

The zref-clever package

Code documentation

gusbrs

<https://github.com/gusbrs/zref-clever>
<https://www.ctan.org/pkg/zref-clever>

Version v0.4.7 – 2024-09-30

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	zref setup	3
4	Plumbing	8
	4.1 Auxiliary	8
	4.2 Messages	9
	4.3 Data extraction	12
	4.4 Option infra	12
	4.5 Reference format	21
	4.6 Languages	25
	4.7 Language files	30
	4.8 Options	43
5	Configuration	69
	5.1 \zcsetup	69
	5.2 \zcRefTypeSetup	69
	5.3 \zcLanguageSetup	74
6	User interface	85
	6.1 \zcref	85
	6.2 \zcpageref	86
7	Sorting	87
8	Typesetting	93

9	Compatibility	128
9.1	appendix	128
9.2	appendices	130
9.3	memoir	131
9.4	amsmath	132
9.5	mathtools	134
9.6	breqn	135
9.7	listings	136
9.8	enumitem	136
9.9	subcaption	137
9.10	subfig	138
10	Language files	138
10.1	Localization guidelines	138
10.2	English	141
10.3	German	145
10.4	French	153
10.5	Portuguese	157
10.6	Spanish	162
10.7	Dutch	166
10.8	Italian	170
10.9	Russian	175
	Index	198

1 Initial setup

Start the DocStrip guards.

```
1 \*package)
```

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

```
2 \@@=zrefclever)
```

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the language files (which became the default input encoding in the 2018-04-01 release). Also, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (`ltxcmdhooks`), with implications to the hook we add to `\appendix` (by Phe-lype 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>). 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. Finally, since we followed the move to e-type expansion, to play safe we require the 2023-11-01 kernel or newer.

```

3 \def\zrefclever@required@kernel{2023-11-01}
4 \NeedsTeXFormat{LaTeX2e}[\zrefclever@required@kernel]
5 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
6 \IfFormatAtLeastTF{\zrefclever@required@kernel}
7   {}
8   {%
9     \PackageError{zref-clever}{LaTeX kernel too old}
10    {%
11      'zref-clever' requires a LaTeX kernel \zrefclever@required@kernel\space or newer.%
12    }%
13  }%

  Identify the package.
14 \ProvidesExplPackage {zref-clever} {2024-09-30} {0.4.7}
15   {Clever LaTeX cross-references based on zref}

```

2 Dependencies

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

```

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

```

3 zref setup

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

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

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

```

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

```

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

```

22 \zref@newprop { thecounter }
23 {
24   \cs_if_exist:cTF { c@ \l__zrefclever_current_counter_tl }
25   { \use:c { the \l__zrefclever_current_counter_tl } }
26   {
27     \cs_if_exist:cT { c@ \@currentcounter }
28     { \use:c { the \@currentcounter } }
29   }
30 }
31 \zref@addprop \ZREF@mainlist { thecounter }

```

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

```

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

```

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

```

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

```

```

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

```

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

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

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of `\newcounter`, `\@addtoreset`, `\counterwithin`, and related infrastructure). The canonical optional argument of `\newcounter` establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “super-counter”, “parent counter” etc.). This information is somewhat tricky to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\cl@<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 `\cl@<counter>`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresettters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\cl@<counter>` cannot possibly fully account for all of the

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

Recursively generate a *sequence* of “enclosing counters” and values, for a given $\langle counter \rangle$ and leave it in the input stream. These functions must be expandable, since they get called from `\zref@newprop` and are the ones 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:n {<counter>}
    \__zrefclever_get_enclosing_counters_value:n {<counter>}

64 \cs_new:Npn \__zrefclever_get_enclosing_counters:n #1
65   {
66     \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
67     {
68       { \__zrefclever_counter_reset_by:n {#1} }
69       \__zrefclever_get_enclosing_counters:e
70       { \__zrefclever_counter_reset_by:n {#1} }
71     }
72   }
73 \cs_new:Npn \__zrefclever_get_enclosing_counters_value:n #1
74   {
75     \cs_if_exist:cT { c@ \__zrefclever_counter_reset_by:n {#1} }
76     {
77       { \int_use:c { c@ \__zrefclever_counter_reset_by:n {#1} } }
78       \__zrefclever_get_enclosing_counters_value:e
79       { \__zrefclever_counter_reset_by:n {#1} }
80     }
81   }

```

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).

```

82 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters:n { e }
83 \cs_generate_variant:Nn \__zrefclever_get_enclosing_counters_value:n { e }

```

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

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

```

\__zrefclever_counter_reset_by:n {<counter>}

```

```

84 \cs_new:Npn \__zrefclever_counter_reset_by:n #1
85   {
86     \bool_if:nTF
87     { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
88     { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
89     {
90       \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
91       { \__zrefclever_counter_reset_by_aux:nn {#1} }
92     }
93   }
94 \cs_new:Npn \__zrefclever_counter_reset_by_aux:nn #1#2
95   {
96     \cs_if_exist:cT { c@ #2 }
97     {
98       \tl_if_empty:cF { cl@ #2 }
99       {
100         \tl_map_tokens:cn { cl@ #2 }
101         { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
102       }
103     }
104   }
105 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
106   {
107     \str_if_eq:nnT {#2} {#3}
108     { \tl_map_break:n { \seq_map_break:n {#1} } }
109   }

```

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

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

```

110 \zref@newprop { zc@enclval }
111   {
112     \__zrefclever_get_enclosing_counters_value:e
113     \l__zrefclever_current_counter_tl
114   }
115 \zref@addprop \ZREF@mainlist { zc@enclval }

```

The `zc@enclcnt` property is provided for the purpose of easing the debugging of counter reset chains, thus it is not added main property list by default.

```

116 \zref@newprop { zc@enclcnt }
117   { \__zrefclever_get_enclosing_counters:e \l__zrefclever_current_counter_tl }

```

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.

```

118 \int_new:N \g__zrefclever_page_format_int
119 \tl_new:N \g__zrefclever_prev_page_format_tl
120 \AddToHook { shipout / before }
121 {
122   \tl_if_eq:NNF \g__zrefclever_prev_page_format_tl \thepage
123   {
124     \int_gincr:N \g__zrefclever_page_format_int
125     \tl_gset_eq:NN \g__zrefclever_prev_page_format_tl \thepage
126   }
127 }
128 \zref@newprop* { zc@pgfmt } { \int_use:N \g__zrefclever_page_format_int }
129 \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.

```

130 \prg_new_conditional:Npnn \_zrefclever_if_package_loaded:n #1 { T , F , TF }
131 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
132 \prg_new_conditional:Npnn \_zrefclever_if_class_loaded:n #1 { T , F , TF }
133 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

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

`\l__zrefclever_tmpa_tl` Temporary scratch variables.

```

\l__zrefclever_tmpb_tl 134 \tl_new:N \l__zrefclever_tmpa_tl
\l__zrefclever_tmpa_seq 135 \tl_new:N \l__zrefclever_tmpb_tl
\g__zrefclever_tmpa_seq 136 \seq_new:N \l__zrefclever_tmpa_seq
\l__zrefclever_tmpa_bool 137 \seq_new:N \g__zrefclever_tmpa_seq
\l__zrefclever_tmpa_int 138 \bool_new:N \l__zrefclever_tmpa_bool
139 \int_new:N \l__zrefclever_tmpa_int

```


(End of definition for \l_zrefclever_tpa_t1 and others.)

4.2 Messages

```
140 \msg_new:nnn { zref-clever } { option-not-type-specific }
141 {
142   Option~'#1'~is~not~type-specific~\msg_line_context:~
143   Set~it~in~'\iow_char:N\zcLanguageSetup'~before~first~'type'~
144   switch~or~as~package~option.
145 }
146 \msg_new:nnn { zref-clever } { option-only-type-specific }
147 {
148   No~type~specified~for~option~'#1'~\msg_line_context:~
149   Set~it~after~'type'~switch.
150 }
151 \msg_new:nnn { zref-clever } { key-requires-value }
152 { The~'#1'~key~'#2'~requires~a~value~\msg_line_context:. }
153 \msg_new:nnn { zref-clever } { language-declared }
154 { Language~'#1'~is~already~declared~\msg_line_context:~Nothing~to~do. }
155 \msg_new:nnn { zref-clever } { unknown-language-alias }
156 {
157   Language~'#1'~is~unknown~\msg_line_context:~Can't~alias~to~it.~
158   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
159   '\iow_char:N\zcDeclareLanguageAlias'.
160 }
161 \msg_new:nnn { zref-clever } { unknown-language-setup }
162 {
163   Language~'#1'~is~unknown~\msg_line_context:~Can't~set~it~up.~
164   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
165   '\iow_char:N\zcDeclareLanguageAlias'.
166 }
167 \msg_new:nnn { zref-clever } { unknown-language-opt }
168 {
169   Language~'#1'~is~unknown~\msg_line_context:~
170   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
171   '\iow_char:N\zcDeclareLanguageAlias'.
172 }
173 \msg_new:nnn { zref-clever } { unknown-language-decl }
174 {
175   Can't~set~declension~'#1'~for~unknown~language~'#2'~\msg_line_context:~
176   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
177   '\iow_char:N\zcDeclareLanguageAlias'.
178 }
179 \msg_new:nnn { zref-clever } { language-no-decl-ref }
180 {
181   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:~
182   Nothing~to~do~with~option~'d=#2'.
183 }
184 \msg_new:nnn { zref-clever } { language-no-gender }
185 {
186   Language~'#1'~has~no~declared~gender~\msg_line_context:~
187   Nothing~to~do~with~option~'#2=#3'.
188 }
189 \msg_new:nnn { zref-clever } { language-no-decl-setup }
```

