if_exist:cT { section }
5360       {
5361         \__zrefclever_zcsetup:n
5362         { counterresetby = { section = zc@appendix } }
5363       }
5364     }
5365     \AddToHook { env / appendices / begin }
5366     {
5367       \stepcounter { zc@save@appendix }
5368       \setcounter { zc@appendix } { \value { zc@save@appendix } }
5369       \__zrefclever_zcsetup:n
5370       {
5371         countertype =
5372         {
5373           chapter      = appendix ,
5374           section      = appendix ,
5375           subsection   = appendix ,
5376           subsubsection = appendix ,
5377           paragraph    = appendix ,
5378           subparagraph = appendix ,
5379         }
5380       }
5381     }
5382     \AddToHook { env / appendices / end }
5383     { \setcounter { zc@appendix } { 0 } }
5384     \AddToHook { cmd / appendix / before }
5385     {
5386       \stepcounter { zc@save@appendix }
5387       \setcounter { zc@appendix } { \value { zc@save@appendix } }
5388     }

```



```

5389 \AddToHook { env / subappendices / begin }
5390 {
5391   \__zrefclever_zcsetup:n
5392   {
5393     countertype =
5394     {
5395       section      = appendix ,
5396       subsection   = appendix ,
5397       subsubsection = appendix ,
5398       paragraph    = appendix ,
5399       subparagraph = appendix ,
5400     } ,
5401   }
5402 }
5403 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5404 }
5405 }

```

9.3 memoir

The `memoir` document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. Some of them are implemented in ways which make difficult the use of `zref`, particularly `\zlabel`, short of redefining the whole stuff ourselves. Fortunately, however, the kernel's new `label` hook, and the `labelhook` option, make things somewhat easier.

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

```

5406 \__zrefclever_compat_module:nn { memoir }
5407 {
5408   \__zrefclever_if_class_loaded:nT { memoir }
5409   {

```

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.

```

5410 \__zrefclever_zcsetup:n
5411 {
5412     countertype =
5413     {
5414         subfigure = figure ,
5415         subtable  = table  ,
5416         poemline  = line   ,
5417     } ,
5418     counterresetby =
5419     {
5420         subfigure = figure ,
5421         subtable  = table  ,
5422     } ,
5423 }

```

Support for subcaption reference.

```

5424 \zref@newprop { subcaption }
5425 { \cs_if_exist_use:c { @@thesub \@capttype } }
5426 \AddToHook{ cmd / @memsubcaption / before }
5427 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

5428 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
5429 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
5430 \AddToHook { env / minipage / begin }
5431 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
5432 \AddToHook { cmd / @makefntext / before }
5433 {
5434     \__zrefclever_zcsetup:x
5435     { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
5436 }
5437 \AddToHook { cmd / @makesidefntext / before }
5438 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }
5439 \__zrefclever_zcsetup:n
5440 {
5441     countertype =
5442     {
5443         sidefootnote = footnote ,
5444         pagenote     = endnote  ,
5445     } ,
5446 }
5447 \AddToHook { file / \jobname.ent / before }
5448 { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
5449 \msg_info:nnn { zref-clever } { compat-class } { memoir }
5450 }
5451 }

```

9.4 amsmath

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

```

5452 \__zrefclever_compat_module:nn { amsmath }
5453 {
5454   \__zrefclever_if_package_loaded:nT { amsmath }
5455   {

```

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).

```

5456       \bool_new:N \l__zrefclever_amsmath_subequations_bool
5457       \AddToHook { env / subequations / begin }
5458       {
5459         \__zrefclever_zcsetup:x
5460         {
5461           counterresetby =
5462           {
5463             parentequation =
5464               \__zrefclever_counter_reset_by:n { equation } ,
5465             equation = parentequation ,
5466           } ,
5467           currentcounter = parentequation ,
5468           countertype = { parentequation = equation } ,
5469         }
5470       \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5471     }

```

`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.

```

5472       \zref@newprop { subeq } { \alph { equation } }
5473       \clist_map_inline:nn
5474       {
5475         equation ,

```

```

5476     equation* ,
5477     align ,
5478     align* ,
5479     alignat ,
5480     alignat* ,
5481     flalign ,
5482     flalign* ,
5483     xalignat ,
5484     xalignat* ,
5485     gather ,
5486     gather* ,
5487     multiline ,
5488     multiline* ,
5489   }
5490   {
5491     \AddToHook { env / #1 / begin }
5492     {
5493       \__zrefclever_zcsetup:n { currentcounter = equation }
5494       \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5495         { \zref@localaddprop \ZREF@mainlist { subeq } }
5496     }
5497   }
5498   \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5499 }
5500 }

```

9.5 mathtools

All math environments defined by `mathtools`, extending the `amsmath` set, are meant to be used within enclosing math environments, hence we don't need to handle them specially, since the numbering and the counting is being done on the side of `amsmath`. This includes the new `cases` and `matrix` variants, and also `multlined`.

Hence, as far as I can tell, the only cross-reference related feature to deal with is the `showonlyrefs` option, whose machinery involves writing an extra internal label to the `.aux` file to track for labels which get actually referred to. This is a little more involved, and implies in doing special handling inside `\zcref`, but the feature is very cool, so it's worth it.

```

5501 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
5502 \__zrefclever_compat_module:nn { mathtools }
5503 {
5504   \__zrefclever_if_package_loaded:nT { mathtools }
5505   {
5506     \MH_if_boolean:nT { show_only_refs }
5507     {
5508       \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
5509       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5510       {
5511         \@bsphack
5512         \seq_map_inline:Nn #1
5513         {
5514           \exp_args:Nx \tl_if_eq:nnTF
5515             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5516             { equation }

```

```

5517         {
5518             \protected@write \@auxout { }
5519             { \string \MT@newlabel {##1} }
5520         }
5521         {
5522             \exp_args:Nx \tl_if_eq:nnT
5523             { \_zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5524             { parentequation }
5525             {
5526                 \protected@write \@auxout { }
5527                 { \string \MT@newlabel {##1} }
5528             }
5529         }
5530     }
5531     \esphack
5532 }
5533 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5534 }
5535 }
5536 }

```

9.6 breqn

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for `\zlabel` just as well.

```

5537 \_zrefclever_compat_module:nn { breqn }
5538 {
5539     \_zrefclever_if_package_loaded:nT { breqn }
5540     {

```

Contrary to the practice in `amsmath`, which prints `\tag` even in unnumbered environments, the starred environments from `breqn` don’t typeset any tag/number at all, even for a manually given `number=` as an option. So, even if one can actually set a label in them, it is not really meaningful to make a reference to them. Also contrary to `amsmath`’s practice, `breqn` uses `\stepcounter` instead of `\refstepcounter` for incrementing the equation counters (see <https://tex.stackexchange.com/a/241150>).

```

5541     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5542     \AddToHook { env / dgroup / begin }
5543     {
5544         \_zrefclever_zcsetup:x
5545         {
5546             counterresetby =
5547             {
5548                 parentequation =
5549                 \_zrefclever_counter_reset_by:n { equation } ,
5550                 equation = parentequation ,
5551             } ,
5552             currentcounter = parentequation ,
5553             countertype = { parentequation = equation } ,
5554         }
5555     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5556 }

```

```

5557 \zref@ifpropundefined { subeq }
5558 { \zref@newprop { subeq } { \alph { equation } } }
5559 { }
5560 \clist_map_inline:nn
5561 {
5562     dmath ,
5563     dseries ,
5564     darray ,
5565 }
5566 {
5567     \AddToHook { env / #1 / begin }
5568     {
5569         \__zrefclever_zcsetup:n { currentcounter = equation }
5570         \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5571         { \zref@localaddprop \ZREF@mainlist { subeq } }
5572     }
5573 }
5574 \msg_info:nnn { zref-clever } { compat-package } { breqn }
5575 }
5576 }

```

9.7 listings

```

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

```

Set `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.

```

5590 \lst@AddToHook { Init }
5591 { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5592 \msg_info:nnn { zref-clever } { compat-package } { listings }
5593 }
5594 }

```

9.8 enumitem

The procedure below will “see” any changes made to the `enumerate` environment (made with `enumitem`’s `\renewlist`) as long as it is done in the preamble. Though, technically,

`\renewlist` can be issued anywhere in the document, this should be more than enough for the purpose at hand. Besides, trying to retrieve this information “on the fly” would be much overkill.

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

```

5595 \__zrefclever_compat_module:nn { enumitem }
5596 {
5597   \__zrefclever_if_package_loaded:nT { enumitem }
5598   {
5599     \int_set:Nn \l_tmpa_int { 5 }
5600     \bool_while_do:nn
5601     {
5602       \cs_if_exist_p:c
5603       { c@ enum \int_to_roman:n { \l_tmpa_int } }
5604     }
5605     {
5606       \__zrefclever_zcsetup:x
5607       {
5608         counterresetby =
5609         {
5610           enum \int_to_roman:n { \l_tmpa_int } =
5611           enum \int_to_roman:n { \l_tmpa_int - 1 }
5612         } ,
5613         countertype =
5614         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
5615       }
5616       \int_incr:N \l_tmpa_int
5617     }
5618     \int_compare:nNnT { \l_tmpa_int } > { 5 }
5619     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5620   }
5621 }

```

9.9 subcaption

```

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

```

```

5634         {
5635             subfigure = figure ,
5636             subtable = table ,
5637         } ,
5638     }

```

Support for subref reference.

```

5639     \zref@newprop { subref }
5640     { \cs_if_exist_use:c { thesub \@capytype } }
5641     \tl_put_right:Nn \caption@subtypehook
5642     { \zref@localaddprop \ZREF@mainlist { subref } }
5643 }
5644 }

```

9.10 subfig

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

```

5645 \__zrefclever_compat_module:nn { subfig }
5646 {
5647     \__zrefclever_if_package_loaded:nT { subfig }
5648     {
5649         \__zrefclever_zcsetup:n
5650         {
5651             countertype =
5652             {
5653                 subfigure = figure ,
5654                 subtable = table ,
5655             } ,
5656             counterresetby =
5657             {
5658                 subfigure = figure ,
5659                 subtable = table ,
5660             } ,
5661         }
5662     }
5663 }
5664 \</package>

```

10 Language files

Initial values for the English, German, French, Portuguese, and Spanish language files have been provided by the author. Translations available for document elements' names in other packages have been an useful reference for the purpose, namely: `babel`, `cleveref`, `translator`, and `translations`.

10.1 Localization guidelines

Since the task of localizing `zref-clever` to work in different languages depends on the generous work of contributors, it is a good idea to set some guidelines not only to ease the task itself but also to document what the package expects in this regard.

The first general observation is that, contrary to a common initial reaction of those faced with the task of localizing the reference types, is that the job is not quite one of

“translation”. The reference type names are just the internal names used by the package to refer to them, technically, they could just as well be foobars. Of course, for practical reasons, they were chosen to be semantic. However, what we are searching for is not really the translation to the reference type name itself, but rather for the word / term / expression which is typically used to refer to the document object that the reference type is meant to represent. And terms that should work well in the contexts which cross-references are commonly used.

That said, some comments about the reference types and common pitfalls.

Sectioning: A number of reference types are provided to support referencing to document sectioning commands. Obviously, `part`, `chapter`, `section`, and `paragraph` are meant to refer to the sectioning commands of the standard classes and elsewhere, which anyone reading this is certainly acquainted with. Note that `zref-clever` uses – by default at least, which is what the language files cater for – the `section` reference type to refer to `\subsections` and `\subsubsections` as well, similarly, `paragraph` also refers to `\subparagraph`. The `appendix` reference type is meant to refer to any sectioning command – be them chapters, sections, or paragraphs – issued after `\appendix`, which corresponds to how the standard classes, the KOMA Script classes, and `memoir` deal with appendices. The `book` reference type deserves some explanation. The word “book” has a good number of meanings, and the most common one is not the one which is intended here. The Webster dictionary gives us a couple of definitions of interest: “1. A collection of sheets of paper, or similar material, blank, written, or printed, bound together; commonly, many folded and bound sheets containing continuous printing or writing.” and “3. A part or subdivision of a treatise or literary work; as, the tenth book of ‘Paradise Lost.’” It is this third meaning which the `book` reference type is meant to support: a major subdivision of a work, much like `\part`. Even if it does not exist in the standard classes, it may exist elsewhere, in particular, it is provided by `memoir`.

Common numbered objects: Nothing surprising here, just being explicit. `table` and `figure` refer to the document’s respective floats objects. `page` to the page number. `item` to the item number in `enumerate` environments. Similarly, `line` is meant to refer to line numbers.

Notes: `zref-clever` provides three reference types in this area: `footnote`, `endnote`, and `note`. The first two refer to footnotes and end notes, respectively. The third is meant as a convenience for a general “note” object, either the other two, or something else. By experience, here is one place where that initial observation of not simply translating the reference types names is particularly relevant. There’s a natural temptation, because three different types exist and are somewhat close to each other, to distinguish them clearly. Duty would compel us to do so. But that may lead to less than ideal results. Different terms work well for some languages, like English and German, which have compound words for the purpose. But less so for other languages, like Portuguese, French, or Italian. For example, in a document in French which only contains footnotes, arguably a very common use case, would it be better to refer to a footnote as just “note”, or be very precise with “note infrapaginale”? Of course, in a document which contains both footnotes and end notes, we may need the distinction. But is it really the better default? True, possibly the inclusion of the `note` reference type, with no clear object to refer to, creates more noise than convenience here. If I recall correctly, my intention was to provide an easy way out for users from possible contentious localizations for `footnote` and `endnote`, but I’m not sure if it’s been working like this in practice, and I should probably have refrained from adding it in the first place.

Math & Co.: A good number of reference types provided by the package are meant to cater for document objects commonly used in Mathematics and related areas. They

are either straight math environments, defined by the kernel, `amsmath` or other packages, or environments which are normally not pre-defined by the kernel or the standard classes, but are traditionally defined by users with the kernel's `\newtheorem` or similar constructs available in the \LaTeX package ecosystem. For most of them, localization should strive as much as possible to use the formal terms, jargon really, typically employed by mathematicians, logicians, and friends. Namely for the reference types: `equation`, `theorem`, `lemma`, `corollary`, `proposition`, `definition`, `proof`, `result`, and `remark`. Regarding `example`, `exercise`, and `solution` being somewhat less formal is admissible. But the chosen terms should still be fit for use in Math related contexts, and should be assumed were created by `\newtheorem` or similar, even if users may well find other uses for these types.

Code: A couple of reference types are provided for code related environments: `algorithm` and `listing`. By experience, the `listing` type has already proven to be a particularly challenging one. Formally, it should be a good default term to encompass anything which may regularly be included in a `lstlisting` environment as provided by the `listings` package. However, it seems that in different languages it is quite difficult to find a satisfying term for it. Though my English is decent, I'm not a native speaker, still I'm not even sure how common the term is used for the purpose even in English. It seems to be traditional enough in the \LaTeX community at least. In doubt, pend to the jargon side, anglicism if need be. Since we are bound to displease mostly everyone anyway, at least we do so in a consistent manner.

Completeness and abbreviated forms: Ideally, the language file should be as complete as possible. "Complete" meaning it contains: i) the defaults for all basic separators, `namesep`, `pairsep`, `listsep`, `lastsep`, `tpairsep`, `tlistsep`, `tlastsep`, `notesep`, and `rangesep`; ii) the non-abbreviated forms of names for all the supported reference types, according to the language definitions, that is, usually for `Name-sg`, `name-sg`, `Name-pl`, `name-pl`, but only for the capitalized forms if the language was declared with `allcaps` option, and names for each declension case, if the language was declared with `declension`; iii) genders for each reference type, if the language was declared with `gender`. The language file may include some other things, like some type specific settings for separators or rebounds, and also some abbreviated name forms. In the case of abbreviated name forms, it is usual and desirable to provide some, but they should be used sparingly, only for cases where the abbreviation is a common and well established tradition for the language. The reason is that `abbrev=true` is quite a common use case, and it is easier to provide an occasional wanted abbreviated form, if the language file didn't include it, than it is to disable several unwanted ones, if the language file includes too many of them. What should be aimed at is to provide a good default abbreviations set. Unusual or disputable abbreviations should be avoided. In particular, there is no need at all to provide the same set of abbreviations for each language. It is not because English has them for a given type that some other language has to have them, and it is not because English lacks them for another type, that other languages shouldn't have them. Still, with regard to abbreviated forms, it is better to be conservative than opinionated.

babel names: As is known, `babel` defines a set of captions for different document objects for each supported language. In some cases, they intersect with the objects referred to with cross-references, in which case consistency with `babel` should be maintained as much as possible. This is specially the case for prominent and traditional objects, such as `\chaptername`, `\figurename`, `\tablename`, `\pagename`, `\partname`, and `\appendixname`. This is not set in stone, but there should be good reason to diverge from it. In particular, if a certain term is contentious in a given language, `babel`'s default should be preferred. For example, "table" vs. "tableau" in French, or "cuadro" vs. "tabla" in Spanish.

Input encoding of language files: When `zref-clever` was released, the L^AT_EX kernel already used UTF-8 as default input encoding. Indeed, `zref-clever` requires a kernel even newer than the one where the default input encoding was changed. That given, UTF-8 input encoding was made a requirement of the package, and hence the language files should be in UTF-8, since it makes them easier to read and maintain than LICR.

Precedence rule for options in the language files: Any option given twice or more times has to have some precedence rule. Normally, the language files should not contain options in duplicity, but they may happen when setting some “group” `refbounds` options, in which case precedence rules become relevant. For user facing options (those set with `\zcLanguageSetup`), the option is always set, regardless of its previous state. Which means that the last value takes precedence. For the language files, we have to load them at `begindocument` (or later), since that’s the point where we know from `babel` or `polyglossia` the `\language`name. But we also don’t want to override any options the user has actively set in the preamble. So the language files only set the values if they were not previously set. In other words, for them the precedence order is inverted, the first value takes precedence.

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

10.2 English

English language file has been initially provided by the author.