```

190 {
191   Language~'#1'~has~no~declared~declension~cases~\msg_line_context:..~
192   Nothing~to~do~with~option~'case=#2'.
193 }
194 \msg_new:nnn { zref-clever } { unknown-decl-case }
195 {
196   Declension~case~'#1'~unknown~for~language~'#2'~\msg_line_context:..~
197   Using~default~declension~case.
198 }
199 \msg_new:nnn { zref-clever } { nudge-multitype }
200 {
201   Reference~with~multiple~types~\msg_line_context:..~
202   You~may~wish~to~separate~them~or~review~language~around~it.
203 }
204 \msg_new:nnn { zref-clever } { nudge-comptosing }
205 {
206   Multiple~labels~have~been~compressed~into~singular~type~name~
207   for~type~'#1'~\msg_line_context:..
208 }
209 \msg_new:nnn { zref-clever } { nudge-plural-when-sg }
210 {
211   Option~'sg'~signals~that~a~singular~type~name~was~expected~
212   \msg_line_context:..~But~type~'#1'~has~plural~type~name.
213 }
214 \msg_new:nnn { zref-clever } { gender-not-declared }
215 { Language~'#1'~has~no~'#2'~gender~declared~\msg_line_context:.. }
216 \msg_new:nnn { zref-clever } { nudge-gender-mismatch }
217 {
218   Gender~mismatch~for~type~'#1'~\msg_line_context:..~
219   You've~specified~'g=#2'~but~type~name~is~'#3'~for~language~'#4'.
220 }
221 \msg_new:nnn { zref-clever } { nudge-gender-not-declared-for-type }
222 {
223   You've~specified~'g=#1'~\msg_line_context:..~
224   But~gender~for~type~'#2'~is~not~declared~for~language~'#3'.
225 }
226 \msg_new:nnn { zref-clever } { nudgeif-unknown-value }
227 { Unknown~value~'#1'~for~'nudgeif'~option~\msg_line_context:.. }
228 \msg_new:nnn { zref-clever } { option-document-only }
229 { Option~'#1'~is~only~available~after~\iow_char:N\begin\{document\}. }
230 \msg_new:nnn { zref-clever } { langfile-loaded }
231 { Loaded~'#1'~language~file. }
232 \msg_new:nnn { zref-clever } { zref-property-undefined }
233 {
234   Option~'ref=#1'~requested~\msg_line_context:..~
235   But~the~property~'#1'~is~not~declared,~falling-back~to~'default'.
236 }
237 \msg_new:nnn { zref-clever } { endrange-property-undefined }
238 {
239   Option~'endrange=#1'~requested~\msg_line_context:..~
240   But~the~property~'#1'~is~not~declared,~'endrange'~not~set.
241 }
242 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
243 {

```

```

244 Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:~
245 To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
246 '\iow_char:N\zcref'.
247 }
248 \msg_new:nnn { zref-clever } { missing-hyperref }
249 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
250 \msg_new:nnn { zref-clever } { option-preamble-only }
251 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:. }
252 \msg_new:nnn { zref-clever } { unknown-compat-module }
253 {
254   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
255   Nothing~to~do.
256 }
257 \msg_new:nnn { zref-clever } { refbounds-must-be-four }
258 {
259   The~value~of~option~'#1'~must~be~a~comma~separated~list~
260   of~four~items.~We~received~'#2'~items~\msg_line_context:~
261   Option~not~set.
262 }
263 \msg_new:nnn { zref-clever } { missing-zref-check }
264 {
265   Option~'check'~requested~\msg_line_context:~
266   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
267 }
268 \msg_new:nnn { zref-clever } { zref-check-too-old }
269 {
270   Option~'check'~requested~\msg_line_context:~
271   But~'zref-check'~newer~than~'#1'~is~required,~can't~run~the~checks.
272 }
273 \msg_new:nnn { zref-clever } { missing-type }
274 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
275 \msg_new:nnn { zref-clever } { missing-property }
276 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:. }
277 \msg_new:nnn { zref-clever } { missing-name }
278 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:. }
279 \msg_new:nnn { zref-clever } { single-element-range }
280 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }
281 \msg_new:nnn { zref-clever } { compat-package }
282 { Loaded~support~for~'#1'~package. }
283 \msg_new:nnn { zref-clever } { compat-class }
284 { Loaded~support~for~'#1'~documentclass. }
285 \msg_new:nnn { zref-clever } { option-deprecated }
286 {
287   Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
288   Use~'#2'~instead.
289 }
290 \msg_new:nnn { zref-clever } { load-time-options }
291 {
292   'zref-clever'~does~not~accept~load-time~options.~
293   To~configure~package~options,~use~'\iow_char:N\zcsetup'.
294 }

```

4.3 Data extraction

`_zrefclever_extract_default:Nnn` Extract property $\langle prop \rangle$ from $\langle label \rangle$ and sets variable $\langle t1 var \rangle$ 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 $\langle t1 var \rangle$ with $\langle default \rangle$.

```

    \_zrefclever_extract_default:Nnnn {\langle t1 var \rangle}
      {\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

295 \cs_new_protected:Npn \_zrefclever_extract_default:Nnnn #1#2#3#4
296   {
297     \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
298       { \zref@extractdefault {#2} {#3} {#4} }
299   }
300 \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}

301 \cs_new:Npn \_zrefclever_extract_unexp:nnn #1#2#3
302   {
303     \exp_args:NNo \exp_args:No
304       \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
305   }
306 \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{\langle label \rangle}{\langle prop \rangle}{\langle default \rangle}

307 \cs_new:Npn \_zrefclever_extract:nnn #1#2#3
308   { \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}

309 \cs_new:Npn \__zrefclever_opt_varname_general:nn #1#2
310 { 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}

311 \cs_new:Npn \__zrefclever_opt_varname_type:nnn #1#2#3
312 { l__zrefclever_opt_type_ #1 _ #2 _ #3 }
313 \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 `\option` for `\lang` (for general language options, those set with `\zcDeclareLanguage`). The “`lang_unknown`” branch should be guarded against, such as we normally should not get there, but this function *must* return some valid csname. The random part is there so that, in the circumstance this could not be avoided, we (hopefully) don't retrieve the value for an “unknown language” inadvertently.

```

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

314 \cs_new:Npn \__zrefclever_opt_varname_language:nnn #1#2#3
315 {
316   \__zrefclever_language_if_declared:nTF {#1}
317   {
318     g__zrefclever_opt_language_
319     \tl_use:c { \__zrefclever_language_varname:n {#1} }
320     _ #2 _ #3
321   }
322   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
323 }
324 \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 `\option` for `\lang`.

```

    \__zrefclever_opt_varname_lang_default:nnn <{lang}> <{option}> <{data type}>
325 \cs_new:Npn \__zrefclever_opt_varname_lang_default:nnn #1#2#3
326 {
327   \__zrefclever_language_if_declared:nTF {#1}
328   {
329     g__zrefclever_opt_lang_
330     \tl_use:c { \__zrefclever_language_varname:n {#1} }
331     _default_ #2 _ #3
332   }
333   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #3 }
334 }
335 \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 <option> for <lang> and <ref type>.

```

    \__zrefclever_opt_varname_lang_type:nnnn <{lang}> <{ref type}>
    <{option}> <{data type}>
336 \cs_new:Npn \__zrefclever_opt_varname_lang_type:nnnn #1#2#3#4
337 {
338   \__zrefclever_language_if_declared:nTF {#1}
339   {
340     g__zrefclever_opt_lang_
341     \tl_use:c { \__zrefclever_language_varname:n {#1} }
342     _type_ #2 _ #3 _ #4
343   }
344   { g__zrefclever_opt_lang_unknown_ \int_rand:n { 1000000 } _ #4 }
345 }
346 \cs_generate_variant:Nn
347   \__zrefclever_opt_varname_lang_type:nnnn { eenn , eeen }

```

(End of definition for __zrefclever_opt_varname_lang_type:nnnn.)

__zrefclever_opt_varname_fallback:nn

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

```

    \__zrefclever_opt_varname_fallback:nn <{option}> <{data type}>
348 \cs_new:Npn \__zrefclever_opt_varname_fallback:nn #1#2
349 { 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>}
```

```
350 \cs_new:Npn \_zrefclever_opt_var_set_bool:n #1
```

```
351   { \cs_to_str:N #1 _is_set_bool }
```

(End of definition for _zrefclever_opt_var_set_bool:n)

```
    \_zrefclever_opt_tl_set:N {<option tl>} {<value>}
```

```
    \_zrefclever_opt_tl_clear:N {<option tl>}
```

```
    \_zrefclever_opt_tl_gset:N {<option tl>} {<value>}
```

```
    \_zrefclever_opt_tl_gclear:N {<option tl>}
```

```
352 \cs_new_protected:Npn \_zrefclever_opt_tl_set:Nn #1#2
```

```
353   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_set:Nn #1 {#2}
```

```
357    \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
```

```
358      { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
359      { \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
360   }
```

```
361 \cs_generate_variant:Nn \_zrefclever_opt_tl_set:Nn { cn }
```

```
362 \cs_new_protected:Npn \_zrefclever_opt_tl_clear:N #1
```

```
363   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_clear:N #1
```

```
367    \bool_if_exist:cF { \_zrefclever_opt_var_set_bool:n {#1} }
```

```
368      { \bool_new:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
369      { \bool_set_true:c { \_zrefclever_opt_var_set_bool:n {#1} } }
```

```
370   }
```

```
371 \cs_generate_variant:Nn \_zrefclever_opt_tl_clear:N { c }
```

```
372 \cs_new_protected:Npn \_zrefclever_opt_tl_gset:Nn #1#2
```

```
373   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_gset:Nn #1 {#2}
```

```
377   }
```

```
378 \cs_generate_variant:Nn \_zrefclever_opt_tl_gset:Nn { cn }
```

```
379 \cs_new_protected:Npn \_zrefclever_opt_tl_gclear:N #1
```

```
380   {
```

```
    \tl_if_exist:NF #1
```

```
      { \tl_new:N #1 }
```

```
    \tl_gclear:N #1
```

```
384   }
```

```
385 \cs_generate_variant:Nn \_zrefclever_opt_tl_gclear:N { c }
```

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

```
\_zrefclever_opt_tl_unset:N Unset <option tl>.
```

```
    \_zrefclever_opt_tl_unset:N {<option tl>}
```

```
386 \cs_new_protected:Npn \_zrefclever_opt_tl_unset:N #1
```

```
387   {
```

```
388     \tl_if_exist:NT #1
```

```

389     {
390       \tl_clear:N #1 % ?
391       \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
392         { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
393         { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
394     }
395 }
396 \cs_generate_variant:Nn \__zrefclever_opt_tl_unset:N { c }

```

(End of definition for __zrefclever_opt_tl_unset:N.)

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

```

\__zrefclever_opt_tl_if_set:N(TF) {<option tl>} {<true>} {<false>}
397 \prg_new_conditional:Npnn \__zrefclever_opt_tl_if_set:N #1 { F , TF }
398 {
399   \tl_if_exist:NTF #1
400   {
401     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
402     {
403       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
404       { \prg_return_true: }
405       { \prg_return_false: }
406     }
407     { \prg_return_true: }
408   }
409   { \prg_return_false: }
410 }

```

(End of definition for __zrefclever_opt_tl_if_set:NTF.)

```

\__zrefclever_opt_tl_gset_if_new:Nn \__zrefclever_opt_tl_gset_if_new:Nn {<option tl>} {<value>}
\_zrefclever_opt_tl_gclear_if_new:N \__zrefclever_opt_tl_gclear_if_new:N {<option tl>}
411 \cs_new_protected:Npn \__zrefclever_opt_tl_gset_if_new:Nn #1#2
412 {
413   \__zrefclever_opt_tl_if_set:NF #1
414   {
415     \tl_if_exist:NF #1
416     { \tl_new:N #1 }
417     \tl_gset:Nn #1 {#2}
418   }
419 }
420 \cs_generate_variant:Nn \__zrefclever_opt_tl_gset_if_new:Nn { cn }
421 \cs_new_protected:Npn \__zrefclever_opt_tl_gclear_if_new:N #1
422 {
423   \__zrefclever_opt_tl_if_set:NF #1
424   {
425     \tl_if_exist:NF #1
426     { \tl_new:N #1 }
427     \tl_gclear:N #1
428   }

```



```

429 }
430 \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>}
431 \prg_new_protected_conditional:Npnn \__zrefclever_opt_tl_get:NN #1#2 { F }
432 {
433   \__zrefclever_opt_tl_if_set:NTF #1
434   {
435     \tl_set_eq:NN #2 #1
436     \prg_return_true:
437   }
438   { \prg_return_false: }
439 }
440 \prg_generate_conditional_variant:Nnn
441 \__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_set_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_gset_clist_split:Nn \__zrefclever_opt_seq_gset_clist_split:Nn {<option seq>} {<value>}
\__zrefclever_opt_seq_set_eq:NN \__zrefclever_opt_seq_set_eq:NN {<option seq>} {<seq var>}
\__zrefclever_opt_seq_gset_eq:NN \__zrefclever_opt_seq_gset_eq:NN {<option seq>} {<seq var>}
442 \cs_new_protected:Npn \__zrefclever_opt_seq_set_clist_split:Nn #1#2
443 { \seq_set_split:Nnn #1 { , } {#2} }
444 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_clist_split:Nn #1#2
445 { \seq_gset_split:Nnn #1 { , } {#2} }
446 \cs_new_protected:Npn \__zrefclever_opt_seq_set_eq:NN #1#2
447 {
448   \seq_if_exist:NF #1
449   { \seq_new:N #1 }
450   \seq_set_eq:NN #1 #2
451   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
452   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
453   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
454 }
455 \cs_generate_variant:Nn \__zrefclever_opt_seq_set_eq:NN { cN }
456 \cs_new_protected:Npn \__zrefclever_opt_seq_gset_eq:NN #1#2
457 {
458   \seq_if_exist:NF #1
459   { \seq_new:N #1 }
460   \seq_gset_eq:NN #1 #2
461 }
462 \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 <option seq>.

```

\__zrefclever_opt_seq_unset:N {<option seq>}

```

```

463 \cs_new_protected:Npn \__zrefclever_opt_seq_unset:N #1
464 {
465   \seq_if_exist:NT #1
466   {
467     \seq_clear:N #1 % ?
468     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
469     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
470     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
471   }
472 }
473 \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) {<option seq>} {<true>} {<false>}
474 \prg_new_conditional:Npnn \__zrefclever_opt_seq_if_set:N #1 { F , TF }
475 {
476   \seq_if_exist:NTF #1
477   {
478     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
479     {
480       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
481       { \prg_return_true: }
482       { \prg_return_false: }
483     }
484     { \prg_return_true: }
485   }
486   { \prg_return_false: }
487 }
488 \prg_generate_conditional_variant:Nnn
489 \__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>}
490 \prg_new_protected_conditional:Npnn \__zrefclever_opt_seq_get:NN #1#2 { F }
491 {
492   \__zrefclever_opt_seq_if_set:NTF #1
493   {
494     \seq_set_eq:NN #2 #1
495     \prg_return_true:
496   }
497   { \prg_return_false: }
498 }
499 \prg_generate_conditional_variant:Nnn
500 \__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>}

```

```

501 \cs_new_protected:Npn \__zrefclever_opt_bool_unset:N #1
502 {
503   \bool_if_exist:NT #1
504   {
505     % \bool_set_false:N #1 % ?
506     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
507     { \bool_set_false:c { \__zrefclever_opt_var_set_bool:n {#1} } }
508     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
509   }
510 }
511 \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>}
512 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if_set:N #1 { F , TF }
513 {
514   \bool_if_exist:NTF #1
515   {
516     \bool_if_exist:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
517     {
518       \bool_if:cTF { \__zrefclever_opt_var_set_bool:n {#1} }
519       { \prg_return_true: }
520       { \prg_return_false: }
521     }
522     { \prg_return_true: }
523   }
524   { \prg_return_false: }
525 }
526 \prg_generate_conditional_variant:Nnn
527 \__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>}
528 \cs_new_protected:Npn \__zrefclever_opt_bool_set_true:N #1
529 {
530   \bool_if_exist:NF #1
531   { \bool_new:N #1 }
532   \bool_set_true:N #1
533   \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
534   { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
535   \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
536 }
537 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_true:N { c }
538 \cs_new_protected:Npn \__zrefclever_opt_bool_set_false:N #1
539 {
540   \bool_if_exist:NF #1
541   { \bool_new:N #1 }

```

```

542     \bool_set_false:N #1
543     \bool_if_exist:cF { \__zrefclever_opt_var_set_bool:n {#1} }
544     { \bool_new:c { \__zrefclever_opt_var_set_bool:n {#1} } }
545     \bool_set_true:c { \__zrefclever_opt_var_set_bool:n {#1} }
546   }
547 \cs_generate_variant:Nn \__zrefclever_opt_bool_set_false:N { c }
548 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_true:N #1
549   {
550     \bool_if_exist:NF #1
551     { \bool_new:N #1 }
552     \bool_gset_true:N #1
553   }
554 \cs_generate_variant:Nn \__zrefclever_opt_bool_gset_true:N { c }
555 \cs_new_protected:Npn \__zrefclever_opt_bool_gset_false:N #1
556   {
557     \bool_if_exist:NF #1
558     { \bool_new:N #1 }
559     \bool_gset_false:N #1
560   }
561 \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:NTF      \__zrefclever_opt_bool_get:NN(TF) {<option bool to get>} {<bool var to set>}
                                   {<true>} {<false>}

```

```

562 \prg_new_protected_conditional:Npnn \__zrefclever_opt_bool_get:NN #1#2 { F }
563   {
564     \__zrefclever_opt_bool_if_set:NTF #1
565     {
566       \bool_set_eq:NN #2 #1
567       \prg_return_true:
568     }
569     { \prg_return_false: }
570   }
571 \prg_generate_conditional_variant:Nnn
572   \__zrefclever_opt_bool_get:NN { cN } { F }

```

(End of definition for __zrefclever_opt_bool_get:NTF.)