```

5665 <*package>
5666 \zcDeclareLanguage { english }
5667 \zcDeclareLanguageAlias { american } { english }
5668 \zcDeclareLanguageAlias { australian } { english }
5669 \zcDeclareLanguageAlias { british } { english }
5670 \zcDeclareLanguageAlias { canadian } { english }
5671 \zcDeclareLanguageAlias { newzealand } { english }
5672 \zcDeclareLanguageAlias { UKenglish } { english }
5673 \zcDeclareLanguageAlias { USenglish } { english }
5674 </package>

5675 <*lang-english>

5676 namesep = {\nobreakspace} ,
5677 pairsep = {\and\nobreakspace} ,
5678 listsep = {,~} ,
5679 lastsep = {\and\nobreakspace} ,
5680 tpairsep = {\and\nobreakspace} ,
5681 tlistsep = {,~} ,
5682 tlastsep = {\and\nobreakspace} ,
5683 notesep = {~} ,
5684 rangesep = {\to\nobreakspace} ,
5685
5686 type = book ,
5687 Name-sg = Book ,
5688 name-sg = book ,
5689 Name-pl = Books ,
5690 name-pl = books ,
5691

```

```

5692 type = part ,
5693     Name-sg = Part ,
5694     name-sg = part ,
5695     Name-pl = Parts ,
5696     name-pl = parts ,
5697
5698 type = chapter ,
5699     Name-sg = Chapter ,
5700     name-sg = chapter ,
5701     Name-pl = Chapters ,
5702     name-pl = chapters ,
5703
5704 type = section ,
5705     Name-sg = Section ,
5706     name-sg = section ,
5707     Name-pl = Sections ,
5708     name-pl = sections ,
5709
5710 type = paragraph ,
5711     Name-sg = Paragraph ,
5712     name-sg = paragraph ,
5713     Name-pl = Paragraphs ,
5714     name-pl = paragraphs ,
5715     Name-sg-ab = Par. ,
5716     name-sg-ab = par. ,
5717     Name-pl-ab = Par. ,
5718     name-pl-ab = par. ,
5719
5720 type = appendix ,
5721     Name-sg = Appendix ,
5722     name-sg = appendix ,
5723     Name-pl = Appendices ,
5724     name-pl = appendices ,
5725
5726 type = page ,
5727     Name-sg = Page ,
5728     name-sg = page ,
5729     Name-pl = Pages ,
5730     name-pl = pages ,
5731     rangesep = {\textendash} ,
5732     rangetopair = false ,
5733
5734 type = line ,
5735     Name-sg = Line ,
5736     name-sg = line ,
5737     Name-pl = Lines ,
5738     name-pl = lines ,
5739
5740 type = figure ,
5741     Name-sg = Figure ,
5742     name-sg = figure ,
5743     Name-pl = Figures ,
5744     name-pl = figures ,
5745     Name-sg-ab = Fig. ,

```

```

5746     name-sg-ab = fig. ,
5747     Name-pl-ab = Figs. ,
5748     name-pl-ab = figs. ,
5749
5750 type = table ,
5751     Name-sg = Table ,
5752     name-sg = table ,
5753     Name-pl = Tables ,
5754     name-pl = tables ,
5755
5756 type = item ,
5757     Name-sg = Item ,
5758     name-sg = item ,
5759     Name-pl = Items ,
5760     name-pl = items ,
5761
5762 type = footnote ,
5763     Name-sg = Footnote ,
5764     name-sg = footnote ,
5765     Name-pl = Footnotes ,
5766     name-pl = footnotes ,
5767
5768 type = endnote ,
5769     Name-sg = Note ,
5770     name-sg = note ,
5771     Name-pl = Notes ,
5772     name-pl = notes ,
5773
5774 type = note ,
5775     Name-sg = Note ,
5776     name-sg = note ,
5777     Name-pl = Notes ,
5778     name-pl = notes ,
5779
5780 type = equation ,
5781     Name-sg = Equation ,
5782     name-sg = equation ,
5783     Name-pl = Equations ,
5784     name-pl = equations ,
5785     Name-sg-ab = Eq. ,
5786     name-sg-ab = eq. ,
5787     Name-pl-ab = Eqs. ,
5788     name-pl-ab = eqs. ,
5789     refbounds-first-sg = {(,),(,)},
5790     refbounds = {(,,)} ,
5791
5792 type = theorem ,
5793     Name-sg = Theorem ,
5794     name-sg = theorem ,
5795     Name-pl = Theorems ,
5796     name-pl = theorems ,
5797
5798 type = lemma ,
5799     Name-sg = Lemma ,

```

```

5800     name-sg = lemma ,
5801     Name-pl = Lemmas ,
5802     name-pl = lemmas ,
5803
5804 type = corollary ,
5805     Name-sg = Corollary ,
5806     name-sg = corollary ,
5807     Name-pl = Corollaries ,
5808     name-pl = corollaries ,
5809
5810 type = proposition ,
5811     Name-sg = Proposition ,
5812     name-sg = proposition ,
5813     Name-pl = Propositions ,
5814     name-pl = propositions ,
5815
5816 type = definition ,
5817     Name-sg = Definition ,
5818     name-sg = definition ,
5819     Name-pl = Definitions ,
5820     name-pl = definitions ,
5821
5822 type = proof ,
5823     Name-sg = Proof ,
5824     name-sg = proof ,
5825     Name-pl = Proofs ,
5826     name-pl = proofs ,
5827
5828 type = result ,
5829     Name-sg = Result ,
5830     name-sg = result ,
5831     Name-pl = Results ,
5832     name-pl = results ,
5833
5834 type = remark ,
5835     Name-sg = Remark ,
5836     name-sg = remark ,
5837     Name-pl = Remarks ,
5838     name-pl = remarks ,
5839
5840 type = example ,
5841     Name-sg = Example ,
5842     name-sg = example ,
5843     Name-pl = Examples ,
5844     name-pl = examples ,
5845
5846 type = algorithm ,
5847     Name-sg = Algorithm ,
5848     name-sg = algorithm ,
5849     Name-pl = Algorithms ,
5850     name-pl = algorithms ,
5851
5852 type = listing ,
5853     Name-sg = Listing ,

```

```

5854 name-sg = listing ,
5855 Name-pl = Listings ,
5856 name-pl = listings ,
5857
5858 type = exercise ,
5859 Name-sg = Exercise ,
5860 name-sg = exercise ,
5861 Name-pl = Exercises ,
5862 name-pl = exercises ,
5863
5864 type = solution ,
5865 Name-sg = Solution ,
5866 name-sg = solution ,
5867 Name-pl = Solutions ,
5868 name-pl = solutions ,
5869 </lang-english>

```

10.3 German

German language file has been initially provided by the author.

`babel-german` also has `.ldfs` for `germanb` and `ngermanb`, but they are deprecated as options and, if used, they fall back respectively to `german` and `ngerman`.

```

5870 <*package>
5871 \zcDeclareLanguage
5872 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5873 { german }
5874 \zcDeclareLanguageAlias { ngerman } { german }
5875 \zcDeclareLanguageAlias { austrian } { german }
5876 \zcDeclareLanguageAlias { naustrian } { german }
5877 \zcDeclareLanguageAlias { swissgerman } { german }
5878 \zcDeclareLanguageAlias { nswissgerman } { german }
5879 </package>
5880 <*lang-german>
5881 namesep = {\nobreakspace} ,
5882 pairsep = {\und\nobreakspace} ,
5883 listsep = {,~} ,
5884 lastsep = {\und\nobreakspace} ,
5885 tpairsep = {\und\nobreakspace} ,
5886 tlistsep = {,~} ,
5887 tlastsep = {\und\nobreakspace} ,
5888 notesep = {~} ,
5889 rangesep = {\bis\nobreakspace} ,
5890
5891 type = book ,
5892 gender = n ,
5893 case = N ,
5894 Name-sg = Buch ,
5895 Name-pl = Bücher ,
5896 case = A ,
5897 Name-sg = Buch ,
5898 Name-pl = Bücher ,
5899 case = D ,

```

```

5900     Name-sg = Buch ,
5901     Name-pl = Büchern ,
5902     case = G ,
5903     Name-sg = Buches ,
5904     Name-pl = Bücher ,
5905
5906 type = part ,
5907     gender = m ,
5908     case = N ,
5909     Name-sg = Teil ,
5910     Name-pl = Teile ,
5911     case = A ,
5912     Name-sg = Teil ,
5913     Name-pl = Teile ,
5914     case = D ,
5915     Name-sg = Teil ,
5916     Name-pl = Teilen ,
5917     case = G ,
5918     Name-sg = Teiles ,
5919     Name-pl = Teile ,
5920
5921 type = chapter ,
5922     gender = n ,
5923     case = N ,
5924     Name-sg = Kapitel ,
5925     Name-pl = Kapitel ,
5926     case = A ,
5927     Name-sg = Kapitel ,
5928     Name-pl = Kapitel ,
5929     case = D ,
5930     Name-sg = Kapitel ,
5931     Name-pl = Kapiteln ,
5932     case = G ,
5933     Name-sg = Kapitels ,
5934     Name-pl = Kapitel ,
5935
5936 type = section ,
5937     gender = m ,
5938     case = N ,
5939     Name-sg = Abschnitt ,
5940     Name-pl = Abschnitte ,
5941     case = A ,
5942     Name-sg = Abschnitt ,
5943     Name-pl = Abschnitte ,
5944     case = D ,
5945     Name-sg = Abschnitt ,
5946     Name-pl = Abschnitten ,
5947     case = G ,
5948     Name-sg = Abschnitts ,
5949     Name-pl = Abschnitte ,
5950
5951 type = paragraph ,
5952     gender = m ,
5953     case = N ,

```



```

5954     Name-sg = Absatz ,
5955     Name-pl = Absätze ,
5956     case = A ,
5957     Name-sg = Absatz ,
5958     Name-pl = Absätze ,
5959     case = D ,
5960     Name-sg = Absatz ,
5961     Name-pl = Absätzen ,
5962     case = G ,
5963     Name-sg = Absatzes ,
5964     Name-pl = Absätze ,
5965
5966 type = appendix ,
5967     gender = m ,
5968     case = N ,
5969     Name-sg = Anhang ,
5970     Name-pl = Anhänge ,
5971     case = A ,
5972     Name-sg = Anhang ,
5973     Name-pl = Anhänge ,
5974     case = D ,
5975     Name-sg = Anhang ,
5976     Name-pl = Anhängen ,
5977     case = G ,
5978     Name-sg = Anhangs ,
5979     Name-pl = Anhänge ,
5980
5981 type = page ,
5982     gender = f ,
5983     case = N ,
5984     Name-sg = Seite ,
5985     Name-pl = Seiten ,
5986     case = A ,
5987     Name-sg = Seite ,
5988     Name-pl = Seiten ,
5989     case = D ,
5990     Name-sg = Seite ,
5991     Name-pl = Seiten ,
5992     case = G ,
5993     Name-sg = Seite ,
5994     Name-pl = Seiten ,
5995     rangesep = {\textendash} ,
5996     rangetopair = false ,
5997
5998 type = line ,
5999     gender = f ,
6000     case = N ,
6001     Name-sg = Zeile ,
6002     Name-pl = Zeilen ,
6003     case = A ,
6004     Name-sg = Zeile ,
6005     Name-pl = Zeilen ,
6006     case = D ,
6007     Name-sg = Zeile ,

```

```

6008     Name-pl = Zeilen ,
6009     case = G ,
6010     Name-sg = Zeile ,
6011     Name-pl = Zeilen ,
6012
6013 type = figure ,
6014     gender = f ,
6015     case = N ,
6016     Name-sg = Abbildung ,
6017     Name-pl = Abbildungen ,
6018     Name-sg-ab = Abb. ,
6019     Name-pl-ab = Abb. ,
6020     case = A ,
6021     Name-sg = Abbildung ,
6022     Name-pl = Abbildungen ,
6023     Name-sg-ab = Abb. ,
6024     Name-pl-ab = Abb. ,
6025     case = D ,
6026     Name-sg = Abbildung ,
6027     Name-pl = Abbildungen ,
6028     Name-sg-ab = Abb. ,
6029     Name-pl-ab = Abb. ,
6030     case = G ,
6031     Name-sg = Abbildung ,
6032     Name-pl = Abbildungen ,
6033     Name-sg-ab = Abb. ,
6034     Name-pl-ab = Abb. ,
6035
6036 type = table ,
6037     gender = f ,
6038     case = N ,
6039     Name-sg = Tabelle ,
6040     Name-pl = Tabellen ,
6041     case = A ,
6042     Name-sg = Tabelle ,
6043     Name-pl = Tabellen ,
6044     case = D ,
6045     Name-sg = Tabelle ,
6046     Name-pl = Tabellen ,
6047     case = G ,
6048     Name-sg = Tabelle ,
6049     Name-pl = Tabellen ,
6050
6051 type = item ,
6052     gender = m ,
6053     case = N ,
6054     Name-sg = Punkt ,
6055     Name-pl = Punkte ,
6056     case = A ,
6057     Name-sg = Punkt ,
6058     Name-pl = Punkte ,
6059     case = D ,
6060     Name-sg = Punkt ,
6061     Name-pl = Punkten ,

```