```

\__zrefclever_opt_bool_if:NTF      \__zrefclever_opt_bool_if:N(TF) {<option bool>} {<true>} {<false>}
573 \prg_new_conditional:Npnn \__zrefclever_opt_bool_if:N #1 { T , F , TF }
574   {
575     \__zrefclever_opt_bool_if_set:NTF #1
576     { \bool_if:NTF #1 { \prg_return_true: } { \prg_return_false: } }
577     { \prg_return_false: }
578   }
579 \prg_generate_conditional_variant:Nnn
580   \__zrefclever_opt_bool_if:N { c } { T , F , TF }

```

(End of definition for __zrefclever_opt_bool_if:NTF.)

4.5 Reference format

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

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

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

```

\l__zrefclever_setup_type_tl Store “current” type, language, and declension cases in different places for type-
  \l_zrefclever_setup_language_tl specific and language-specific options handling, notably in \__zrefclever_provide_
  \l_zrefclever_lang_decl_case_tl langfile:n, \zcRefTypeSetup, and \zcLanguageSetup, but also for language specific
\l_zrefclever_lang_declension_seq options retrieval.
  \l_zrefclever_lang_gender_seq
581 \tl_new:N \l__zrefclever_setup_type_tl
582 \tl_new:N \l__zrefclever_setup_language_tl
583 \tl_new:N \l__zrefclever_lang_decl_case_tl
584 \seq_new:N \l__zrefclever_lang_declension_seq
585 \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
efclever_rf_opts_tl_maybe_type_specific_seq
\g_zrefclever_rf_opts_seq_refbounds_seq
clever_rf_opts_bool_maybe_type_specific_seq
\g_zrefclever_rf_opts_tl_type_names_seq
\g_zrefclever_rf_opts_tl_typesetup_seq
\g_zrefclever_rf_opts_tl_reference_seq

```

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

```

586 \seq_new:N \g_zrefclever_rf_opts_tl_not_type_specific_seq
587 \seq_gset_from_clist:Nn
588   \g_zrefclever_rf_opts_tl_not_type_specific_seq
589   {
590     tpairsep ,
591     tlistsep ,
592     tlastsep ,
593     notesep ,
594   }
595 \seq_new:N \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
596 \seq_gset_from_clist:Nn
597   \g_zrefclever_rf_opts_tl_maybe_type_specific_seq
598   {
599     namesep ,
600     pairsep ,
601     listsep ,
602     lastsep ,
603     rangesep ,
604     namefont ,
605     reffont ,
606   }
607 \seq_new:N \g_zrefclever_rf_opts_seq_refbounds_seq
608 \seq_gset_from_clist:Nn
609   \g_zrefclever_rf_opts_seq_refbounds_seq
610   {
611     refbounds-first ,
612     refbounds-first-sg ,
613     refbounds-first-pb ,
614     refbounds-first-rb ,
615     refbounds-mid ,
616     refbounds-mid-rb ,
617     refbounds-mid-re ,
618     refbounds-last ,
619     refbounds-last-pe ,
620     refbounds-last-re ,
621   }
622 \seq_new:N \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
623 \seq_gset_from_clist:Nn
624   \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
625   {
626     cap ,
627     abbrev ,
628     rangetopair ,
629   }

```

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:.

```
630 \seq_new:N \g__zrefclever_rf_opts_tl_type_names_seq
631 \seq_gset_from_clist:Nn
632 \g__zrefclever_rf_opts_tl_type_names_seq
633 {
634   Name-sg ,
635   name-sg ,
636   Name-pl ,
637   name-pl ,
638   Name-sg-ab ,
639   name-sg-ab ,
640   Name-pl-ab ,
641   name-pl-ab ,
642 }
```

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

```
643 \seq_new:N \g__zrefclever_rf_opts_tl_typesetup_seq
644 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_typesetup_seq
645 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
646 \g__zrefclever_rf_opts_tl_type_names_seq
647 \seq_new:N \g__zrefclever_rf_opts_tl_reference_seq
648 \seq_gconcat:NNN \g__zrefclever_rf_opts_tl_reference_seq
649 \g__zrefclever_rf_opts_tl_not_type_specific_seq
650 \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.

```
651 \clist_map_inline:nn
652 {
653   reference ,
654   typesetup ,
655   langsetup ,
656   langfile ,
657 }
658 {
659   \keys_define:nn { zref-clever/ #1 }
660   {
661     +refbounds-first .meta:n =
662     {
663       refbounds-first = {##1} ,
664       refbounds-first-sg = {##1} ,
665       refbounds-first-pb = {##1} ,
666       refbounds-first-rb = {##1} ,
667     } ,
668     +refbounds-mid .meta:n =
669     {
670       refbounds-mid = {##1} ,
671       refbounds-mid-rb = {##1} ,
672       refbounds-mid-re = {##1} ,
673     } ,
674     +refbounds-last .meta:n =
675     {
676       refbounds-last = {##1} ,
```

```

677         refbounds-last-pe = {##1} ,
678         refbounds-last-re = {##1} ,
679     } ,
680 +refbounds-rb .meta:n =
681     {
682         refbounds-first-rb = {##1} ,
683         refbounds-mid-rb = {##1} ,
684     } ,
685 +refbounds-re .meta:n =
686     {
687         refbounds-mid-re = {##1} ,
688         refbounds-last-re = {##1} ,
689     } ,
690 +refbounds .meta:n =
691     {
692         +refbounds-first = {##1} ,
693         +refbounds-mid = {##1} ,
694         +refbounds-last = {##1} ,
695     } ,
696     refbounds .meta:n = { +refbounds = {##1} } ,
697 }
698 }
699 \clist_map_inline:nn
700 {
701     reference ,
702     typesetup ,
703 }
704 {
705     \keys_define:nn { zref-clever/ #1 }
706     {
707         +refbounds-first .default:o = \c_novalue_tl ,
708         +refbounds-mid .default:o = \c_novalue_tl ,
709         +refbounds-last .default:o = \c_novalue_tl ,
710         +refbounds-rb .default:o = \c_novalue_tl ,
711         +refbounds-re .default:o = \c_novalue_tl ,
712         +refbounds .default:o = \c_novalue_tl ,
713         refbounds .default:o = \c_novalue_tl ,
714     }
715 }
716 \clist_map_inline:nn
717 {
718     langsetup ,
719     langfile ,
720 }
721 {
722     \keys_define:nn { zref-clever/ #1 }
723     {
724         +refbounds-first .value_required:n = true ,
725         +refbounds-mid .value_required:n = true ,
726         +refbounds-last .value_required:n = true ,
727         +refbounds-rb .value_required:n = true ,
728         +refbounds-re .value_required:n = true ,
729         +refbounds .value_required:n = true ,
730         refbounds .value_required:n = true ,

```



```

731     }
732 }

```

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.

```

733 \tl_new:N \l__zrefclever_ref_language_tl
734 \tl_new:N \l__zrefclever_current_language_tl
735 \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`.

```

736 \tl_new:N \l_zrefclever_ref_language_tl
737 \tl_set:Nn \l_zrefclever_ref_language_tl { \l__zrefclever_ref_language_tl }

```

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

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

```

\__zrefclever_language_varname:n {<language>}

```

```

738 \cs_new:Npn \__zrefclever_language_varname:n #1
739 { 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`.

```

740 \cs_set_eq:NN \zrefclever_language_varname:n
741 \__zrefclever_language_varname:n

```

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

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

```

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

```

```

742 \prg_new_conditional:Npnn \_zrefclever_language_if_declared:n #1 { T , F , TF }
743 {
744   \tl_if_exist:cTF { \__zrefclever_language_varname:n {#1} }
745   { \prg_return_true: }
746   { \prg_return_false: }
747 }
748 \prg_generate_conditional_variant:Nnn
749 \_zrefclever_language_if_declared:n { e } { 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`.

```
750 \prg_set_eq_conditional:NNn \zrefclever_language_if_declared:n
751 \__zrefclever_language_if_declared:n { TF }
```

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

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

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

752 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
753 {
754   \group_begin:
755   \tl_if_empty:nF {#2}
756   {
757     \__zrefclever_language_if_declared:nTF {#2}
758     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
759     {
760       \tl_new:c { \__zrefclever_language_varname:n {#2} }
761       \tl_gset:cn { \__zrefclever_language_varname:n {#2} } {#2}
762       \tl_set:Nn \l__zrefclever_setup_language_tl {#2}
763       \keys_set:nn { zref-clever/declarelang } {#1}
764     }
765   }
766   \group_end:
767 }
768 \@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>}

769 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
770 {
771   \tl_if_empty:nF {#1}
772   {
```

```

773     \_zrefclever_language_if_declared:nTF {#2}
774     {
775         \tl_new:c { \_zrefclever_language_varname:n {#1} }
776         \tl_gset:ce { \_zrefclever_language_varname:n {#1} }
777             { \tl_use:c { \_zrefclever_language_varname:n {#2} } }
778     }
779     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
780 }
781 }
782 \@onlypreamble \zcDeclareLanguageAlias

```

(End of definition for \zcDeclareLanguageAlias.)

```

783 \keys_define:nn { zref-clever/declarelang }
784 {
785     declension .code:n =
786     {
787         \seq_new:c
788         {
789             \_zrefclever_opt_varname_language:enn
790             { \l__zrefclever_setup_language_tl } { declension } { seq }
791         }
792         \seq_gset_from_clist:cn
793         {
794             \_zrefclever_opt_varname_language:enn
795             { \l__zrefclever_setup_language_tl } { declension } { seq }
796         }
797         {#1}
798     } ,
799     declension .value_required:n = true ,
800     gender .code:n =
801     {
802         \seq_new:c
803         {
804             \_zrefclever_opt_varname_language:enn
805             { \l__zrefclever_setup_language_tl } { gender } { seq }
806         }
807         \seq_gset_from_clist:cn
808         {
809             \_zrefclever_opt_varname_language:enn
810             { \l__zrefclever_setup_language_tl } { gender } { seq }
811         }
812         {#1}
813     } ,
814     gender .value_required:n = true ,
815     allcaps .choices:nn =
816     { true , false }
817     {
818         \bool_new:c
819         {
820             \_zrefclever_opt_varname_language:enn
821             { \l__zrefclever_setup_language_tl } { allcaps } { bool }
822         }
823         \use:c { bool_gset_ \l_keys_choice_tl :c }
824         {