```

6062     case = G ,
6063         Name-sg = Punktes ,
6064         Name-pl = Punkte ,
6065
6066 type = footnote ,
6067     gender = f ,
6068     case = N ,
6069         Name-sg = Fußnote ,
6070         Name-pl = Fußnoten ,
6071     case = A ,
6072         Name-sg = Fußnote ,
6073         Name-pl = Fußnoten ,
6074     case = D ,
6075         Name-sg = Fußnote ,
6076         Name-pl = Fußnoten ,
6077     case = G ,
6078         Name-sg = Fußnote ,
6079         Name-pl = Fußnoten ,
6080
6081 type = endnote ,
6082     gender = f ,
6083     case = N ,
6084         Name-sg = Endnote ,
6085         Name-pl = Endnoten ,
6086     case = A ,
6087         Name-sg = Endnote ,
6088         Name-pl = Endnoten ,
6089     case = D ,
6090         Name-sg = Endnote ,
6091         Name-pl = Endnoten ,
6092     case = G ,
6093         Name-sg = Endnote ,
6094         Name-pl = Endnoten ,
6095
6096 type = note ,
6097     gender = f ,
6098     case = N ,
6099         Name-sg = Anmerkung ,
6100         Name-pl = Anmerkungen ,
6101     case = A ,
6102         Name-sg = Anmerkung ,
6103         Name-pl = Anmerkungen ,
6104     case = D ,
6105         Name-sg = Anmerkung ,
6106         Name-pl = Anmerkungen ,
6107     case = G ,
6108         Name-sg = Anmerkung ,
6109         Name-pl = Anmerkungen ,
6110
6111 type = equation ,
6112     gender = f ,
6113     case = N ,
6114         Name-sg = Gleichung ,
6115         Name-pl = Gleichungen ,

```

```

6116 case = A ,
6117     Name-sg = Gleichung ,
6118     Name-pl = Gleichungen ,
6119 case = D ,
6120     Name-sg = Gleichung ,
6121     Name-pl = Gleichungen ,
6122 case = G ,
6123     Name-sg = Gleichung ,
6124     Name-pl = Gleichungen ,
6125 refbounds-first-sg = {(,),(,)},
6126 refbounds = {(,),(,)} ,
6127
6128 type = theorem ,
6129     gender = n ,
6130     case = N ,
6131         Name-sg = Theorem ,
6132         Name-pl = Theoreme ,
6133     case = A ,
6134         Name-sg = Theorem ,
6135         Name-pl = Theoreme ,
6136     case = D ,
6137         Name-sg = Theorem ,
6138         Name-pl = Theoremen ,
6139     case = G ,
6140         Name-sg = Theorems ,
6141         Name-pl = Theoreme ,
6142
6143 type = lemma ,
6144     gender = n ,
6145     case = N ,
6146         Name-sg = Lemma ,
6147         Name-pl = Lemmata ,
6148     case = A ,
6149         Name-sg = Lemma ,
6150         Name-pl = Lemmata ,
6151     case = D ,
6152         Name-sg = Lemma ,
6153         Name-pl = Lemmata ,
6154     case = G ,
6155         Name-sg = Lemmas ,
6156         Name-pl = Lemmata ,
6157
6158 type = corollary ,
6159     gender = n ,
6160     case = N ,
6161         Name-sg = Korollar ,
6162         Name-pl = Korollare ,
6163     case = A ,
6164         Name-sg = Korollar ,
6165         Name-pl = Korollare ,
6166     case = D ,
6167         Name-sg = Korollar ,
6168         Name-pl = Korollaren ,
6169     case = G ,

```

```

6170     Name-sg = Korollars ,
6171     Name-pl = Korollare ,
6172
6173 type = proposition ,
6174     gender = m ,
6175     case = N ,
6176     Name-sg = Satz ,
6177     Name-pl = Sätze ,
6178     case = A ,
6179     Name-sg = Satz ,
6180     Name-pl = Sätze ,
6181     case = D ,
6182     Name-sg = Satz ,
6183     Name-pl = Sätzen ,
6184     case = G ,
6185     Name-sg = Satzes ,
6186     Name-pl = Sätze ,
6187
6188 type = definition ,
6189     gender = f ,
6190     case = N ,
6191     Name-sg = Definition ,
6192     Name-pl = Definitionen ,
6193     case = A ,
6194     Name-sg = Definition ,
6195     Name-pl = Definitionen ,
6196     case = D ,
6197     Name-sg = Definition ,
6198     Name-pl = Definitionen ,
6199     case = G ,
6200     Name-sg = Definition ,
6201     Name-pl = Definitionen ,
6202
6203 type = proof ,
6204     gender = m ,
6205     case = N ,
6206     Name-sg = Beweis ,
6207     Name-pl = Beweise ,
6208     case = A ,
6209     Name-sg = Beweis ,
6210     Name-pl = Beweise ,
6211     case = D ,
6212     Name-sg = Beweis ,
6213     Name-pl = Beweisen ,
6214     case = G ,
6215     Name-sg = Beweises ,
6216     Name-pl = Beweise ,
6217
6218 type = result ,
6219     gender = n ,
6220     case = N ,
6221     Name-sg = Ergebnis ,
6222     Name-pl = Ergebnisse ,
6223     case = A ,

```

```

6224     Name-sg = Ergebnis ,
6225     Name-pl = Ergebnisse ,
6226     case = D ,
6227     Name-sg = Ergebnis ,
6228     Name-pl = Ergebnissen ,
6229     case = G ,
6230     Name-sg = Ergebnisses ,
6231     Name-pl = Ergebnisse ,
6232
6233     type = remark ,
6234     gender = f ,
6235     case = N ,
6236     Name-sg = Bemerkung ,
6237     Name-pl = Bemerkungen ,
6238     case = A ,
6239     Name-sg = Bemerkung ,
6240     Name-pl = Bemerkungen ,
6241     case = D ,
6242     Name-sg = Bemerkung ,
6243     Name-pl = Bemerkungen ,
6244     case = G ,
6245     Name-sg = Bemerkung ,
6246     Name-pl = Bemerkungen ,
6247
6248     type = example ,
6249     gender = n ,
6250     case = N ,
6251     Name-sg = Beispiel ,
6252     Name-pl = Beispiele ,
6253     case = A ,
6254     Name-sg = Beispiel ,
6255     Name-pl = Beispiele ,
6256     case = D ,
6257     Name-sg = Beispiel ,
6258     Name-pl = Beispielen ,
6259     case = G ,
6260     Name-sg = Beispiels ,
6261     Name-pl = Beispiele ,
6262
6263     type = algorithm ,
6264     gender = m ,
6265     case = N ,
6266     Name-sg = Algorithmus ,
6267     Name-pl = Algorithmen ,
6268     case = A ,
6269     Name-sg = Algorithmus ,
6270     Name-pl = Algorithmen ,
6271     case = D ,
6272     Name-sg = Algorithmus ,
6273     Name-pl = Algorithmen ,
6274     case = G ,
6275     Name-sg = Algorithmus ,
6276     Name-pl = Algorithmen ,
6277

```

```

6278 type = listing ,
6279 gender = n ,
6280 case = N ,
6281   Name-sg = Listing ,
6282   Name-pl = Listings ,
6283 case = A ,
6284   Name-sg = Listing ,
6285   Name-pl = Listings ,
6286 case = D ,
6287   Name-sg = Listing ,
6288   Name-pl = Listings ,
6289 case = G ,
6290   Name-sg = Listings ,
6291   Name-pl = Listings ,
6292
6293 type = exercise ,
6294 gender = f ,
6295 case = N ,
6296   Name-sg = Übungsaufgabe ,
6297   Name-pl = Übungsaufgaben ,
6298 case = A ,
6299   Name-sg = Übungsaufgabe ,
6300   Name-pl = Übungsaufgaben ,
6301 case = D ,
6302   Name-sg = Übungsaufgabe ,
6303   Name-pl = Übungsaufgaben ,
6304 case = G ,
6305   Name-sg = Übungsaufgabe ,
6306   Name-pl = Übungsaufgaben ,
6307
6308 type = solution ,
6309 gender = f ,
6310 case = N ,
6311   Name-sg = Lösung ,
6312   Name-pl = Lösungen ,
6313 case = A ,
6314   Name-sg = Lösung ,
6315   Name-pl = Lösungen ,
6316 case = D ,
6317   Name-sg = Lösung ,
6318   Name-pl = Lösungen ,
6319 case = G ,
6320   Name-sg = Lösung ,
6321   Name-pl = Lösungen ,
6322 </lang-german>

```

10.4 French

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

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

```

6323 <*package>
6324 \zcDeclareLanguage [ gender = { f , m } ] { french }
6325 \zcDeclareLanguageAlias { acadian } { french }
6326 </package>
6327 <*lang-french>
6328 namesep = {\nobreakspace} ,
6329 pairsep = {\~et\nobreakspace} ,
6330 listsep = { , ~ } ,
6331 lastsep = {\~et\nobreakspace} ,
6332 tpairsep = {\~et\nobreakspace} ,
6333 tlistsep = { , ~ } ,
6334 tlastsep = {\~et\nobreakspace} ,
6335 notesep = { ~ } ,
6336 rangesep = {\~à\nobreakspace} ,
6337
6338 type = book ,
6339   gender = m ,
6340   Name-sg = Livre ,
6341   name-sg = livre ,
6342   Name-pl = Livres ,
6343   name-pl = livres ,
6344
6345 type = part ,
6346   gender = f ,
6347   Name-sg = Partie ,
6348   name-sg = partie ,
6349   Name-pl = Parties ,
6350   name-pl = parties ,
6351
6352 type = chapter ,
6353   gender = m ,
6354   Name-sg = Chapitre ,
6355   name-sg = chapitre ,
6356   Name-pl = Chapitres ,
6357   name-pl = chapitres ,
6358
6359 type = section ,
6360   gender = f ,
6361   Name-sg = Section ,
6362   name-sg = section ,
6363   Name-pl = Sections ,
6364   name-pl = sections ,
6365
6366 type = paragraph ,
6367   gender = m ,
6368   Name-sg = Paragraphe ,
6369   name-sg = paragraphe ,
6370   Name-pl = Paragraphes ,
6371   name-pl = paragraphes ,
6372
6373 type = appendix ,

```



```

6374     gender = f ,
6375     Name-sg = Annexe ,
6376     name-sg = annexe ,
6377     Name-pl = Annexes ,
6378     name-pl = annexes ,
6379
6380     type = page ,
6381     gender = f ,
6382     Name-sg = Page ,
6383     name-sg = page ,
6384     Name-pl = Pages ,
6385     name-pl = pages ,
6386     rangesep = {-} ,
6387     rangetopair = false ,
6388
6389     type = line ,
6390     gender = f ,
6391     Name-sg = Ligne ,
6392     name-sg = ligne ,
6393     Name-pl = Lignes ,
6394     name-pl = lignes ,
6395
6396     type = figure ,
6397     gender = f ,
6398     Name-sg = Figure ,
6399     name-sg = figure ,
6400     Name-pl = Figures ,
6401     name-pl = figures ,
6402
6403     type = table ,
6404     gender = f ,
6405     Name-sg = Table ,
6406     name-sg = table ,
6407     Name-pl = Tables ,
6408     name-pl = tables ,
6409
6410     type = item ,
6411     gender = m ,
6412     Name-sg = Point ,
6413     name-sg = point ,
6414     Name-pl = Points ,
6415     name-pl = points ,
6416
6417     type = footnote ,
6418     gender = f ,
6419     Name-sg = Note ,
6420     name-sg = note ,
6421     Name-pl = Notes ,
6422     name-pl = notes ,
6423
6424     type = endnote ,
6425     gender = f ,
6426     Name-sg = Note ,
6427     name-sg = note ,

```

```

6428     Name-pl = Notes ,
6429     name-pl = notes ,
6430
6431     type = note ,
6432     gender = f ,
6433     Name-sg = Note ,
6434     name-sg = note ,
6435     Name-pl = Notes ,
6436     name-pl = notes ,
6437
6438     type = equation ,
6439     gender = f ,
6440     Name-sg = Équation ,
6441     name-sg = équation ,
6442     Name-pl = Équations ,
6443     name-pl = équations ,
6444     refbounds-first-sg = {,(,),} ,
6445     refbounds = {(,,)} ,
6446
6447     type = theorem ,
6448     gender = m ,
6449     Name-sg = Théorème ,
6450     name-sg = théorème ,
6451     Name-pl = Théorèmes ,
6452     name-pl = théorèmes ,
6453
6454     type = lemma ,
6455     gender = m ,
6456     Name-sg = Lemme ,
6457     name-sg = lemme ,
6458     Name-pl = Lemmes ,
6459     name-pl = lemmes ,
6460
6461     type = corollary ,
6462     gender = m ,
6463     Name-sg = Corollaire ,
6464     name-sg = corollaire ,
6465     Name-pl = Corollaires ,
6466     name-pl = corollaires ,
6467
6468     type = proposition ,
6469     gender = f ,
6470     Name-sg = Proposition ,
6471     name-sg = proposition ,
6472     Name-pl = Propositions ,
6473     name-pl = propositions ,
6474
6475     type = definition ,
6476     gender = f ,
6477     Name-sg = Définition ,
6478     name-sg = définition ,
6479     Name-pl = Définitions ,
6480     name-pl = définitions ,
6481

```

```

6482 type = proof ,
6483     gender = f ,
6484     Name-sg = Démonstration ,
6485     name-sg = démonstration ,
6486     Name-pl = Démonstrations ,
6487     name-pl = démonstrations ,
6488
6489 type = result ,
6490     gender = m ,
6491     Name-sg = Résultat ,
6492     name-sg = résultat ,
6493     Name-pl = Résultats ,
6494     name-pl = résultats ,
6495
6496 type = remark ,
6497     gender = f ,
6498     Name-sg = Remarque ,
6499     name-sg = remarque ,
6500     Name-pl = Remarques ,
6501     name-pl = remarques ,
6502
6503 type = example ,
6504     gender = m ,
6505     Name-sg = Exemple ,
6506     name-sg = exemple ,
6507     Name-pl = Exemples ,
6508     name-pl = exemples ,
6509
6510 type = algorithm ,
6511     gender = m ,
6512     Name-sg = Algorithme ,
6513     name-sg = algorithme ,
6514     Name-pl = Algorithmes ,
6515     name-pl = algorithmes ,
6516
6517 type = listing ,
6518     gender = m ,
6519     Name-sg = Listing ,
6520     name-sg = listing ,
6521     Name-pl = Listings ,
6522     name-pl = listings ,
6523
6524 type = exercise ,
6525     gender = m ,
6526     Name-sg = Exercice ,
6527     name-sg = exercice ,
6528     Name-pl = Exercices ,
6529     name-pl = exercices ,
6530
6531 type = solution ,
6532     gender = f ,
6533     Name-sg = Solution ,
6534     name-sg = solution ,
6535     Name-pl = Solutions ,

```

```

6536   name-pl = solutions ,
6537 </lang-french>

```

10.5 Portuguese

Portuguese language file provided by the author, who's a native speaker of (Brazilian) Portuguese. I do expect this to be sufficiently general, but if Portuguese speakers from other places feel the need for a Portuguese variant, please let me know.