```

```

825         \_zrefclever_opt_varname_language:enn
826         { \l__zrefclever_setup_language_tl } { allcaps } { bool }
827     }
828 },
829 allcaps .default:n = true ,
830 }

```

`_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.

```

831 \cs_new_protected:Npn \_zrefclever_process_language_settings:
832 {
833     \_zrefclever_language_if_declared:eTF
834     { \l__zrefclever_ref_language_tl }
835     {

```

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.

```

836     \_zrefclever_opt_seq_get:cNF
837     {
838         \_zrefclever_opt_varname_language:enn
839         { \l__zrefclever_ref_language_tl } { declension } { seq }
840     }
841     \l__zrefclever_lang_declension_seq
842     { \seq_clear:N \l__zrefclever_lang_declension_seq }
843     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
844     {
845         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
846         {
847             \msg_warning:nnee { zref-clever }
848             { language-no-decl-ref }
849             { \l__zrefclever_ref_language_tl }
850             { \l__zrefclever_ref_decl_case_tl }
851             \tl_clear:N \l__zrefclever_ref_decl_case_tl
852         }
853     }
854     {
855         \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
856         {
857             \seq_get_left:NN \l__zrefclever_lang_declension_seq
858             \l__zrefclever_ref_decl_case_tl
859         }
860         {
861             \seq_if_in:NVF \l__zrefclever_lang_declension_seq

```

```

862         \l__zrefclever_ref_decl_case_tl
863         {
864             \msg_warning:nnee { zref-clever }
865             { unknown-decl-case }
866             { \l__zrefclever_ref_decl_case_tl }
867             { \l__zrefclever_ref_language_tl }
868             \seq_get_left:NN \l__zrefclever_lang_declension_seq
869             \l__zrefclever_ref_decl_case_tl
870         }
871     }
872 }

```

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.

```

873     \__zrefclever_opt_seq_get:cNF
874     {
875         \__zrefclever_opt_varname_language:enn
876         { \l__zrefclever_ref_language_tl } { gender } { seq }
877     }
878     \l__zrefclever_lang_gender_seq
879     { \seq_clear:N \l__zrefclever_lang_gender_seq }
880     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
881     {
882         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
883         {
884             \msg_warning:nneee { zref-clever }
885             { language-no-gender }
886             { \l__zrefclever_ref_language_tl }
887             { g }
888             { \l__zrefclever_ref_gender_tl }
889             \tl_clear:N \l__zrefclever_ref_gender_tl
890         }
891     }
892     {
893         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
894         {
895             \seq_if_in:NVF \l__zrefclever_lang_gender_seq
896             \l__zrefclever_ref_gender_tl
897             {
898                 \msg_warning:nnee { zref-clever }
899                 { gender-not-declared }
900                 { \l__zrefclever_ref_language_tl }
901                 { \l__zrefclever_ref_gender_tl }
902                 \tl_clear:N \l__zrefclever_ref_gender_tl
903             }
904         }
905     }

```

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

```

906     \__zrefclever_opt_bool_if:cT
907     {
908         \__zrefclever_opt_varname_language:enn
909         { \l__zrefclever_ref_language_tl } { allcaps } { bool }

```

```

910     }
911     { \keys_set:nn { zref-clever/reference } { cap = true } }
912   }
913   {

```

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

```

914     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
915     {
916       \msg_warning:nnee { zref-clever } { unknown-language-decl }
917       { \l__zrefclever_ref_decl_case_tl }
918       { \l__zrefclever_ref_language_tl }
919       \tl_clear:N \l__zrefclever_ref_decl_case_tl
920     }
921     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
922     {
923       \msg_warning:nneee { zref-clever }
924       { language-no-gender }
925       { \l__zrefclever_ref_language_tl }
926       { g }
927       { \l__zrefclever_ref_gender_tl }
928       \tl_clear:N \l__zrefclever_ref_gender_tl
929     }
930   }
931 }

```

(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 `.lbf` 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.

```
932 \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>}
```

```
933 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
934 {
935   \group_begin:
936   \@bsphack
937   \__zrefclever_language_if_declared:nT {#1}
938   {
939     \seq_if_in:NeF
940     \g_zrefclever_loaded_langfiles_seq
941     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
942     {
943       \exp_args:Ne \file_get:nnNTF
944       {
945         zref-clever-
946         \tl_use:c { \__zrefclever_language_varname:n {#1} }
947         .lang
948       }
949       { \ExplSyntaxOn }
950       \l__zrefclever_tmpa_tl
951       {
952         \tl_set:Nn \l__zrefclever_setup_language_tl {#1}
953         \tl_clear:N \l__zrefclever_setup_type_tl
954         \__zrefclever_opt_seq_get:cNF
955         {
956           \__zrefclever_opt_varname_language:nnn
957           {#1} { declension } { seq }
958         }
959         \l__zrefclever_lang_declension_seq
960         { \seq_clear:N \l__zrefclever_lang_declension_seq }
961         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
962         { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
```

```

963         {
964             \seq_get_left:NN \l__zrefclever_lang_declension_seq
965             \l__zrefclever_lang_decl_case_tl
966         }
967     \__zrefclever_opt_seq_get:cNF
968     {
969         \__zrefclever_opt_varname_language:nnn
970         {#1} { gender } { seq }
971     }
972     \l__zrefclever_lang_gender_seq
973     { \seq_clear:N \l__zrefclever_lang_gender_seq }
974     \keys_set:nV { zref-clever/langfile } \l__zrefclever_tmpa_tl
975     \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
976     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
977     \msg_info:nne { zref-clever } { langfile-loaded }
978     { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
979     }
980     {

```

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.

```

981         \seq_gput_right:Ne \g__zrefclever_loaded_langfiles_seq
982         { \tl_use:c { \__zrefclever_language_varname:n {#1} } }
983     }
984 }
985 }
986 \@esphack
987 \group_end:
988 }
989 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { e }

```

(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.

```

990 \keys_define:nn { zref-clever/langfile }
991 {
992     type .code:n =
993     {
994         \tl_if_empty:nTF {#1}
995         { \tl_clear:N \l__zrefclever_setup_type_tl }
996         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
997     } ,
998
999     case .code:n =
1000     {
1001         \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1002         {
1003             \msg_info:nnee { zref-clever } { language-no-decl-setup }
1004             { \l__zrefclever_setup_language_tl } {#1}
1005         }

```



```

1006     {
1007         \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
1008         { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
1009         {
1010             \msg_info:nnee { zref-clever } { unknown-decl-case }
1011             {#1} { \l__zrefclever_setup_language_tl }
1012             \seq_get_left:NN \l__zrefclever_lang_declension_seq
1013             \l__zrefclever_lang_decl_case_tl
1014         }
1015     }
1016 } ,
1017 case .value_required:n = true ,
1018
1019 gender .value_required:n = true ,
1020 gender .code:n =
1021 {
1022     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1023     {
1024         \msg_info:nnee { zref-clever } { language-no-gender }
1025         { \l__zrefclever_setup_language_tl } { gender } {#1}
1026     }
1027     {
1028         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1029         {
1030             \msg_info:nnn { zref-clever }
1031             { option-only-type-specific } { gender }
1032         }
1033         {
1034             \seq_clear:N \l__zrefclever_tmpa_seq
1035             \clist_map_inline:nn {#1}
1036             {
1037                 \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1038                 { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
1039                 {
1040                     \msg_info:nnee { zref-clever }
1041                     { gender-not-declared }
1042                     { \l__zrefclever_setup_language_tl } {##1}
1043                 }
1044             }
1045             \__zrefclever_opt_seq_if_set:cF
1046             {
1047                 \__zrefclever_opt_varname_lang_type:eenn
1048                 { \l__zrefclever_setup_language_tl }
1049                 { \l__zrefclever_setup_type_tl }
1050                 { gender }
1051                 { seq }
1052             }
1053             {
1054                 \seq_new:c
1055                 {
1056                     \__zrefclever_opt_varname_lang_type:eenn
1057                     { \l__zrefclever_setup_language_tl }
1058                     { \l__zrefclever_setup_type_tl }
1059                     { gender }

```

```

1060         { seq }
1061     }
1062     \seq_gset_eq:cN
1063     {
1064         \__zrefclever_opt_varname_lang_type:enn
1065         { \l__zrefclever_setup_language_tl }
1066         { \l__zrefclever_setup_type_tl }
1067         { gender }
1068         { seq }
1069     }
1070     \l__zrefclever_tmpa_seq
1071 }
1072 }
1073 }
1074 } ,
1075 }
1076 \seq_map_inline:Nn
1077 \g__zrefclever_rf_opts_tl_not_type_specific_seq
1078 {
1079     \keys_define:nn { zref-clever/langfile }
1080     {
1081         #1 .value_required:n = true ,
1082         #1 .code:n =
1083         {
1084             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1085             {
1086                 \__zrefclever_opt_tl_gset_if_new:cn
1087                 {
1088                     \__zrefclever_opt_varname_lang_default:enn
1089                     { \l__zrefclever_setup_language_tl }
1090                     {#1} { tl }
1091                 }
1092                 {##1}
1093             }
1094             {
1095                 \msg_info:nnn { zref-clever }
1096                 { option-not-type-specific } {#1}
1097             }
1098         } ,
1099     }
1100 }
1101 \seq_map_inline:Nn
1102 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
1103 {
1104     \keys_define:nn { zref-clever/langfile }
1105     {
1106         #1 .value_required:n = true ,
1107         #1 .code:n =
1108         {
1109             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1110             {
1111                 \__zrefclever_opt_tl_gset_if_new:cn
1112                 {
1113                     \__zrefclever_opt_varname_lang_default:enn

```

```

1114         { \l__zrefclever_setup_language_tl }
1115         {#1} { tl }
1116     }
1117     {##1}
1118 }
1119 {
1120     \__zrefclever_opt_tl_gset_if_new:cn
1121     {
1122         \__zrefclever_opt_varname_lang_type:eenn
1123         { \l__zrefclever_setup_language_tl }
1124         { \l__zrefclever_setup_type_tl }
1125         {#1} { tl }
1126     }
1127     {##1}
1128 }
1129 },
1130 }
1131 }
1132 \keys_define:nn { zref-clever/langfile }
1133 {
1134     endrange .value_required:n = true ,
1135     endrange .code:n =
1136     {
1137         \str_case:nnF {#1}
1138         {
1139             { ref }
1140             {
1141                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1142                 {
1143                     \__zrefclever_opt_tl_gclear_if_new:c
1144                     {
1145                         \__zrefclever_opt_varname_lang_default:enn
1146                         { \l__zrefclever_setup_language_tl }
1147                         { endrangefunc } { tl }
1148                     }
1149                     \__zrefclever_opt_tl_gclear_if_new:c
1150                     {
1151                         \__zrefclever_opt_varname_lang_default:enn
1152                         { \l__zrefclever_setup_language_tl }
1153                         { endrangeprop } { tl }
1154                     }
1155                 }
1156             }
1157             \__zrefclever_opt_tl_gclear_if_new:c
1158             {
1159                 \__zrefclever_opt_varname_lang_type:eenn
1160                 { \l__zrefclever_setup_language_tl }
1161                 { \l__zrefclever_setup_type_tl }
1162                 { endrangefunc } { tl }
1163             }
1164             \__zrefclever_opt_tl_gclear_if_new:c
1165             {
1166                 \__zrefclever_opt_varname_lang_type:eenn
1167                 { \l__zrefclever_setup_language_tl }

```

```

1168         { \l__zrefclever_setup_type_tl }
1169         { endrangeprop } { tl }
1170     }
1171 }
1172 }
1173
1174 { stripprefix }
1175 {
1176   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1177   {
1178     \__zrefclever_opt_tl_gset_if_new:cn
1179     {
1180       \__zrefclever_opt_varname_lang_default:enn
1181       { \l__zrefclever_setup_language_tl }
1182       { endrangefunc } { tl }
1183     }
1184     { __zrefclever_get_endrange_stripprefix }
1185     \__zrefclever_opt_tl_gclear_if_new:c
1186     {
1187       \__zrefclever_opt_varname_lang_default:enn
1188       { \l__zrefclever_setup_language_tl }
1189       { endrangeprop } { tl }
1190     }
1191   }
1192   {
1193     \__zrefclever_opt_tl_gset_if_new:cn
1194     {
1195       \__zrefclever_opt_varname_lang_type:eenn
1196       { \l__zrefclever_setup_language_tl }
1197       { \l__zrefclever_setup_type_tl }
1198       { endrangefunc } { tl }
1199     }
1200     { __zrefclever_get_endrange_stripprefix }
1201     \__zrefclever_opt_tl_gclear_if_new:c
1202     {
1203       \__zrefclever_opt_varname_lang_type:eenn
1204       { \l__zrefclever_setup_language_tl }
1205       { \l__zrefclever_setup_type_tl }
1206       { endrangeprop } { tl }
1207     }
1208   }
1209 }
1210
1211 { pagecomp }
1212 {
1213   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1214   {
1215     \__zrefclever_opt_tl_gset_if_new:cn
1216     {
1217       \__zrefclever_opt_varname_lang_default:enn
1218       { \l__zrefclever_setup_language_tl }
1219       { endrangefunc } { tl }
1220     }
1221     { __zrefclever_get_endrange_pagecomp }

```

```

1222     \_zrefclever_opt_tl_gclear_if_new:c
1223     {
1224         \_zrefclever_opt_varname_lang_default:enn
1225         { \l_zrefclever_setup_language_tl }
1226         { endrangeprop } { tl }
1227     }
1228 }
1229 {
1230     \_zrefclever_opt_tl_gset_if_new:cn
1231     {
1232         \_zrefclever_opt_varname_lang_type:eenn
1233         { \l_zrefclever_setup_language_tl }
1234         { \l_zrefclever_setup_type_tl }
1235         { endrangefunc } { tl }
1236     }
1237     { __zrefclever_get_endrange_pagecomp }
1238     \_zrefclever_opt_tl_gclear_if_new:c
1239     {
1240         \_zrefclever_opt_varname_lang_type:eenn
1241         { \l_zrefclever_setup_language_tl }
1242         { \l_zrefclever_setup_type_tl }
1243         { endrangeprop } { tl }
1244     }
1245 }
1246 }
1247
1248 { pagecomp2 }
1249 {
1250     \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1251     {
1252         \_zrefclever_opt_tl_gset_if_new:cn
1253         {
1254             \_zrefclever_opt_varname_lang_default:enn
1255             { \l_zrefclever_setup_language_tl }
1256             { endrangefunc } { tl }
1257         }
1258         { __zrefclever_get_endrange_pagecomptwo }
1259         \_zrefclever_opt_tl_gclear_if_new:c
1260         {
1261             \_zrefclever_opt_varname_lang_default:enn
1262             { \l_zrefclever_setup_language_tl }
1263             { endrangeprop } { tl }
1264         }
1265     }
1266     {
1267         \_zrefclever_opt_tl_gset_if_new:cn
1268         {
1269             \_zrefclever_opt_varname_lang_type:eenn
1270             { \l_zrefclever_setup_language_tl }
1271             { \l_zrefclever_setup_type_tl }
1272             { endrangefunc } { tl }
1273         }
1274         { __zrefclever_get_endrange_pagecomptwo }
1275         \_zrefclever_opt_tl_gclear_if_new:c

```

```

1276         {
1277             \__zrefclever_opt_varname_lang_type:eenn
1278             { \l__zrefclever_setup_language_tl }
1279             { \l__zrefclever_setup_type_tl }
1280             { endrangeprop } { tl }
1281         }
1282     }
1283 }
1284 }
1285 {
1286     \tl_if_empty:nTF {#1}
1287     {
1288         \msg_info:nnn { zref-clever }
1289         { endrange-property-undefined } {#1}
1290     }
1291     {
1292         \zref@ifpropundefined {#1}
1293         {
1294             \msg_info:nnn { zref-clever }
1295             { endrange-property-undefined } {#1}
1296         }
1297         {
1298             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1299             {
1300                 \__zrefclever_opt_tl_gset_if_new:cn
1301                 {
1302                     \__zrefclever_opt_varname_lang_default:enn
1303                     { \l__zrefclever_setup_language_tl }
1304                     { endrangefunc } { tl }
1305                 }
1306                 { __zrefclever_get_endrange_property }
1307                 \__zrefclever_opt_tl_gset_if_new:cn
1308                 {
1309                     \__zrefclever_opt_varname_lang_default:enn
1310                     { \l__zrefclever_setup_language_tl }
1311                     { endrangeprop } { tl }
1312                 }
1313                 {#1}
1314             }
1315             {
1316                 \__zrefclever_opt_tl_gset_if_new:cn
1317                 {
1318                     \__zrefclever_opt_varname_lang_type:eenn
1319                     { \l__zrefclever_setup_language_tl }
1320                     { \l__zrefclever_setup_type_tl }
1321                     { endrangefunc } { tl }
1322                 }
1323                 { __zrefclever_get_endrange_property }
1324                 \__zrefclever_opt_tl_gset_if_new:cn
1325                 {
1326                     \__zrefclever_opt_varname_lang_type:eenn
1327                     { \l__zrefclever_setup_language_tl }
1328                     { \l__zrefclever_setup_type_tl }
1329                     { endrangeprop } { tl }

```

```

1330         }
1331         {#1}
1332     }
1333 }
1334 }
1335 }
1336 } ,
1337 }
1338 \seq_map_inline:Nn
1339 \g__zrefclever_rf_opts_tl_type_names_seq
1340 {
1341     \keys_define:nn { zref-clever/langfile }
1342     {
1343         #1 .value_required:n = true ,
1344         #1 .code:n =
1345         {
1346             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1347             {
1348                 \msg_info:nnn { zref-clever }
1349                 { option-only-type-specific } {#1}
1350             }
1351             {
1352                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1353                 {
1354                     \__zrefclever_opt_tl_gset_if_new:cn
1355                     {
1356                         \__zrefclever_opt_varname_lang_type:een
1357                         { \l__zrefclever_setup_language_tl }
1358                         { \l__zrefclever_setup_type_tl }
1359                         {#1} { tl }
1360                     }
1361                     {##1}
1362                 }
1363                 {
1364                     \__zrefclever_opt_tl_gset_if_new:cn
1365                     {
1366                         \__zrefclever_opt_varname_lang_type:een
1367                         { \l__zrefclever_setup_language_tl }
1368                         { \l__zrefclever_setup_type_tl }
1369                         { \l__zrefclever_lang_decl_case_tl - #1 } { tl }
1370                     }
1371                     {##1}
1372                 }
1373             }
1374         } ,
1375     }
1376 }
1377 \seq_map_inline:Nn
1378 \g__zrefclever_rf_opts_seq_refbounds_seq
1379 {
1380     \keys_define:nn { zref-clever/langfile }
1381     {
1382         #1 .value_required:n = true ,
1383         #1 .code:n =