```

6538 <*package>
6539 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6540 \zcDeclareLanguageAlias { brazilian } { portuguese }
6541 \zcDeclareLanguageAlias { brazil   } { portuguese }
6542 \zcDeclareLanguageAlias { portuges  } { portuguese }
6543 </package>
6544 <*lang-portuguese>
6545 namesep = {\nobreakspace} ,
6546 pairsep  = {\~e\nobreakspace} ,
6547 listsep  = { ,~ } ,
6548 lastsep  = {\~e\nobreakspace} ,
6549 tpairsep = {\~e\nobreakspace} ,
6550 tlistsep = { ,~ } ,
6551 tlastsep = {\~e\nobreakspace} ,
6552 notesep  = {~} ,
6553 rangesep = {\~a\nobreakspace} ,
6554
6555 type = book ,
6556   gender = m ,
6557   Name-sg = Livro ,
6558   name-sg  = livro ,
6559   Name-pl  = Livros ,
6560   name-pl  = livros ,
6561
6562 type = part ,
6563   gender = f ,
6564   Name-sg = Parte ,
6565   name-sg = parte ,
6566   Name-pl = Partes ,
6567   name-pl = partes ,
6568
6569 type = chapter ,
6570   gender = m ,
6571   Name-sg = Capítulo ,
6572   name-sg = capítulo ,
6573   Name-pl = Capítulos ,
6574   name-pl = capítulos ,
6575
6576 type = section ,
6577   gender = f ,
6578   Name-sg = Seção ,
6579   name-sg = seção ,
6580   Name-pl = Seções ,
6581   name-pl = seções ,

```

```

6582
6583 type = paragraph ,
6584     gender = m ,
6585     Name-sg = Parágrafo ,
6586     name-sg = parágrafo ,
6587     Name-pl = Parágrafos ,
6588     name-pl = parágrafos ,
6589     Name-sg-ab = Par. ,
6590     name-sg-ab = par. ,
6591     Name-pl-ab = Par. ,
6592     name-pl-ab = par. ,
6593
6594 type = appendix ,
6595     gender = m ,
6596     Name-sg = Apêndice ,
6597     name-sg = apêndice ,
6598     Name-pl = Apêndices ,
6599     name-pl = apêndices ,
6600
6601 type = page ,
6602     gender = f ,
6603     Name-sg = Página ,
6604     name-sg = página ,
6605     Name-pl = Páginas ,
6606     name-pl = páginas ,
6607     rangesep = {\textendash} ,
6608     rangetopair = false ,
6609
6610 type = line ,
6611     gender = f ,
6612     Name-sg = Linha ,
6613     name-sg = linha ,
6614     Name-pl = Linhas ,
6615     name-pl = linhas ,
6616
6617 type = figure ,
6618     gender = f ,
6619     Name-sg = Figura ,
6620     name-sg = figura ,
6621     Name-pl = Figuras ,
6622     name-pl = figuras ,
6623     Name-sg-ab = Fig. ,
6624     name-sg-ab = fig. ,
6625     Name-pl-ab = Figs. ,
6626     name-pl-ab = figs. ,
6627
6628 type = table ,
6629     gender = f ,
6630     Name-sg = Tabela ,
6631     name-sg = tabela ,
6632     Name-pl = Tabelas ,
6633     name-pl = tabelas ,
6634
6635 type = item ,

```

```

6636     gender = m ,
6637     Name-sg = Item ,
6638     name-sg = item ,
6639     Name-pl = Itens ,
6640     name-pl = itens ,
6641
6642     type = footnote ,
6643     gender = f ,
6644     Name-sg = Nota ,
6645     name-sg = nota ,
6646     Name-pl = Notas ,
6647     name-pl = notas ,
6648
6649     type = endnote ,
6650     gender = f ,
6651     Name-sg = Nota ,
6652     name-sg = nota ,
6653     Name-pl = Notas ,
6654     name-pl = notas ,
6655
6656     type = note ,
6657     gender = f ,
6658     Name-sg = Nota ,
6659     name-sg = nota ,
6660     Name-pl = Notas ,
6661     name-pl = notas ,
6662
6663     type = equation ,
6664     gender = f ,
6665     Name-sg = Equação ,
6666     name-sg = equação ,
6667     Name-pl = Equações ,
6668     name-pl = equações ,
6669     Name-sg-ab = Eq. ,
6670     name-sg-ab = eq. ,
6671     Name-pl-ab = Eqs. ,
6672     name-pl-ab = eqs. ,
6673     refbounds-first-sg = {,(,),} ,
6674     refbounds = {(,,)} ,
6675
6676     type = theorem ,
6677     gender = m ,
6678     Name-sg = Teorema ,
6679     name-sg = teorema ,
6680     Name-pl = Teoremas ,
6681     name-pl = teoremas ,
6682
6683     type = lemma ,
6684     gender = m ,
6685     Name-sg = Lema ,
6686     name-sg = lema ,
6687     Name-pl = Lemas ,
6688     name-pl = lemas ,
6689

```

```

6690 type = corollary ,
6691     gender = m ,
6692     Name-sg = Corolário ,
6693     name-sg = corolário ,
6694     Name-pl = Corolários ,
6695     name-pl = corolários ,
6696
6697 type = proposition ,
6698     gender = f ,
6699     Name-sg = Proposição ,
6700     name-sg = proposição ,
6701     Name-pl = Proposições ,
6702     name-pl = proposições ,
6703
6704 type = definition ,
6705     gender = f ,
6706     Name-sg = Definição ,
6707     name-sg = definição ,
6708     Name-pl = Definições ,
6709     name-pl = definições ,
6710
6711 type = proof ,
6712     gender = f ,
6713     Name-sg = Demonstração ,
6714     name-sg = demonstração ,
6715     Name-pl = Demonstrações ,
6716     name-pl = demonstrações ,
6717
6718 type = result ,
6719     gender = m ,
6720     Name-sg = Resultado ,
6721     name-sg = resultado ,
6722     Name-pl = Resultados ,
6723     name-pl = resultados ,
6724
6725 type = remark ,
6726     gender = f ,
6727     Name-sg = Observação ,
6728     name-sg = observação ,
6729     Name-pl = Observações ,
6730     name-pl = observações ,
6731
6732 type = example ,
6733     gender = m ,
6734     Name-sg = Exemplo ,
6735     name-sg = exemplo ,
6736     Name-pl = Exemplos ,
6737     name-pl = exemplos ,
6738
6739 type = algorithm ,
6740     gender = m ,
6741     Name-sg = Algoritmo ,
6742     name-sg = algoritmo ,
6743     Name-pl = Algoritmos ,

```

```

6744   name-pl = algoritmos ,
6745
6746   type = listing ,
6747   gender = f ,
6748   Name-sg = Listagem ,
6749   name-sg = listagem ,
6750   Name-pl = Listagens ,
6751   name-pl = listagens ,
6752
6753   type = exercise ,
6754   gender = m ,
6755   Name-sg = Exercício ,
6756   name-sg = exercício ,
6757   Name-pl = Exercícios ,
6758   name-pl = exercícios ,
6759
6760   type = solution ,
6761   gender = f ,
6762   Name-sg = Solução ,
6763   name-sg = solução ,
6764   Name-pl = Soluções ,
6765   name-pl = soluções ,
6766 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```

6767 <*package>
6768 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6769 </package>
6770 <*lang-spanish>
6771 namesep = {\nobreakspace} ,
6772 pairsep = {\~y\nobreakspace} ,
6773 listsep = {,~} ,
6774 lastsep = {\~y\nobreakspace} ,
6775 tpairsep = {\~y\nobreakspace} ,
6776 tlistsep = {,~} ,
6777 tlastsep = {\~y\nobreakspace} ,
6778 notesep = {\~} ,
6779 rangesep = {\~a\nobreakspace} ,
6780
6781 type = book ,
6782 gender = m ,
6783 Name-sg = Libro ,
6784 name-sg = libro ,
6785 Name-pl = Libros ,
6786 name-pl = libros ,
6787
6788 type = part ,
6789 gender = f ,
6790 Name-sg = Parte ,
6791 name-sg = parte ,

```



```

6792     Name-pl = Partes ,
6793     name-pl = partes ,
6794
6795     type = chapter ,
6796     gender = m ,
6797     Name-sg = Capítulo ,
6798     name-sg = capítulo ,
6799     Name-pl = Capítulos ,
6800     name-pl = capítulos ,
6801
6802     type = section ,
6803     gender = f ,
6804     Name-sg = Sección ,
6805     name-sg = sección ,
6806     Name-pl = Secciones ,
6807     name-pl = secciones ,
6808
6809     type = paragraph ,
6810     gender = m ,
6811     Name-sg = Párrafo ,
6812     name-sg = párrafo ,
6813     Name-pl = Párrafos ,
6814     name-pl = párrafos ,
6815
6816     type = appendix ,
6817     gender = m ,
6818     Name-sg = Apéndice ,
6819     name-sg = apéndice ,
6820     Name-pl = Apéndices ,
6821     name-pl = apéndices ,
6822
6823     type = page ,
6824     gender = f ,
6825     Name-sg = Página ,
6826     name-sg = página ,
6827     Name-pl = Páginas ,
6828     name-pl = páginas ,
6829     rangesep = {\textendash} ,
6830     rangetopair = false ,
6831
6832     type = line ,
6833     gender = f ,
6834     Name-sg = Línea ,
6835     name-sg = línea ,
6836     Name-pl = Líneas ,
6837     name-pl = líneas ,
6838
6839     type = figure ,
6840     gender = f ,
6841     Name-sg = Figura ,
6842     name-sg = figura ,
6843     Name-pl = Figuras ,
6844     name-pl = figuras ,
6845

```

```

6846 type = table ,
6847   gender = m ,
6848   Name-sg = Cuadro ,
6849   name-sg = cuadro ,
6850   Name-pl = Cuadros ,
6851   name-pl = cuadros ,
6852
6853 type = item ,
6854   gender = m ,
6855   Name-sg = Punto ,
6856   name-sg = punto ,
6857   Name-pl = Puntos ,
6858   name-pl = puntos ,
6859
6860 type = footnote ,
6861   gender = f ,
6862   Name-sg = Nota ,
6863   name-sg = nota ,
6864   Name-pl = Notas ,
6865   name-pl = notas ,
6866
6867 type = endnote ,
6868   gender = f ,
6869   Name-sg = Nota ,
6870   name-sg = nota ,
6871   Name-pl = Notas ,
6872   name-pl = notas ,
6873
6874 type = note ,
6875   gender = f ,
6876   Name-sg = Nota ,
6877   name-sg = nota ,
6878   Name-pl = Notas ,
6879   name-pl = notas ,
6880
6881 type = equation ,
6882   gender = f ,
6883   Name-sg = Ecuación ,
6884   name-sg = ecuación ,
6885   Name-pl = Ecuaciones ,
6886   name-pl = ecuaciones ,
6887   refbounds-first-sg = {(,),(,)} ,
6888   refbounds = {(,,)} ,
6889
6890 type = theorem ,
6891   gender = m ,
6892   Name-sg = Teorema ,
6893   name-sg = teorema ,
6894   Name-pl = Teoremas ,
6895   name-pl = teoremas ,
6896
6897 type = lemma ,
6898   gender = m ,
6899   Name-sg = Lema ,

```

```

6900     name-sg = lema ,
6901     Name-pl = Lemas ,
6902     name-pl = lemas ,
6903
6904 type = corollary ,
6905     gender = m ,
6906     Name-sg = Corolario ,
6907     name-sg = corolario ,
6908     Name-pl = Corolarios ,
6909     name-pl = corolarios ,
6910
6911 type = proposition ,
6912     gender = f ,
6913     Name-sg = Proposición ,
6914     name-sg = proposición ,
6915     Name-pl = Proposiciones ,
6916     name-pl = proposiciones ,
6917
6918 type = definition ,
6919     gender = f ,
6920     Name-sg = Definición ,
6921     name-sg = definición ,
6922     Name-pl = Definiciones ,
6923     name-pl = definiciones ,
6924
6925 type = proof ,
6926     gender = f ,
6927     Name-sg = Demostración ,
6928     name-sg = demostración ,
6929     Name-pl = Demostraciones ,
6930     name-pl = demostraciones ,
6931
6932 type = result ,
6933     gender = m ,
6934     Name-sg = Resultado ,
6935     name-sg = resultado ,
6936     Name-pl = Resultados ,
6937     name-pl = resultados ,
6938
6939 type = remark ,
6940     gender = f ,
6941     Name-sg = Observación ,
6942     name-sg = observación ,
6943     Name-pl = Observaciones ,
6944     name-pl = observaciones ,
6945
6946 type = example ,
6947     gender = m ,
6948     Name-sg = Ejemplo ,
6949     name-sg = ejemplo ,
6950     Name-pl = Ejemplos ,
6951     name-pl = ejemplos ,
6952
6953 type = algorithm ,

```

```

6954   gender = m ,
6955   Name-sg = Algoritmo ,
6956   name-sg = algoritmo ,
6957   Name-pl = Algoritmos ,
6958   name-pl = algoritmos ,
6959
6960   type = listing ,
6961   gender = m ,
6962   Name-sg = Listado ,
6963   name-sg = listado ,
6964   Name-pl = Listados ,
6965   name-pl = listados ,
6966
6967   type = exercise ,
6968   gender = m ,
6969   Name-sg = Ejercicio ,
6970   name-sg = ejercicio ,
6971   Name-pl = Ejercicios ,
6972   name-pl = ejercicios ,
6973
6974   type = solution ,
6975   gender = f ,
6976   Name-sg = Solución ,
6977   name-sg = solución ,
6978   Name-pl = Soluciones ,
6979   name-pl = soluciones ,
6980 </lang-spanish>

```

10.7 Dutch

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

```

6981 <*package>
6982 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
6983 </package>
6984 <*lang-dutch>
6985 namesep   = {\nobreakspace} ,
6986 pairsep   = {\~en\nobreakspace} ,
6987 listsep    = {,~} ,
6988 lastsep    = {\~en\nobreakspace} ,
6989 tpairsep   = {\~en\nobreakspace} ,
6990 tlistsep   = {,~} ,
6991 tlastsep   = {,~en\nobreakspace} ,
6992 notesep    = {~} ,
6993 rangesep   = {\~t/m\nobreakspace} ,
6994
6995 type = book ,
6996   gender = n ,
6997   Name-sg = Boek ,
6998   name-sg = boek ,
6999   Name-pl = Boeken ,
7000   name-pl = boeken ,

```

```

7001
7002 type = part ,
7003     gender = n ,
7004     Name-sg = Deel ,
7005     name-sg = deel ,
7006     Name-pl = Delen ,
7007     name-pl = delen ,
7008
7009 type = chapter ,
7010     gender = n ,
7011     Name-sg = Hoofdstuk ,
7012     name-sg = hoofdstuk ,
7013     Name-pl = Hoofdstukken ,
7014     name-pl = hoofdstukken ,
7015
7016 type = section ,
7017     gender = m ,
7018     Name-sg = Paragraaf ,
7019     name-sg = paragraaf ,
7020     Name-pl = Paragrafen ,
7021     name-pl = paragrafen ,
7022
7023 type = paragraph ,
7024     gender = f ,
7025     Name-sg = Alinea ,
7026     name-sg = alinea ,
7027     Name-pl = Alinea's ,
7028     name-pl = alinea's ,
7029

```

2022-12-27, ‘niluxv’: “bijlage” is chosen over “appendix” (plural “appendices”, gender: m, n) for consistency with babel/polyglossia. “bijlages” is also a valid plural; “bijlagen” is chosen for consistency with babel/polyglossia.

```

7030 type = appendix ,
7031     gender = { f , m } ,
7032     Name-sg = Blage ,
7033     name-sg = blage ,
7034     Name-pl = Blagen ,
7035     name-pl = blagen ,
7036
7037 type = page ,
7038     gender = { f , m } ,
7039     Name-sg = Pagina ,
7040     name-sg = pagina ,
7041     Name-pl = Pagina's ,
7042     name-pl = pagina's ,
7043     rangesep = {\textendash} ,
7044     rangetopair = false ,
7045
7046 type = line ,
7047     gender = m ,
7048     Name-sg = Regel ,
7049     name-sg = regel ,
7050     Name-pl = Regels ,

```

```

7051     name-pl = regels ,
7052
7053     type = figure ,
7054     gender = { n , f , m } ,
7055     Name-sg = Figuur ,
7056     name-sg = figuur ,
7057     Name-pl = Figuren ,
7058     name-pl = figuren ,
7059
7060     type = table ,
7061     gender = { f , m } ,
7062     Name-sg = Tabel ,
7063     name-sg = tabel ,
7064     Name-pl = Tabellen ,
7065     name-pl = tabellen ,
7066
7067     type = item ,
7068     gender = n ,
7069     Name-sg = Punt ,
7070     name-sg = punt ,
7071     Name-pl = Punten ,
7072     name-pl = punten ,
7073
7074     type = footnote ,
7075     gender = { f , m } ,
7076     Name-sg = Voetnoot ,
7077     name-sg = voetnoot ,
7078     Name-pl = Voetnoten ,
7079     name-pl = voetnoten ,
7080
7081     type = endnote ,
7082     gender = { f , m } ,
7083     Name-sg = Eindnoot ,
7084     name-sg = eindnoot ,
7085     Name-pl = Eindnoten ,
7086     name-pl = eindnoten ,
7087
7088     type = note ,
7089     gender = f ,
7090     Name-sg = Opmerking ,
7091     name-sg = opmerking ,
7092     Name-pl = Opmerkingen ,
7093     name-pl = opmerkingen ,
7094
7095     type = equation ,
7096     gender = f ,
7097     Name-sg = Vergelking ,
7098     name-sg = vergelking ,
7099     Name-pl = Vergelkingen ,
7100     name-pl = vergelkingen ,
7101     Name-sg-ab = Vgl. ,
7102     name-sg-ab = vgl. ,
7103     Name-pl-ab = Vgl.'s ,
7104     name-pl-ab = vgl.'s ,

```

```

7105   refbounds-first-sg = {,(,),} ,
7106   refbounds = {(,,)} ,
7107
7108   type = theorem ,
7109   gender = f ,
7110   Name-sg = Stelling ,
7111   name-sg = stelling ,
7112   Name-pl = Stellingen ,
7113   name-pl = stellingen ,
7114

```

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”.

```

7115   type = lemma ,
7116   gender = n ,
7117   Name-sg = Lemma ,
7118   name-sg = lemma ,
7119   Name-pl = Lemma’s ,
7120   name-pl = lemma’s ,
7121
7122   type = corollary ,
7123   gender = n ,
7124   Name-sg = Gevolg ,
7125   name-sg = gevolg ,
7126   Name-pl = Gevolgen ,
7127   name-pl = gevolgen ,
7128
7129   type = proposition ,
7130   gender = f ,
7131   Name-sg = Propositie ,
7132   name-sg = propositie ,
7133   Name-pl = Propositions ,
7134   name-pl = propositions ,
7135
7136   type = definition ,
7137   gender = f ,
7138   Name-sg = Definitie ,
7139   name-sg = definitie ,
7140   Name-pl = Definities ,
7141   name-pl = definities ,
7142
7143   type = proof ,
7144   gender = n ,
7145   Name-sg = Bews ,
7146   name-sg = bews ,
7147   Name-pl = Bewzen ,
7148   name-pl = bewzen ,
7149
7150   type = result ,
7151   gender = n ,
7152   Name-sg = Resultaat ,
7153   name-sg = resultaat ,
7154   Name-pl = Resultaten ,

```

```

7155     name-pl = resultaten ,
7156
7157 type = remark ,
7158     gender = f ,
7159     Name-sg = Opmerking ,
7160     name-sg = opmerking ,
7161     Name-pl = Opmerkingen ,
7162     name-pl = opmerkingen ,
7163
7164 type = example ,
7165     gender = n ,
7166     Name-sg = Voorbeeld ,
7167     name-sg = voorbeeld ,
7168     Name-pl = Voorbeelden ,
7169     name-pl = voorbeelden ,
7170

```

2022-12-27, ‘niluxv’: “algoritmes” is also a valid plural. “algoritmen” is chosen to be consistent with using “bijlagen” (and not “bijlages”) as the plural of “bijlage”.

```

7171 type = algorithm ,
7172     gender = { n , f , m } ,
7173     Name-sg = Algoritme ,
7174     name-sg = algoritme ,
7175     Name-pl = Algoritmen ,
7176     name-pl = algoritmen ,
7177

```

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

```

7178 type = listing ,
7179     gender = m ,
7180     Name-sg = Listing ,
7181     name-sg = listing ,
7182     Name-pl = Listings ,
7183     name-pl = listings ,
7184
7185 type = exercise ,
7186     gender = { f , m } ,
7187     Name-sg = Opgave ,
7188     name-sg = opgave ,
7189     Name-pl = Opgaven ,
7190     name-pl = opgaven ,
7191
7192 type = solution ,
7193     gender = f ,
7194     Name-sg = Oplossing ,
7195     name-sg = oplossing ,
7196     Name-pl = Oplossingen ,
7197     name-pl = oplossingen ,
7198 </lang-dutch>

```


10.8 Italian

Italian language file initially contributed by Matteo Ferrigato (issue #11), with the help of participants of the Gruppo Utilizzatori Italiani di T_EX (GuIT) forum (at <https://www.guitex.org/home/it/forum/5-tex-e-latex/121856-closed-zref-clever-e-localizzazione-in->

```
7199 <*package>
7200 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7201 </package>

7202 <*lang-italian>

7203 namesep    = {\nobreakspace} ,
7204 pairsep    = {\e\nobreakspace} ,
7205 listsep    = {,~} ,
7206 lastsep    = {\e\nobreakspace} ,
7207 tpairsep   = {\e\nobreakspace} ,
7208 tlistsep    = {,~} ,
7209 tlastsep   = {,~e\nobreakspace} ,
7210 notesep    = {~} ,
7211 rangesep   = {\a\nobreakspace} ,
7212 +refbounds-rb = {da\nobreakspace,,,} ,
7213
7214 type = book ,
7215     gender = m ,
7216     Name-sg = Libro ,
7217     name-sg = libro ,
7218     Name-pl = Libri ,
7219     name-pl = libri ,
7220
7221 type = part ,
7222     gender = f ,
7223     Name-sg = Parte ,
7224     name-sg = parte ,
7225     Name-pl = Parti ,
7226     name-pl = parti ,
7227
7228 type = chapter ,
7229     gender = m ,
7230     Name-sg = Capitolo ,
7231     name-sg = capitolo ,
7232     Name-pl = Capitoli ,
7233     name-pl = capitoli ,
7234
7235 type = section ,
7236     gender = m ,
7237     Name-sg = Paragrafo ,
7238     name-sg = paragrafo ,
7239     Name-pl = Paragrafi ,
7240     name-pl = paragrafi ,
7241
7242 type = paragraph ,
7243     gender = m ,
7244     Name-sg = Capoverso ,
7245     name-sg = capoverso ,
7246     Name-pl = Capoversi ,
```

```

7247     name-pl = capoversi ,
7248
7249 type = appendix ,
7250     gender = f ,
7251     Name-sg = Appendice ,
7252     name-sg = appendice ,
7253     Name-pl = Appendici ,
7254     name-pl = appendici ,
7255
7256 type = page ,
7257     gender = f ,
7258     Name-sg = Pagina ,
7259     name-sg = pagina ,
7260     Name-pl = Pagine ,
7261     name-pl = pagine ,
7262     Name-sg-ab = Pag. ,
7263     name-sg-ab = pag. ,
7264     Name-pl-ab = Pag. ,
7265     name-pl-ab = pag. ,
7266     rangesep = {\textendash} ,
7267     rangetopair = false ,
7268     +refbounds-rb = {,,} ,
7269
7270 type = line ,
7271     gender = f ,
7272     Name-sg = Riga ,
7273     name-sg = riga ,
7274     Name-pl = Righe ,
7275     name-pl = righe ,
7276
7277 type = figure ,
7278     gender = f ,
7279     Name-sg = Figura ,
7280     name-sg = figura ,
7281     Name-pl = Figure ,
7282     name-pl = figure ,
7283     Name-sg-ab = Fig. ,
7284     name-sg-ab = fig. ,
7285     Name-pl-ab = Fig. ,
7286     name-pl-ab = fig. ,
7287
7288 type = table ,
7289     gender = f ,
7290     Name-sg = Tabella ,
7291     name-sg = tabella ,
7292     Name-pl = Tabelle ,
7293     name-pl = tabelle ,
7294     Name-sg-ab = Tab. ,
7295     name-sg-ab = tab. ,
7296     Name-pl-ab = Tab. ,
7297     name-pl-ab = tab. ,
7298
7299 type = item ,
7300     gender = m ,

```

```

7301 Name-sg = Punto ,
7302 name-sg = punto ,
7303 Name-pl = Punti ,
7304 name-pl = punti ,
7305
7306 type = footnote ,
7307 gender = f ,
7308 Name-sg = Nota ,
7309 name-sg = nota ,
7310 Name-pl = Note ,
7311 name-pl = note ,
7312
7313 type = endnote ,
7314 gender = f ,
7315 Name-sg = Nota ,
7316 name-sg = nota ,
7317 Name-pl = Note ,
7318 name-pl = note ,
7319
7320 type = note ,
7321 gender = f ,
7322 Name-sg = Nota ,
7323 name-sg = nota ,
7324 Name-pl = Note ,
7325 name-pl = note ,
7326
7327 type = equation ,
7328 gender = f ,
7329 Name-sg = Equazione ,
7330 name-sg = equazione ,
7331 Name-pl = Equazioni ,
7332 name-pl = equazioni ,
7333 Name-sg-ab = Eq. ,
7334 name-sg-ab = eq. ,
7335 Name-pl-ab = Eq. ,
7336 name-pl-ab = eq. ,
7337 +refbounds-rb = {da\nobreakspace(,,)} ,
7338 refbounds-first-sg = {(,)}, ,
7339 refbounds = {(,,)} ,
7340
7341 type = theorem ,
7342 gender = m ,
7343 Name-sg = Teorema ,
7344 name-sg = teorema ,
7345 Name-pl = Teoremi ,
7346 name-pl = teoremi ,
7347
7348 type = lemma ,
7349 gender = m ,
7350 Name-sg = Lemma ,
7351 name-sg = lemma ,
7352 Name-pl = Lemmi ,
7353 name-pl = lemmi ,
7354

```

```

7355 type = corollary ,
7356     gender = m ,
7357     Name-sg = Corollario ,
7358     name-sg = corollario ,
7359     Name-pl = Corollari ,
7360     name-pl = corollari ,
7361
7362 type = proposition ,
7363     gender = f ,
7364     Name-sg = Proposizione ,
7365     name-sg = proposizione ,
7366     Name-pl = Proposizioni ,
7367     name-pl = proposizioni ,
7368
7369 type = definition ,
7370     gender = f ,
7371     Name-sg = Definizione ,
7372     name-sg = definizione ,
7373     Name-pl = Definizioni ,
7374     name-pl = definizioni ,
7375
7376 type = proof ,
7377     gender = f ,
7378     Name-sg = Dimostrazione ,
7379     name-sg = dimostrazione ,
7380     Name-pl = Dimostrazioni ,
7381     name-pl = dimostrazioni ,
7382
7383 type = result ,
7384     gender = m ,
7385     Name-sg = Risultato ,
7386     name-sg = risultato ,
7387     Name-pl = Risultati ,
7388     name-pl = risultati ,
7389
7390 type = remark ,
7391     gender = f ,
7392     Name-sg = Osservazione ,
7393     name-sg = osservazione ,
7394     Name-pl = Osservazioni ,
7395     name-pl = osservazioni ,
7396
7397 type = example ,
7398     gender = m ,
7399     Name-sg = Esempio ,
7400     name-sg = esempio ,
7401     Name-pl = Esempi ,
7402     name-pl = esempi ,
7403
7404 type = algorithm ,
7405     gender = m ,
7406     Name-sg = Algoritmo ,
7407     name-sg = algoritmo ,
7408     Name-pl = Algoritmi ,

```