```

```

1384 {
1385   \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1386   {
1387     \__zrefclever_opt_seq_if_set:cF
1388     {
1389       \__zrefclever_opt_varname_lang_default:enn
1390       { \l__zrefclever_setup_language_tl } {#1} { seq }
1391     }
1392     {
1393       \seq_gclear:N \g__zrefclever_tmpa_seq
1394       \__zrefclever_opt_seq_gset_clist_split:Nn
1395       \g__zrefclever_tmpa_seq {##1}
1396       \bool_lazy_or:nnTF
1397       { \tl_if_empty_p:n {##1} }
1398       {
1399         \int_compare_p:nNn
1400         { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1401       }
1402       {
1403         \__zrefclever_opt_seq_gset_eq:cN
1404         {
1405           \__zrefclever_opt_varname_lang_default:enn
1406           { \l__zrefclever_setup_language_tl }
1407           {#1} { seq }
1408         }
1409         \g__zrefclever_tmpa_seq
1410       }
1411       {
1412         \msg_info:nnee { zref-clever }
1413         { refbounds-must-be-four }
1414         {#1} { \seq_count:N \g__zrefclever_tmpa_seq }
1415       }
1416     }
1417   }
1418   {
1419     \__zrefclever_opt_seq_if_set:cF
1420     {
1421       \__zrefclever_opt_varname_lang_type:eenn
1422       { \l__zrefclever_setup_language_tl }
1423       { \l__zrefclever_setup_type_tl } {#1} { seq }
1424     }
1425     {
1426       \seq_gclear:N \g__zrefclever_tmpa_seq
1427       \__zrefclever_opt_seq_gset_clist_split:Nn
1428       \g__zrefclever_tmpa_seq {##1}
1429       \bool_lazy_or:nnTF
1430       { \tl_if_empty_p:n {##1} }
1431       {
1432         \int_compare_p:nNn
1433         { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
1434       }
1435       {
1436         \__zrefclever_opt_seq_gset_eq:cN
1437         {

```



```

1438         \_zrefclever_opt_varname_lang_type:eenn
1439         { \l_zrefclever_setup_language_tl }
1440         { \l_zrefclever_setup_type_tl }
1441         {#1} { seq }
1442     }
1443     \g_zrefclever_tmpa_seq
1444 }
1445 {
1446     \msg_info:nnee { zref-clever }
1447     { refbounds-must-be-four }
1448     {#1} { \seq_count:N \g_zrefclever_tmpa_seq }
1449 }
1450 }
1451 } ,
1452 }
1453 }
1454 }
1455 \seq_map_inline:Nn
1456 \g_zrefclever_rf_opts_bool_maybe_type_specific_seq
1457 {
1458     \keys_define:nn { zref-clever/langfile }
1459     {
1460         #1 .choice: ,
1461         #1 / true .code:n =
1462         {
1463             \tl_if_empty:NTF \l_zrefclever_setup_type_tl
1464             {
1465                 \_zrefclever_opt_bool_if_set:cF
1466                 {
1467                     \_zrefclever_opt_varname_lang_default:enn
1468                     { \l_zrefclever_setup_language_tl }
1469                     {#1} { bool }
1470                 }
1471                 {
1472                     \_zrefclever_opt_bool_gset_true:c
1473                     {
1474                         \_zrefclever_opt_varname_lang_default:enn
1475                         { \l_zrefclever_setup_language_tl }
1476                         {#1} { bool }
1477                     }
1478                 }
1479             }
1480         }
1481         \_zrefclever_opt_bool_if_set:cF
1482         {
1483             \_zrefclever_opt_varname_lang_type:eenn
1484             { \l_zrefclever_setup_language_tl }
1485             { \l_zrefclever_setup_type_tl }
1486             {#1} { bool }
1487         }
1488         {
1489             \_zrefclever_opt_bool_gset_true:c
1490             {
1491                 \_zrefclever_opt_varname_lang_type:eenn

```

```

1492         { \l__zrefclever_setup_language_tl }
1493         { \l__zrefclever_setup_type_tl }
1494         {#1} { bool }
1495     }
1496 }
1497 }
1498 } ,
1499 #1 / false .code:n =
1500 {
1501     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1502     {
1503         \__zrefclever_opt_bool_if_set:cF
1504         {
1505             \__zrefclever_opt_varname_lang_default:enn
1506             { \l__zrefclever_setup_language_tl }
1507             {#1} { bool }
1508         }
1509         {
1510             \__zrefclever_opt_bool_gset_false:c
1511             {
1512                 \__zrefclever_opt_varname_lang_default:enn
1513                 { \l__zrefclever_setup_language_tl }
1514                 {#1} { bool }
1515             }
1516         }
1517     }
1518     {
1519         \__zrefclever_opt_bool_if_set:cF
1520         {
1521             \__zrefclever_opt_varname_lang_type:eenn
1522             { \l__zrefclever_setup_language_tl }
1523             { \l__zrefclever_setup_type_tl }
1524             {#1} { bool }
1525         }
1526         {
1527             \__zrefclever_opt_bool_gset_false:c
1528             {
1529                 \__zrefclever_opt_varname_lang_type:eenn
1530                 { \l__zrefclever_setup_language_tl }
1531                 { \l__zrefclever_setup_type_tl }
1532                 {#1} { bool }
1533             }
1534         }
1535     }
1536 } ,
1537 #1 .default:n = true ,
1538 no #1 .meta:n = { #1 = false } ,
1539 no #1 .value_forbidden:n = true ,
1540 }
1541 }

```

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.

```

1542 \cs_new_protected:Npn \__zrefclever_opt_tl_cset_fallback:nn #1#2
1543 {
1544   \tl_const:cn
1545     { \__zrefclever_opt_varname_fallback:nn {#1} { tl } } {#2}
1546 }
1547 \keyval_parse:nnn
1548 { }
1549 { \__zrefclever_opt_tl_cset_fallback:nn }
1550 {
1551   tpairsep = {,~} ,
1552   tlistsep = {,~} ,
1553   tlastsep = {,~} ,
1554   notesep = {~} ,
1555   namesep = {\nobreakspace} ,
1556   pairsep = {,~} ,
1557   listsep = {,~} ,
1558   lastsep = {,~} ,
1559   rangsep = {\textendash} ,
1560 }

```

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>}

```

```

1561 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
1562 {
1563   \tl_if_empty:nTF {#3}
1564     { \prop_remove:Nn #1 {#2} }
1565     { \prop_put:Nnn #1 {#2} {#3} }
1566 }

```

(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`.

```

1567 \tl_new:N \l__zrefclever_ref_property_tl

```

```

1568 \keys_define:nn { zref-clever/reference }
1569 {
1570   ref .code:n =
1571   {
1572     \tl_if_empty:nTF {#1}
1573     {
1574       \msg_warning:nnn { zref-clever }
1575       { zref-property-undefined } {#1}
1576       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1577     }
1578     {
1579       \zref@ifpropundefined {#1}
1580       {
1581         \msg_warning:nnn { zref-clever }
1582         { zref-property-undefined } {#1}
1583         \tl_set:Nn \l__zrefclever_ref_property_tl { default }
1584       }
1585       { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
1586     }
1587   } ,
1588   ref .initial:n = default ,
1589   ref .value_required:n = true ,
1590   page .meta:n = { ref = page } ,
1591   page .value_forbidden:n = true ,
1592 }

```

typeset option

```

1593 \bool_new:N \l__zrefclever_typeset_ref_bool
1594 \bool_new:N \l__zrefclever_typeset_name_bool
1595 \keys_define:nn { zref-clever/reference }
1596 {
1597   typeset .choice: ,
1598   typeset / both .code:n =
1599   {
1600     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1601     \bool_set_true:N \l__zrefclever_typeset_name_bool
1602   } ,
1603   typeset / ref .code:n =
1604   {
1605     \bool_set_true:N \l__zrefclever_typeset_ref_bool
1606     \bool_set_false:N \l__zrefclever_typeset_name_bool
1607   } ,
1608   typeset / name .code:n =
1609   {
1610     \bool_set_false:N \l__zrefclever_typeset_ref_bool
1611     \bool_set_true:N \l__zrefclever_typeset_name_bool
1612   } ,
1613   typeset .initial:n = both ,
1614   typeset .value_required:n = true ,
1615
1616   noname .meta:n = { typeset = ref } ,
1617   noname .value_forbidden:n = true ,
1618   noref .meta:n = { typeset = name } ,

```

```

1619     noref .value_forbidden:n = true ,
1620   }

```

sort option

```

1621 \bool_new:N \l__zrefclever_typeset_sort_bool
1622 \keys_define:nn { zref-clever/reference }
1623 {
1624   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
1625   sort .initial:n = true ,
1626   sort .default:n = true ,
1627   nosort .meta:n = { sort = false },
1628   nosort .value_forbidden:n = true ,
1629 }

```

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`.