```

7409     name-pl = algoritmi ,
7410
7411 type = listing ,
7412     gender = m ,
7413     Name-sg = Listato ,
7414     name-sg = listato ,
7415     Name-pl = Listati ,
7416     name-pl = listati ,
7417
7418 type = exercise ,
7419     gender = m ,
7420     Name-sg = Esercizio ,
7421     name-sg = esercizio ,
7422     Name-pl = Esercizi ,
7423     name-pl = esercizi ,
7424
7425 type = solution ,
7426     gender = f ,
7427     Name-sg = Soluzione ,
7428     name-sg = soluzione ,
7429     Name-pl = Soluzioni ,
7430     name-pl = soluzioni ,
7431 </lang-italian>

```

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A	
\A	1887
\AddToHook	110, 2033, 2077, 2100, 2131, 2133, 2171, 2244, 2286, 2449, 2462, 2470, 5329, 5365, 5382, 5384, 5389, 5426, 5430, 5432, 5437, 5447, 5457, 5491, 5542, 5567
\AddToHookWithArguments	2424
\alph	5472, 5558
\appendix	2, <i>127</i> , <i>129</i> , <i>137</i>
\appendixname	<i>127</i> , <i>138</i>
\AtEndOfPackage	2460
B	
\babelname	2087
\babelprovide	<i>30</i> , <i>55</i>
bool commands:	
\bool_gset_false:N	543
\bool_gset_true:N	536, 2436
\bool_if:NTF ...	<i>387</i> , <i>464</i> , 502, 560, 1763, 1814, 2037, 2041, 2426, 2472, 3446, 3849, 3990, 4117, 4153, 4187, 4254, 4267, 4279, 4330, 4347, 4357,
	4362, 4408, 4412, 4468, 4493, 4500, 4506, 4517, 4523, 4551, 4596, 4618, 4647, 4796, 4949, 4951, 5494, 5570
\bool_if:nTF	78, 3560, 3570, 3594, 3611, 3626, 3691, 3699, 4340, 4717, 4758, 4839, 4856
\bool_if_exist:NTF	341, 351, 375, 385, 435, 452, 462, 487, 490, 498, 500, 514, 517, 524, 527, 534, 541
\bool_lazy_all:nTF	4436, 4961
\bool_lazy_and:nnTF	51, 56, 2593, 3417, 3438, 4221, 4292, 4670, 4934, 4976
\bool_lazy_any:nTF	5113, 5122
\bool_lazy_or:nnTF	1380, 1413, 1963, 2542, 2806, 3305, 3330, 3421, 4922
\bool_new:N	342, 352, 377, 436, 454, 492, 515, 518, 525, 528, 535, 542, 802, 1577, 1578, 1605, 1629, 1981, 1988, 1995, 2008, 2009, 2179, 2180, 2181, 2182, 2183, 2279, 2280, 2417, 2429, 3454, 3469, 3731, 3732, 3743, 3744, 3751, 3753, 3754, 3767, 3768,

3769, 3781, 3782, 5456, 5501, 5541	\cs_new:Npn 66, 76, 86, 97,
\bool_set:Nn 3414	285, 291, 293, 295, 298, 309, 320,
\bool_set_eq:NN 550	332, 334, 722, 4713, 4753, 4791, 5146
\bool_set_false:N . . . 376, 453, 489,	\cs_new_protected:Npn 279, 336, 346,
491, 526, 1590, 1594, 1766, 1864,	356, 363, 370, 395, 405, 426, 428,
1909, 1951, 2016, 2025, 2026, 2043,	430, 440, 447, 485, 512, 522, 532,
2050, 2191, 2195, 2202, 2210, 2211,	539, 815, 917, 1526, 1545, 1749,
2212, 2304, 2316, 3459, 3552, 3798,	1848, 1893, 1935, 2468, 2599, 3409,
3799, 3816, 3855, 3866, 4266, 4458,	3473, 3515, 3526, 3539, 3669, 3721,
4459, 4466, 4467, 4700, 5120, 5137	3783, 3997, 4462, 4709, 4711, 4905,
\bool_set_true:N	5140, 5157, 5228, 5261, 5294, 5509
. 343, 353, 437, 516, 519,	\cs_set_eq:NN 724
529, 1584, 1585, 1589, 1595, 1788,	\cs_to_str:N 335
1810, 1873, 1875, 1913, 1915, 1924,	
1926, 1955, 1957, 1970, 1972, 2015,	D
2020, 2021, 2189, 2196, 2201, 2218,	\d 1887
2220, 2222, 2225, 2226, 2227, 2292,	E
2297, 3566, 3576, 3580, 3602, 3617,	\endinput 14
3632, 3656, 3824, 3850, 3856, 3860,	exp commands:
3867, 4013, 4024, 4035, 4085, 4121,	\exp_args:NNe 38, 41
4157, 4191, 4208, 4289, 4323, 4495,	\exp_args:NNNo 281
4555, 4601, 4623, 4652, 4910, 4917,	\exp_args:NNNV 1828, 1877, 1928, 1974
5039, 5098, 5136, 5173, 5180, 5181,	\exp_args:NNNo 281, 287
5199, 5216, 5218, 5470, 5508, 5555	\exp_args:No 287
\bool_until_do:Nn	\exp_args:Nx 927, 5514, 5522
. 1865, 1916, 1958, 3592, 3817	\exp_args:Nxx
\bool_while_do:nn 5600	1767, 1779, 1791, 1801, 1867, 1918,
\l_tmpa_bool 1766, 1788,	1960, 5159, 5165, 5187, 5191, 5206
1810, 1814, 1864, 1865, 1873, 1875,	\exp_not:N 113,
1909, 1913, 1915, 1916, 1924, 1926,	4364, 4377, 4380, 4383, 4729, 4733,
1951, 1955, 1957, 1958, 1970, 1972	4741, 4745, 4770, 4773, 4781, 4784,
	4812, 4815, 4819, 4824, 4830, 4833,
C	4845, 4848, 4876, 4881, 4891, 4896
\chaptername 138	\exp_not:n 288, 4029, 4052,
clist commands:	4060, 4078, 4091, 4095, 4130, 4139,
\clist_map_inline:nn	4161, 4169, 4176, 4200, 4214, 4231,
. 635, 683, 700, 1019,	4241, 4283, 4306, 4316, 4365, 4376,
2213, 2365, 2438, 2942, 5473, 5560	4381, 4382, 4539, 4562, 4574, 4608,
\contsubbottom 129	4630, 4639, 4659, 4680, 4690, 4730,
\contsubcaption 129	4742, 4771, 4772, 4782, 4783, 4813,
\counterwithin 5	4814, 4816, 4820, 4831, 4832, 4834,
\crefstripprefix 45	4842, 4846, 4847, 4850, 4877, 4892
cs commands:	\ExplSyntaxOn 30, 933
\cs_generate_variant:Nn 75,	
284, 290, 297, 308, 319, 330, 345,	F
355, 362, 369, 380, 404, 414, 439,	\figurename 138
446, 457, 495, 521, 531, 538, 545,	file commands:
973, 1847, 1883, 1934, 1980, 2601,	\file_get:nnNTF 927
4752, 4790, 5156, 5260, 5293, 5326	\fmtversion 5
\cs_if_exist:NTF	\footnote 2, 129, 130
. 26, 29, 68, 88, 5353, 5359	
\cs_if_exist_p:N	G
. 53, 58, 4223, 4294, 4672, 5602	group commands:
\cs_if_exist_use:N 5425, 5640	\group_begin: 738, 919, 1765, 1852,

1897, 1939, 2866, 3411, 3425, 3457,
4364, 4380, 4729, 4741, 4770, 4781,
4812, 4819, 4830, 4845, 4876, 4891
\group_end: .. 750, 971, 1829, 1878,
1929, 1975, 2894, 3428, 3451, 3461,
4377, 4383, 4733, 4745, 4773, 4784,
4815, 4824, 4833, 4848, 4881, 4896

H

\hyperlink 5143

I

\IfBooleanT 3458
\IfClassLoadedTF 123
\ifdraft 2194
\IfFormatAtLeastTF 5, 6
\ifoptionfinal 2200
\IfPackageAtLeastTF 2290
\IfPackageLoadedTF 121
\input 30

int commands:

\int_case:nnTF
.. 4000, 4042, 4104, 4415, 4530, 4587
\int_compare:nNnTF
3603, 3618, 3633, 3645, 3657, 3677,
3679, 3723, 3897, 4020, 4048, 4070,
4125, 4195, 4400, 4402, 4479, 4509,
4568, 5169, 5175, 5195, 5201, 5618
\int_compare_p:nNn 1383, 1416, 1964,
1965, 2544, 2808, 3308, 3333, 3693,
3701, 4440, 4926, 4937, 4966, 5133
\int_gincr:N 114
\int_incr:N
... 4455, 4487, 4499, 4501, 4516,
4518, 4522, 4524, 4536, 4559, 4571,
4605, 4627, 4636, 4656, 4707, 5616
\int_new:N 108, 3470,
3471, 3733, 3734, 3735, 3748, 3749
\int_rand:n 306, 317, 328
\int_set:Nn 3678, 3680, 3684, 3687, 5599
\int_to_roman:n 5603, 5610, 5611, 5614
\int_use:N 54, 59, 64, 70, 118
\int_zero:N 3671,
3672, 3793, 3794, 3795, 3796, 3797,
4453, 4454, 4456, 4457, 4702, 4703
\l_tmpa_int 5599,
5603, 5610, 5611, 5614, 5616, 5618

iow commands:

\iow_char:N
.... 127, 142, 143, 148, 149, 154,
155, 160, 161, 213, 230, 277, 2261, 2270
\iow_newline: 271

J

\jobname 5447

K

keys commands:

\l_keys_choice_tl 807
\keys_define:nn
. 20, 643, 689, 706, 767, 974, 1063,
1088, 1116, 1325, 1364, 1442, 1552,
1579, 1606, 1615, 1630, 1639, 1982,
1989, 1996, 2002, 2010, 2045, 2054,
2068, 2096, 2135, 2166, 2173, 2185,
2246, 2253, 2255, 2264, 2274, 2281,
2293, 2305, 2317, 2325, 2332, 2361,
2387, 2411, 2418, 2431, 2451, 2480,
2498, 2528, 2562, 2585, 2611, 2623,
2647, 2759, 2789, 2829, 2897, 2968,
2992, 3019, 3225, 3255, 3295, 3357
\keys_set:nn 27, 30, 57, 59, 84, 747,
895, 958, 2298, 2600, 2605, 2891, 3412

keyval commands:

\keyval_parse:nnn ... 1531, 2336, 2391

L

\label 2, 60, 63, 133
\labelformat 3
\language_name 24, 139, 2081

M

\mainbabelname 24, 2088
\MessageBreak 12

MH commands:

\MH_if_boolean:nTF 5506

msg commands:

\msg_info:nnn 961,
1014, 1079, 1272, 1278, 1332, 5403,
5449, 5498, 5533, 5574, 5592, 5619
\msg_info:nnnn
..... 987, 994, 1024, 1396, 1430
\msg_info:nnnnn 1008
\msg_line_context: 126, 132,
136, 138, 141, 147, 153, 159, 165,
170, 175, 180, 185, 191, 196, 199,
202, 207, 211, 218, 223, 228, 235,
244, 249, 254, 258, 260, 262, 264, 271
\msg_new:nnn 124, 130,
135, 137, 139, 145, 151, 157, 163,
168, 173, 178, 183, 188, 193, 198,
200, 205, 210, 212, 214, 216, 221,
226, 232, 234, 236, 241, 247, 252,
257, 259, 261, 263, 265, 267, 269, 274
\msg_warning:nn
..... 2042, 2048, 2320, 2596, 4442
\msg_warning:nnn
..... 742, 763, 1558, 1565, 1721,
1727, 2120, 2157, 2169, 2231, 2242,
2284, 2309, 2393, 2455, 2465, 2615,

<code>\zref@localaddprop</code>	<code>\tl_if_exist:N</code>
..... 5427, 5495, 5571, 5642	338,
<code>\ZREF@mainlist</code>	348, 358, 365, 372, 383, 399, 409, 728
63, 65, 107, 119, 5427, 5495, 5571, 5642	<code>\tl_if_exist_p:N</code>
<code>\zref@newprop</code>	2594
6, 7, 22, 24, 34,	<code>\tl_if_novalue:nTF</code>
49, 64, 102, 118, 5424, 5472, 5558, 5639 2485, 2533, 2628, 2794
<code>\zref@refused</code>	<code>\tl_map_break:n</code>
3874	100
<code>\zref@wrapper@babel</code> 63, 83, 2427, 3408	<code>\tl_map_tokens:Nn</code>
<code>\textendash</code>	92
1543,	<code>\tl_new:N</code>
5731, 5995, 6607, 6829, 7043, 7266	109, 339, 349,
<code>\thechapter</code>	359, 366, 400, 410, 565, 566, 567,
127	717, 718, 719, 720, 744, 759, 1551,
<code>\thelstnumber</code>	2165, 2184, 2252, 2273, 2324, 2410,
134	3463, 3464, 3465, 3466, 3467, 3468,
<code>\thepage</code>	3736, 3737, 3738, 3739, 3740, 3741,
7, 112, 115	3742, 3745, 3746, 3750, 3752, 3755,
<code>\thesection</code>	3756, 3757, 3758, 3759, 3760, 3761,
127	3762, 3763, 3764, 3765, 3766, 5428
tl commands:	<code>\tl_put_left:Nn</code> . 4343, 4350, 4393,
<code>\c_novalue_tl</code> .. 691, 692, 693, 694,	5024, 5025, 5067, 5069, 5071, 5073
695, 696, 697, 2482, 2530, 2625, 2791	<code>\tl_put_right:Nn</code> . 4027, 4050, 4058,
<code>\tl_clear:N</code>	4076, 4089, 4128, 4137, 4159, 4167,
350, 374, 835,	4174, 4198, 4212, 4229, 4239, 4537,
873, 886, 903, 912, 937, 946, 979,	4560, 4572, 4606, 4628, 4637, 4657,
2606, 2869, 2879, 2902, 3787, 3788,	4678, 4688, 4932, 4933, 4944, 5641
3789, 3790, 3791, 3792, 3823, 4448,	<code>\tl_reverse:N</code>
4449, 4450, 4451, 4452, 4498, 4515,	3543, 3546
4909, 4916, 4946, 5038, 5097, 5254	<code>\tl_set:Nn</code>
<code>\tl_const:Nn</code> 281, 340, 721, 746, 936, 980, 992,
1528	1560, 1567, 1569, 1758, 1826, 1830,
<code>\tl_gc_clear:N</code>	1843, 1854, 1859, 1870, 1871, 1879,
367, 411	1881, 1899, 1904, 1921, 1922, 1930,
<code>\tl_gset:Nn</code>	1932, 1941, 1946, 1967, 1968, 1976,
360, 401, 745, 760	1978, 2081, 2082, 2087, 2088, 2091,
<code>\tl_gset_eq:NN</code>	2092, 2106, 2112, 2117, 2143, 2149,
115	2154, 2604, 2870, 2903, 2915, 3650,
<code>\tl_head:N</code>	3652, 3833, 3834, 4007, 4009, 4281,
.....	4304, 4314, 4360, 4483, 4485, 4496,
.. 3621, 3634, 3646, 3648, 3658, 3660	4513, 4928, 4929, 4942, 5429, 5431
<code>\tl_head:n</code> 1868, 1919, 1961, 1965	<code>\tl_set_eq:NN</code>
<code>\tl_if_empty:N</code>	419, 4446
36, 90,	<code>\tl_show:N</code>
829, 839, 866, 877, 898, 905, 1012,	4409
1068, 1093, 1125, 1160, 1197, 1234,	<code>\tl_tail:N</code>
1282, 1330, 1336, 1369, 1447, 1485,	3651, 3653
1751, 1872, 1923, 2935, 2973, 2997,	<code>\tl_tail:n</code>
3028, 3063, 3100, 3137, 3185, 3260,
3266, 3300, 3362, 3383, 3429, 4256,	.. 1870, 1871, 1921, 1922, 1967, 1968
4849, 4931, 4953, 5011, 5022, 5065	<code>\tl_use:N</code>
<code>\tl_if_empty:nTF</code>	303,
.. 739, 755, 978, 1270, 1547, 1556,	314, 325, 761, 925, 930, 960, 962, 966
1719, 2435, 2727, 2901, 3173, 5142	<code>\l_tmpa_tl</code>
<code>\tl_if_empty_p:N</code>	934, 958, 1854,
.....	1868, 1870, 1899, 1910, 1919, 1921,
52, 57, 2595, 4222, 4293,	1941, 1952, 1961, 1967, 3434, 3435
4671, 4964, 4978, 5117, 5127, 5131	<code>\l_tmpb_tl</code> . 1816, 1823, 1826, 1830,
<code>\tl_if_empty_p:n</code> 1381, 1414,	1859, 1868, 1871, 1872, 1879, 1904,
2543, 2807, 3306, 3331, 3562, 3563,	1912, 1919, 1922, 1923, 1930, 1946,
3572, 3573, 3598, 3599, 3614, 3629	1954, 1961, 1964, 1965, 1968, 1976
<code>\tl_if_eq:NNTF</code> . 112, 3533, 3556, 3863	
<code>\tl_if_eq:NnTF</code>	
... 1777, 3476, 3508, 3683, 3686,	
3711, 3714, 3831, 3879, 4913, 5163	
<code>\tl_if_eq:nnTF</code> .. 1767, 1779, 1791,	
1801, 1867, 1918, 1960, 3675, 5159,	
5165, 5187, 5191, 5206, 5514, 5522	

U

use commands:

`\use:N` 27, 30, 807, 4225, 4300, 4674, 5320

`\UseHook` 1853, 1898, 1940

V

`\value` 5368, 5387
`\verbfootnote` 129, 130

Z

`\Z` 1887
`\zcDeclareLanguage` . 13, 25, 736, 5666, 5871, 6324, 6539, 6768, 6982, 7200
`\zcDeclareLanguageAlias`
 .. 26, 753, 5667, 5668, 5669, 5670, 5671, 5672, 5673, 5874, 5875, 5876, 5877, 5878, 6325, 6540, 6541, 6542
`\zcLanguageSetup`
 21, 30, 68, 73, 74, 139, 2864
`\zcpageref` 85, 3455
`\zceref` 21, 65, 67, 83, 85–87, 93, 95, 132, 3407, 3460
`\zcRefTypeSetup` 21, 68, 2602
`\zcsetup` 21, 55, 65, 67, 2597
`\zlabel` 2, 63, 129, 131, 133

zrefcheck commands:

`\zrefcheck_zceref_beg_label:` .. 3420
`\zrefcheck_zceref_end_label_-maybe:` 3442
`\zrefcheck_zceref_run_checks_on_labels:n` 3443

zrefclever commands:

`\zrefclever_language_if_declared:n` 734
`\zrefclever_language_if_declared:nTF` 734
`\zrefclever_language_varname:n` .. 724, 724
`\l_zrefclever_ref_language_tl` .. 720

zrefclever internal commands:

`\l_zrefclever_abbrev_bool` 3755, 3942, 4935
`\l_zrefclever_amsmath_subequations_bool` 5456, 5470, 5494
`\l_zrefclever_breqn_dgroup_bool` 5541, 5555, 5570
`\l_zrefclever_cap_bool` 3755, 3938, 4923
`\l_zrefclever_capfirst_bool` ... 1988, 1991, 4925
`_zrefclever_compat_module:nn` .. 64, 2468, 2468, 5327, 5345, 5406, 5452, 5502, 5537, 5577, 5595, 5622, 5645
`_zrefclever_counter_reset_by:n` 6, 61, 62, 68, 70, 72, 76, 76, 5464, 5549
`_zrefclever_counter_reset_by_-aux:nn` 83, 86
`_zrefclever_counter_reset_by_-aux:nnn` 93, 97
`\l_zrefclever_counter_resetby_-prop` 6, 62, 79, 80, 2386, 2398
`\l_zrefclever_counter_resettters_seq` 5, 6, 61, 62, 82, 2360, 2367, 2370
`\l_zrefclever_counter_type_prop` 4, 60, 61, 38, 41, 2331, 2343
`\l_zrefclever_current_counter_-tl` 3, 5, 63, 22, 26, 27, 39, 42, 44, 52, 53, 54, 105, 2410, 2413
`\l_zrefclever_current_language_-tl` 24, 55, 718, 2081, 2087, 2091, 2107, 2144
`\l_zrefclever_endrangefunc_tl` .. 3755, 3930, 4222, 4223, 4225, 4293, 4294, 4300, 4671, 4672, 4674
`\l_zrefclever_endrangeprop_tl` .. 48, 1751, 1761, 3755, 3934
`_zrefclever_extract:nnn` 12, 291, 291, 1856, 1861, 1901, 1906, 1943, 1948, 3604, 3606, 3619, 3636, 3724, 3726, 5170, 5172, 5176, 5178, 5196, 5198, 5202, 5204
`_zrefclever_extract_default:Nnnn` ... 11, 279, 279, 284, 1755, 1816, 1823, 1833, 1840, 3517, 3528, 3530, 3541, 3544, 3547, 3549, 3837, 3840
`_zrefclever_extract_unexp:nnn` . 11, 122, 285, 285, 290, 1769, 1773, 1781, 1785, 1793, 1797, 1803, 1807, 4372, 4726, 4731, 4743, 4767, 4808, 4821, 4870, 4878, 4893, 5149, 5152, 5153, 5160, 5161, 5166, 5167, 5188, 5189, 5192, 5193, 5208, 5212, 5515, 5523
`_zrefclever_extract_url_-unexp:n` 4368, 4725, 4766, 4804, 4866, 5146, 5146, 5156
`_zrefclever_get_enclosing_-counters_value:n` 6, 66, 66, 71, 75, 104
`_zrefclever_get_endrange_-pagecomp:nnN` 1893, 1934
`_zrefclever_get_endrange_-pagecomptwo:nnN` 1935, 1980
`_zrefclever_get_endrange_-property:nnN` 45, 1749, 1847
`_zrefclever_get_endrange_-stripprefix:nnN` 1848, 1883
`_zrefclever_get_ref:nnN` 113, 114, 4030, 4053, 4061, 4079,

4092, 4096, 4131, 4140, 4162, 4170,
 4177, 4201, 4215, 4242, 4284, 4317,
 4352, 4540, 4563, 4575, 4609, 4631,
 4640, 4660, 4691, 4713, 4713, 4752
 _zrefclever_get_ref_endrange:nnN
 45,
 114, 4232, 4307, 4681, 4753, 4753, 4790
 _zrefclever_get_ref_first: ...
 113, 114, 118, 4344, 4394, 4791, 4791
 _zrefclever_get_rf_opt_bool:nN 126
 _zrefclever_get_rf_opt_-
 bool:nnnnN 20,
 3935, 3939, 3943, 5294, 5294, 5326
 _zrefclever_get_rf_opt_-
 seq:nnnN .. 20, 125, 3947, 3951,
 3955, 3959, 3963, 3967, 3971, 3975,
 3979, 3983, 4980, 5261, 5261, 5293
 _zrefclever_get_rf_opt_tl:nnnN
 20,
 22, 45, 125, 3431, 3802, 3806, 3810,
 3899, 3903, 3907, 3911, 3915, 3919,
 3923, 3927, 3931, 5228, 5228, 5260
 _zrefclever_hyperlink:nnn
 122, 4366,
 4724, 4765, 4802, 4864, 5140, 5140
 \l_zrefclever_hyperlink_bool ...
 2008, 2015, 2020, 2025, 2037, 2043,
 2050, 3459, 4719, 4760, 4858, 5115
 \l_zrefclever_hyperref_warn_-
 bool ... 2009, 2016, 2021, 2026, 2041
 _zrefclever_if_class_loaded:n .
 120, 122
 _zrefclever_if_class_loaded:nTF
 5408
 _zrefclever_if_package_-
 loaded:n 120, 120
 _zrefclever_if_package_-
 loaded:nTF 2035,
 2079, 2085, 2288, 5347, 5454,
 5504, 5539, 5579, 5597, 5624, 5647
 _zrefclever_is_integer_rgx:n ..
 1884, 1885, 1892
 _zrefclever_is_integer_rgx:nTF
 1910, 1912, 1952, 1954
 \l_zrefclever_label_a_tl
 . 92, 3736, 3820, 3839, 3852, 3874,
 3876, 3882, 3885, 3891, 4008, 4030,
 4053, 4061, 4096, 4162, 4177, 4227,
 4233, 4242, 4274, 4284, 4302, 4308,
 4317, 4470, 4474, 4484, 4497, 4514,
 4540, 4576, 4640, 4676, 4682, 4691
 \l_zrefclever_label_b_tl
 92, 3736,
 3823, 3828, 3842, 3854, 3859, 4474
 \l_zrefclever_label_count_int ..
 93, 3733, 3793,
 3897, 4000, 4453, 4479, 4707, 4967
 \l_zrefclever_label_enclval_a_-
 tl 3463, 3541, 3543, 3598,
 3614, 3634, 3646, 3650, 3651, 3658
 \l_zrefclever_label_enclval_b_-
 tl 3463, 3544, 3546, 3599,
 3621, 3629, 3648, 3652, 3653, 3660
 \l_zrefclever_label_extdoc_a_tl
 .. 3463, 3547, 3557, 3562, 3572, 3585
 \l_zrefclever_label_extdoc_b_tl
 .. 3463, 3549, 3558, 3563, 3573, 3584
 \l_zrefclever_label_type_a_tl ..
 3432, 3463, 3518, 3520,
 3523, 3529, 3534, 3683, 3711, 3803,
 3807, 3811, 3833, 3838, 3864, 3879,
 3900, 3904, 3908, 3912, 3916, 3920,
 3924, 3928, 3932, 3936, 3940, 3944,
 3948, 3952, 3956, 3960, 3964, 3968,
 3972, 3976, 3980, 3984, 4010, 4486
 \l_zrefclever_label_type_b_tl ..
 3463, 3531,
 3535, 3686, 3714, 3834, 3841, 3865
 _zrefclever_label_type_put_-
 new_right:n 86, 87, 3479, 3515, 3515
 \l_zrefclever_label_types_seq ..
 87, 3472, 3475, 3519, 3522, 3709
 \l_zrefclever_labelhook_bool ...
 2417, 2420, 2426
 _zrefclever_labels_in_sequence:nn
 .. 48, 93, 123, 4272, 4473, 5157, 5157
 \l_zrefclever_lang_decl_case_tl
 565, 946, 949, 992, 997, 1336, 1353,
 2879, 2882, 2915, 2920, 3266, 3283
 \l_zrefclever_lang_declension_-
 seq 565,
 825, 826, 827, 841, 845, 852, 943,
 944, 945, 948, 985, 991, 996, 2876,
 2877, 2878, 2881, 2908, 2914, 2919
 \l_zrefclever_lang_gender_seq ..
 565, 862, 863, 864, 879, 956,
 957, 1006, 1021, 2889, 2890, 2929, 2944
 _zrefclever_language_if_-
 declared:n 25, 726, 733, 735
 _zrefclever_language_if_-
 declared:n(TF) 25
 _zrefclever_language_if_-
 declared:nTF 300, 311, 322, 726,
 741, 757, 817, 921, 2118, 2155, 2867
 _zrefclever_language_varname:n
 24, 25, 303, 314,
 325, 722, 722, 725, 728, 744, 745,
 759, 760, 761, 925, 930, 960, 962, 966

\l_zrefclever_last_of_type_bool	\l_zrefclever_nocompat_modules_-
..... 93, 3730, 3850,	seq 64
3855, 3856, 3860, 3866, 3867, 3990	\l_zrefclever_nudge_comptosing_-
\l_zrefclever_lastsep_tl . 3755,	bool ... 2181, 2211, 2220, 2226, 4963
3914, 4060, 4095, 4139, 4176, 4214	\l_zrefclever_nudge_enabled_-
\l_zrefclever_link_star_bool ...	bool 2179, 2189, 2191,
.. 3414, 3453, 4720, 4761, 4859, 5116	2195, 2196, 2201, 2202, 4438, 4949
\l_zrefclever_listsep_tl	\l_zrefclever_nudge_gender_bool
... 3755, 3910, 4091, 4169, 4539, 2183, 2212, 2222, 2227, 4977
4562, 4574, 4608, 4630, 4639, 4659	\l_zrefclever_nudge_multitype_-
\g_zrefclever_loaded_langfiles_-	bool ... 2180, 2210, 2218, 2225, 4439
seq 916, 924, 959, 965	\l_zrefclever_nudge_singular_-
\l_zrefclever_main_language_tl .	bool 2182, 2238, 4951
..... 24,	_zrefclever_opt_bool_get:NN ...
55, 719, 2082, 2088, 2092, 2113, 2150 546, 556
_zrefclever_mathtools_showonlyrefs:n	_zrefclever_opt_bool_get:NN(TF)
..... 3448, 5509 19
\l_zrefclever_mathtools_-	_zrefclever_opt_bool_get:NNTF .
showonlyrefs_bool 3446, 5501, 5508	... 546, 5297, 5302, 5307, 5312, 5317
\l_zrefclever_memoir_footnote_-	_zrefclever_opt_bool_gset_-
type_tl 5428, 5429, 5431, 5435	false:N 19,
_zrefclever_name_default:	512, 539, 545, 1494, 1511, 3385, 3393
..... 4709, 4711, 4841	_zrefclever_opt_bool_gset_-
\l_zrefclever_name_format_-	true:N 19,
fallback_tl 3742, 4942,	512, 532, 538, 1456, 1473, 3364, 3372
4946, 5011, 5060, 5072, 5074, 5092	_zrefclever_opt_bool_if:N 557, 564
\l_zrefclever_name_format_tl ...	_zrefclever_opt_bool_if:N(TF) . 20
... 3742, 4928, 4929, 4932, 4933,	_zrefclever_opt_bool_if:N(TF) ...
4943, 4944, 5017, 5024, 5025, 5033, 557, 890
5041, 5051, 5068, 5069, 5082, 5102	_zrefclever_opt_bool_if_set:N .
\l_zrefclever_name_in_link_bool 496, 511
..... 115,	_zrefclever_opt_bool_if_-
118, 3742, 4362, 4796, 5120, 5136, 5137	set:N(TF) 18
\l_zrefclever_namefont_tl 3755,	_zrefclever_opt_bool_if_-
3922, 4365, 4381, 4813, 4831, 4846	set:N(TF) 496,
\l_zrefclever_nameinlink_str ...	548, 559, 1449, 1465, 1487, 1503
..... 2053, 2058, 2060,	_zrefclever_opt_bool_set_-
2062, 2064, 5118, 5124, 5126, 5130	false:N 19, 512, 522, 531, 2572, 2843
\l_zrefclever_namesep_tl	_zrefclever_opt_bool_set_-
.. 3755, 3902, 4816, 4834, 4842, 4850	true:N 19, 512, 512, 521, 2567, 2834
\l_zrefclever_next_is_same_bool	_zrefclever_opt_bool_unset:N ...
..... 93, 123, 3748, 18, 485, 485, 495, 2577, 2852
4467, 4500, 4517, 4523, 5181, 5219	_zrefclever_opt_seq_get:NN 474, 484
\l_zrefclever_next_maybe_range_-	_zrefclever_opt_seq_get:NN(TF) 18
bool 93, 123, 3748, 4266, 4279, 4466,	_zrefclever_opt_seq_get:NNTF ...
4493, 4506, 5173, 5180, 5199, 5217	... 474, 820, 857, 938, 951, 2871,
\l_zrefclever_noabbrev_first_-	2884, 5264, 5269, 5274, 5279, 5284
bool 1995, 1998, 4939	_zrefclever_opt_seq_gset_-
\g_zrefclever_nocompat_bool ...	clist_split:Nn 16,
..... 2429, 2436, 2472	426, 428, 1378, 1411, 3303, 3328
\l_zrefclever_nocompat_bool ... 64	_zrefclever_opt_seq_gset_eq:NN
\g_zrefclever_nocompat_modules_- 16, 426,
seq 2430, 2440, 2443, 2464, 2473, 2474	440, 446, 1387, 1420, 2952, 3312, 3337

```

\__zrefclever_opt_seq_if_set:N . .
    . . . . . 458, 473
\__zrefclever_opt_seq_if_-
    set:N(TF) . . . . . 17
\__zrefclever_opt_seq_if_set:NTF
    . . . . . 458, 476, 1029, 1371, 1403
\__zrefclever_opt_seq_set_clist_-
    split:Nn . . 16, 426, 426, 2540, 2804
\__zrefclever_opt_seq_set_eq:NN .
    . . . . . 16, 426, 430, 439, 2546, 2810
\__zrefclever_opt_seq_unset:N . . .
    . . . . . 17, 447, 447, 457, 2535, 2796
\__zrefclever_opt_tl_clear:N . . .
    . . . . . 14, 336,
    346, 355, 1647, 1652, 1667, 1682,
    1697, 2655, 2660, 2675, 2690, 2705
\__zrefclever_opt_tl_cset_-
    fallback:nn . . . . . 1526, 1533
\__zrefclever_opt_tl_gclear:N . . .
    . . . . . 14, 336,
    363, 369, 3030, 3036, 3044, 3051,
    3072, 3088, 3109, 3125, 3146, 3162
\__zrefclever_opt_tl_gclear_if_-
    new:N . . . . . 16, 395,
    405, 414, 1127, 1133, 1141, 1148,
    1169, 1185, 1206, 1222, 1243, 1259
\__zrefclever_opt_tl_get:NN 415, 425
\__zrefclever_opt_tl_get:NN(TF) . 16
\__zrefclever_opt_tl_get:NNTF . . .
    415, 5013, 5028, 5047, 5056, 5077,
    5087, 5231, 5236, 5241, 5246, 5251
\__zrefclever_opt_tl_gset:N . . . 14
\__zrefclever_opt_tl_gset:Nn . . .
    . . 336, 356, 362, 2975, 2999, 3007,
    3065, 3080, 3102, 3117, 3139, 3154,
    3187, 3194, 3203, 3211, 3268, 3278
\__zrefclever_opt_tl_gset_if_-
    new:Nn . . . . . 16,
    395, 395, 404, 1070, 1095, 1104,
    1162, 1177, 1199, 1214, 1236, 1251,
    1284, 1291, 1300, 1308, 1338, 1348
\__zrefclever_opt_tl_if_set:N . . 381
\__zrefclever_opt_tl_if_set:N(TF)
    . . . . . 15
\__zrefclever_opt_tl_if_set:NTF .
    . . . . . 381, 397, 407, 417
\__zrefclever_opt_tl_set:N . . . . 14
\__zrefclever_opt_tl_set:Nn . . . .
    . . . . . 336, 336, 345,
    1661, 1676, 1691, 1731, 1737, 2491,
    2637, 2669, 2684, 2699, 2739, 2746
\__zrefclever_opt_tl_unset:N . . .
    . . . . . 15, 370, 370,
    380, 1706, 1711, 2487, 2630, 2714, 2719

\__zrefclever_opt_var_set_bool:n
    . . . . . 14, 334, 334, 341,
    342, 343, 351, 352, 353, 375, 376,
    377, 385, 387, 435, 436, 437, 452,
    453, 454, 462, 464, 490, 491, 492,
    500, 502, 517, 518, 519, 527, 528, 529
\__zrefclever_opt_varname_-
    fallback:nn . . . . . 14,
    332, 332, 1529, 5252, 5285, 5318
\__zrefclever_opt_varname_-
    general:nn . . . . . 12,
    293, 293, 1649, 1654, 1663, 1669,
    1678, 1684, 1693, 1699, 1708, 1713,
    1733, 1739, 2488, 2492, 2536, 2547,
    2568, 2573, 2578, 5232, 5265, 5298
\__zrefclever_opt_varname_lang_-
    default:nnn . . 13, 309, 309, 319,
    1072, 1097, 1129, 1135, 1164, 1171,
    1201, 1208, 1238, 1245, 1286, 1293,
    1373, 1389, 1451, 1458, 1489, 1496,
    2977, 3001, 3032, 3038, 3067, 3074,
    3104, 3111, 3141, 3148, 3189, 3196,
    3314, 3366, 3387, 5247, 5280, 5313
\__zrefclever_opt_varname_lang_-
    type:nnnn . . . . . 13,
    320, 320, 331, 1031, 1040, 1048,
    1106, 1143, 1150, 1179, 1187, 1216,
    1224, 1253, 1261, 1302, 1310, 1340,
    1350, 1405, 1422, 1467, 1475, 1505,
    1513, 2954, 3009, 3046, 3053, 3082,
    3090, 3119, 3127, 3156, 3164, 3205,
    3213, 3270, 3280, 3339, 3374, 3395,
    5030, 5079, 5089, 5242, 5275, 5308
\__zrefclever_opt_varname_-
    language:nnn . . . . . 13, 298,
    298, 308, 773, 778, 788, 793, 804,
    809, 822, 859, 892, 940, 953, 2873, 2886
\__zrefclever_opt_varname_-
    type:nnn 12, 295, 295, 297, 2632,
    2639, 2657, 2662, 2671, 2677, 2686,
    2692, 2701, 2707, 2716, 2721, 2741,
    2748, 2798, 2812, 2836, 2845, 2854,
    5015, 5049, 5058, 5237, 5270, 5303
\g__zrefclever_page_format_int . .
    . . . . . 108, 114, 118
\l__zrefclever_pairsep_tl . . . . .
    . . . . . 3755, 3906, 4029,
    4052, 4078, 4130, 4161, 4200, 4283
\g__zrefclever_prev_page_format_-
    tl . . . . . 7, 109, 112, 115
\__zrefclever_process_language_-
    settings: . . . 57, 59, 815, 815, 3416
\__zrefclever_prop_put_non_-
    empty:Nnn 43, 1545, 1545, 2342, 2397

```

_zrefclever_provide_langfile:n	1560, 1567, 1569, 1753, 1777, 1821,
..... 21,	1838, 1850, 1857, 1862, 1895, 1902,
30, 32, 84, 917, 917, 973, 2124, 3415	1907, 1937, 1944, 1949, 3508, 3831,
\l_zrefclever_range_beg_is_	3886, 3890, 4715, 4800, 4854, 5163
first_bool 3748,	\l_zrefclever_ref_propserity_tl 3476
3798, 4117, 4153, 4187, 4458,	\l_zrefclever_ref_typeset_font_
4495, 4551, 4596, 4618, 4647, 4700	tl 2252, 2254, 3426
\l_zrefclever_range_beg_label_	\l_zrefclever_refbounds_first_
tl 93,	pb_seq 3770,
3748, 3791, 4080, 4093, 4132, 4141,	3958, 4034, 4084, 4156, 4207, 4288
4171, 4202, 4216, 4226, 4451, 4496,	\l_zrefclever_refbounds_first_
4513, 4564, 4610, 4632, 4661, 4675	rb_seq . 3770, 3962, 4190, 4322, 4651
\l_zrefclever_range_count_int ..	\l_zrefclever_refbounds_first_
..... 93,	seq 3770, 3950, 4333, 4554, 4600, 4622
3748, 3796, 4042, 4106, 4456, 4499,	\l_zrefclever_refbounds_first_
4510, 4516, 4522, 4530, 4589, 4702	sg_seq . 3770, 3954, 4012, 4023, 4120
\l_zrefclever_range_end_ref_tl .	\l_zrefclever_refbounds_last_
... 3748, 3792, 4228, 4234, 4303,	pe_seq 3770, 3982,
4309, 4452, 4498, 4515, 4677, 4683	4031, 4054, 4081, 4133, 4163, 4285
\l_zrefclever_range_same_count_	\l_zrefclever_refbounds_last_
int 93,	re_seq 3770, 3986, 4235, 4243, 4310, 4318
3748, 3797, 4020, 4071, 4107, 4457,	\l_zrefclever_refbounds_last_
4501, 4518, 4524, 4569, 4590, 4703	seq 3770, 3978, 4062, 4097, 4142, 4178
\l_zrefclever_rangesep_tl	\l_zrefclever_refbounds_mid_rb_
..... 3755, 3918,	seq ... 3770, 3970, 4203, 4217, 4662
4231, 4241, 4306, 4316, 4680, 4690	\l_zrefclever_refbounds_mid_re_
\l_zrefclever_rangetopair_bool .	seq 3770, 3974, 4684, 4692
..... 3755, 3946, 4267	\l_zrefclever_refbounds_mid_seq
\l_zrefclever_ref_count_int 3770, 3966, 4094, 4172,
..... 3733, 3795,	4541, 4565, 4577, 4611, 4633, 4641
4048, 4126, 4196, 4454, 4487, 4536,	\l_zrefclever_reffont_tl
4559, 4571, 4605, 4627, 4636, 4656 3755, 3926, 4730,
\l_zrefclever_ref_decl_case_tl .	4742, 4771, 4782, 4820, 4877, 4892
..... 27, 829, 834, 835, 839, 842,	\l_zrefclever_reftype_override_
846, 850, 853, 898, 901, 903, 2165,	tl 36, 46, 2324, 2327
2175, 5022, 5026, 5065, 5070, 5075	\g_zrefclever_rf_opts_bool_
_zrefclever_ref_default: 4709,	maybe_type_specific_seq
4709, 4750, 4756, 4794, 4835, 4901	.. 52, 53, 570, 1440, 2560, 2827, 3355
\l_zrefclever_ref_gender_tl ...	\g_zrefclever_rf_opts_seq_
..... 28, 866, 872,	refbounds_seq 570, 1362, 2526, 2787, 3293
873, 877, 880, 885, 886, 905, 911,	\g_zrefclever_rf_opts_tl_maybe_
912, 2184, 2248, 4978, 4986, 4992, 5000	type_specific_seq 570, 1086, 2990
\l_zrefclever_ref_language_tl ..	\g_zrefclever_rf_opts_tl_not_
..... 24, 27, 55, 717, 721, 818,	type_specific_seq 570, 1061, 2609, 2966
823, 833, 851, 860, 870, 884, 893,	\g_zrefclever_rf_opts_tl_
902, 909, 2106, 2112, 2117, 2125,	reference_seq 570, 2478
2143, 2149, 2154, 3415, 3433, 3804,	\g_zrefclever_rf_opts_tl_type_
3808, 3812, 3901, 3905, 3909, 3913,	names_seq 570, 1323, 3253
3917, 3921, 3925, 3929, 3933, 3937,	\g_zrefclever_rf_opts_tl_
3941, 3945, 3949, 3953, 3957, 3961,	typesetup_seq 570, 2621
3965, 3969, 3973, 3977, 3981, 3985,	
4982, 4994, 5005, 5031, 5080, 5090	
\l_zrefclever_ref_property_tl ..	
..... 43, 48, 1551,	

\l_zrefclever_setup_language_tl	\l_zrefclever_tlistsep_tl
..... 565, 746, 774, 779, 789, 3755, 3809, 4403
794, 805, 810, 936, 988, 995, 1009,	\l_zrefclever_tpairsep_tl
1026, 1032, 1041, 1049, 1073, 1098, 3755, 3805, 4425
1107, 1130, 1136, 1144, 1151, 1165,	\l_zrefclever_type_count_int
1172, 1180, 1188, 1202, 1209, 1217,	. 93, 118, 3733, 3794, 4400, 4402,
1225, 1239, 1246, 1254, 1262, 1287,	4415, 4440, 4455, 4926, 4938, 5133
1294, 1303, 1311, 1341, 1351, 1374,	\l_zrefclever_type_first_label_-
1390, 1406, 1423, 1452, 1459, 1468,	tl
1476, 1490, 1497, 1506, 1514, 2870, 93,
2911, 2918, 2932, 2949, 2955, 2978,	115, 3736, 3789, 4007, 4258, 4269,
3002, 3010, 3033, 3039, 3047, 3054,	4273, 4301, 4352, 4369, 4373, 4449,
3068, 3075, 3083, 3091, 3105, 3112,	4483, 4793, 4799, 4805, 4809, 4822,
3120, 3128, 3142, 3149, 3157, 3165,	4853, 4867, 4871, 4879, 4894, 4907
3190, 3197, 3206, 3214, 3271, 3281,	\l_zrefclever_type_first_label_-
3315, 3340, 3367, 3375, 3388, 3396	type_tl
\l_zrefclever_setup_type_tl 93, 118, 3736, 3790,
565,	4009, 4262, 4450, 4485, 4914, 4957,
937, 979, 980, 1012, 1033, 1042,	4973, 4981, 4993, 4999, 5016, 5032,
1050, 1068, 1093, 1108, 1125, 1145,	5042, 5050, 5059, 5081, 5091, 5103
1152, 1160, 1181, 1189, 1197, 1218,	\l_zrefclever_type_first_-
1226, 1234, 1255, 1263, 1282, 1304,	refbounds_seq
1312, 1330, 1342, 1352, 1369, 1407, 3770, 4011, 4022, 4033, 4083,
1424, 1447, 1469, 1477, 1485, 1507,	4119, 4155, 4189, 4206, 4287, 4321,
1515, 2604, 2606, 2633, 2640, 2658,	4332, 4353, 4553, 4599, 4621, 4650,
2663, 2672, 2678, 2687, 2693, 2702,	4817, 4818, 4825, 4827, 4863, 4875,
2708, 2717, 2722, 2742, 2749, 2799,	4883, 4886, 4889, 4890, 4897, 4898
2813, 2837, 2846, 2855, 2869, 2902,	\l_zrefclever_type_first_-
2903, 2935, 2956, 2973, 2997, 3011,	refbounds_set_bool
3028, 3048, 3055, 3063, 3084, 3092, 3770,
3100, 3121, 3129, 3137, 3158, 3166,	3799, 4013, 4024, 4035, 4086, 4122,
3185, 3207, 3215, 3260, 3272, 3282,	4158, 4192, 4209, 4289, 4323,
3300, 3341, 3362, 3376, 3383, 3397	4330, 4459, 4556, 4602, 4624, 4653
\l_zrefclever_sort_decided_bool	\l_zrefclever_type_name_gender_-
..... 3469, 3552, 3566, 3576,	seq
3580, 3592, 3602, 3617, 3632, 3656 3742, 4983, 4985, 4988, 5003
_zrefclever_sort_default:nn	\l_zrefclever_type_name_-
..... 88, 3510, 3526, 3526	missing_bool
_zrefclever_sort_default_- 3742, 4839, 4910, 4917, 5039, 5099
different_types:nn	_zrefclever_type_name_setup:
..... 44, 86, 91, 3537, 3669, 3669 20, 22, 115, 4339, 4905, 4905
_zrefclever_sort_default_same_-	\l_zrefclever_type_name_tl
type:nn 115, 118,
..... 86, 88, 3536, 3539, 3539	3742, 4376, 4382, 4814, 4832, 4847,
_zrefclever_sort_labels:	4849, 4909, 4916, 5020, 5036, 5038,
..... 86-88, 92, 3424, 3473, 3473	5054, 5063, 5085, 5095, 5097, 5117
_zrefclever_sort_page:nn	\l_zrefclever_typeset_compress_-
..... 92, 3509, 3721, 3721	bool
\l_zrefclever_sort_prior_a_int 1629, 1632, 4468
..... 3470,	\l_zrefclever_typeset_labels_-
3671, 3677, 3678, 3684, 3694, 3702	seq
\l_zrefclever_sort_prior_b_int 92, 3730, 3785, 3819, 3821, 3827
..... 3470,	\l_zrefclever_typeset_last_bool
3672, 3679, 3680, 3687, 3695, 3703 93, 3730,
\l_zrefclever_tlastsep_tl	3816, 3817, 3824, 3849, 4412, 5132
..... 3755, 3813, 4432	\l_zrefclever_typeset_name_bool
 1578, 1585, 1590, 1595, 4341, 4357
	\l_zrefclever_typeset_queue_-
	curr_tl
 93,
	95, 113, 118, 3736, 3788, 4027,

4050, 4058, 4076, 4089, 4128,	\l__zrefclever_typesort_seq
4137, 4159, 4167, 4174, 4198, 4212,	. 44, 91, 1614, 1619, 1620, 1626, 3673
4229, 4239, 4256, 4281, 4304, 4314,	\l__zrefclever_verbose_testing_-
4343, 4350, 4360, 4393, 4409, 4420,	bool 3782, 4408
4426, 4433, 4447, 4448, 4537, 4560,	__zrefclever_zceref:nnn
4572, 4606, 4628, 4637, 4657, 4678, 27, 56, 3408, 3409
4688, 4931, 4953, 4964, 5127, 5131	__zrefclever_zceref:nnnn 83, 86, 3409
\l__zrefclever_typeset_queue_-	\l__zrefclever_zceref_labels_seq .
prev_tl . 93, 3736, 3787, 4404, 4446 86, 87, 3413,
\l__zrefclever_typeset_range_-	3444, 3449, 3453, 3478, 3481, 3786
bool . . . 1763, 1981, 1984, 3423, 4254	\l__zrefclever_zceref_note_tl . . .
\l__zrefclever_typeset_ref_bool 2273, 2276, 3429, 3436
. . 1577, 1584, 1589, 1594, 4341, 4347	\l__zrefclever_zceref_with_check_-
__zrefclever_typeset_refs:	bool 2280, 2297, 3419, 3440
. 92, 94, 95, 3427, 3783, 3783	__zrefclever_zcsetup:n
__zrefclever_typeset_refs_last_-	. 68, 2598, 2599, 2599, 2601, 5331,
of_type:	5355, 5361, 5369, 5391, 5410, 5434,
. . 99, 113, 115, 118, 3992, 3997, 3997	5438, 5439, 5448, 5459, 5493, 5544,
__zrefclever_typeset_refs_not_-	5569, 5581, 5591, 5606, 5626, 5649
last_of_type:	\l__zrefclever_zrefcheck_-
. . 93, 100, 113, 123, 3994, 4462, 4462	available_bool
\l__zrefclever_typeset_sort_bool	. . 2279, 2292, 2304, 2316, 3418, 3439
. 1605, 1608, 3422	