```

1630 \seq_new:N \l__zrefclever_typesort_seq
1631 \keys_define:nn { zref-clever/reference }
1632 {
1633   typesort .code:n =
1634     {
1635       \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
1636       \seq_reverse:N \l__zrefclever_typesort_seq
1637     } ,
1638   typesort .initial:n =
1639     { part , chapter , section , paragraph },
1640   typesort .value_required:n = true ,
1641   notypesort .code:n =
1642     { \seq_clear:N \l__zrefclever_typesort_seq } ,
1643   notypesort .value_forbidden:n = true ,
1644 }

```

comp option

```

1645 \bool_new:N \l__zrefclever_typeset_compress_bool
1646 \keys_define:nn { zref-clever/reference }
1647 {
1648   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
1649   comp .initial:n = true ,
1650   comp .default:n = true ,
1651   nocomp .meta:n = { comp = false },
1652   nocomp .value_forbidden:n = true ,
1653 }

```

endrange option

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

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

```

1654 \NewHook { zref-clever/endrange-setup }
1655 \keys_define:nn { zref-clever/reference }
1656 {
1657   endrange .code:n =
1658   {
1659     \str_case:nnF {#1}
1660     {
1661       { ref }
1662       {
1663         \__zrefclever_opt_tl_clear:c
1664         {
1665           \__zrefclever_opt_varname_general:nn
1666             { endrangefunc } { tl }
1667         }
1668         \__zrefclever_opt_tl_clear:c
1669         {
1670           \__zrefclever_opt_varname_general:nn
1671             { endrangeprop } { tl }
1672         }
1673       }
1674     }
1675     { stripprefix }
1676     {
1677       \__zrefclever_opt_tl_set:cn
1678       {
1679         \__zrefclever_opt_varname_general:nn

```

```

1680         { endrangefunc } { t1 }
1681     }
1682     { __zrefclever_get_endrange_stripprefix }
1683     \__zrefclever_opt_t1_clear:c
1684     {
1685         \__zrefclever_opt_varname_general:nn
1686         { endrangeprop } { t1 }
1687     }
1688 }
1689
1690 { pagecomp }
1691 {
1692     \__zrefclever_opt_t1_set:cn
1693     {
1694         \__zrefclever_opt_varname_general:nn
1695         { endrangefunc } { t1 }
1696     }
1697     { __zrefclever_get_endrange_pagecomp }
1698     \__zrefclever_opt_t1_clear:c
1699     {
1700         \__zrefclever_opt_varname_general:nn
1701         { endrangeprop } { t1 }
1702     }
1703 }
1704
1705 { pagecomp2 }
1706 {
1707     \__zrefclever_opt_t1_set:cn
1708     {
1709         \__zrefclever_opt_varname_general:nn
1710         { endrangefunc } { t1 }
1711     }
1712     { __zrefclever_get_endrange_pagecomptwo }
1713     \__zrefclever_opt_t1_clear:c
1714     {
1715         \__zrefclever_opt_varname_general:nn
1716         { endrangeprop } { t1 }
1717     }
1718 }
1719
1720 { unset }
1721 {
1722     \__zrefclever_opt_t1_unset:c
1723     {
1724         \__zrefclever_opt_varname_general:nn
1725         { endrangefunc } { t1 }
1726     }
1727     \__zrefclever_opt_t1_unset:c
1728     {
1729         \__zrefclever_opt_varname_general:nn
1730         { endrangeprop } { t1 }
1731     }
1732 }
1733 }

```

```

1734     {
1735       \tl_if_empty:nTF {#1}
1736       {
1737         \msg_warning:nnn { zref-clever }
1738         { endrange-property-undefined } {#1}
1739       }
1740       {
1741         \zref@ifpropundefined {#1}
1742         {
1743           \msg_warning:nnn { zref-clever }
1744           { endrange-property-undefined } {#1}
1745         }
1746         {
1747           \__zrefclever_opt_tl_set:cn
1748           {
1749             \__zrefclever_opt_varname_general:nn
1750             { endrangefunc } { tl }
1751           }
1752           { __zrefclever_get_endrange_property }
1753           \__zrefclever_opt_tl_set:cn
1754           {
1755             \__zrefclever_opt_varname_general:nn
1756             { endrangeprop } { tl }
1757           }
1758           {#1}
1759         }
1760       }
1761     }
1762   } ,
1763   endrange .value_required:n = true ,
1764 }
1765 \cs_new_protected:Npn \__zrefclever_get_endrange_property:nnN #1#2#3
1766 {
1767   \tl_if_empty:NTF \l__zrefclever_endrangeprop_tl
1768   {
1769     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1770     {
1771       \__zrefclever_extract_default:Nnvn #3
1772       {#2} { \l__zrefclever_ref_property_tl } { }
1773     }
1774     { \tl_set:Nn #3 { zc@missingproperty } }
1775   }
1776   {
1777     \zref@ifrefcontainsprop {#2} { \l__zrefclever_endrangeprop_tl }
1778     {

```

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.

```

1779       \bool_if:NTF \l__zrefclever_typeset_range_bool
1780       {
1781         \group_begin:

```



```

1782 \bool_set_false:N \l__zrefclever_tmpa_bool
1783 \exp_args:Nee \tl_if_eq:nnT
1784 {
1785   \__zrefclever_extract_unexp:nnn
1786   {#1} { externaldocument } { }
1787 }
1788 {
1789   \__zrefclever_extract_unexp:nnn
1790   {#2} { externaldocument } { }
1791 }
1792 {
1793   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1794   {
1795     \exp_args:Nee \tl_if_eq:nnT
1796     {
1797       \__zrefclever_extract_unexp:nnn
1798       {#1} { zc@pgfmt } { }
1799     }
1800     {
1801       \__zrefclever_extract_unexp:nnn
1802       {#2} { zc@pgfmt } { }
1803     }
1804     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1805   }
1806   {
1807     \exp_args:Nee \tl_if_eq:nnT
1808     {
1809       \__zrefclever_extract_unexp:nnn
1810       {#1} { zc@counter } { }
1811     }
1812     {
1813       \__zrefclever_extract_unexp:nnn
1814       {#2} { zc@counter } { }
1815     }
1816     {
1817       \exp_args:Nee \tl_if_eq:nnT
1818       {
1819         \__zrefclever_extract_unexp:nnn
1820         {#1} { zc@enclval } { }
1821       }
1822       {
1823         \__zrefclever_extract_unexp:nnn
1824         {#2} { zc@enclval } { }
1825       }
1826       { \bool_set_true:N \l__zrefclever_tmpa_bool }
1827     }
1828   }
1829 }
1830 \bool_if:NTF \l__zrefclever_tmpa_bool
1831 {
1832   \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1833   {#2} { l__zrefclever_endrangeprop_tl } { }
1834 }
1835 {

```

```

1836         \zref@ifrefcontainsprop
1837         {#2} { \l__zrefclever_ref_property_tl }
1838         {
1839         \__zrefclever_extract_default:Nnvn \l__zrefclever_tmpb_tl
1840         {#2} { l__zrefclever_ref_property_tl } { }
1841         }
1842         { \tl_set:Nn \l__zrefclever_tmpb_tl { zc@missingproperty } }
1843     }
1844     \exp_args:NNNV
1845     \group_end:
1846     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1847 }
1848 {
1849     \__zrefclever_extract_default:Nnvn #3
1850     {#2} { l__zrefclever_endrangeprop_tl } { }
1851 }
1852 }
1853 {
1854     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1855     {
1856         \__zrefclever_extract_default:Nnvn #3
1857         {#2} { l__zrefclever_ref_property_tl } { }
1858     }
1859     { \tl_set:Nn #3 { zc@missingproperty } }
1860 }
1861 }
1862 }
1863 \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>.

```

1864 \cs_new_protected:Npn \__zrefclever_get_endrange_stripprefix:nnN #1#2#3
1865 {
1866     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1867     {
1868         \group_begin:
1869         \UseHook { zref-clever/endrange-setup }
1870         \tl_set:Ne \l__zrefclever_tmpa_tl
1871         {
1872             \__zrefclever_extract:nnn
1873             {#1} { \l__zrefclever_ref_property_tl } { }
1874         }
1875         \tl_set:Ne \l__zrefclever_tmpb_tl
1876         {
1877             \__zrefclever_extract:nnn
1878             {#2} { \l__zrefclever_ref_property_tl } { }
1879         }
1880         \bool_set_false:N \l__zrefclever_tmpa_bool
1881         \bool_until_do:Nn \l__zrefclever_tmpa_bool
1882         {
1883             \exp_args:Nee \tl_if_eq:nnTF
1884             { \tl_head:V \l__zrefclever_tmpa_tl }
1885             { \tl_head:V \l__zrefclever_tmpb_tl }
1886             {

```

```

1887         \tl_set:Ne \l__zrefclever_tmpa_tl
1888         { \tl_tail:V \l__zrefclever_tmpa_tl }
1889         \tl_set:Ne \l__zrefclever_tmpb_tl
1890         { \tl_tail:V \l__zrefclever_tmpb_tl }
1891         \tl_if_empty:NT \l__zrefclever_tmpb_tl
1892         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1893     }
1894     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1895 }
1896 \exp_args:NNNV
1897 \group_end:
1898 \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1899 }
1900 { \tl_set:Nn #3 { zc@missingproperty } }
1901 }
1902 \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>).

```

1903 \prg_new_protected_conditional:Npnn
1904 \__zrefclever_is_integer_rgx:n #1 { F , TF }
1905 {
1906     \regex_match:nnTF { \A\d+\Z } {#1}
1907     { \prg_return_true: }
1908     { \prg_return_false: }
1909 }
1910 \prg_generate_conditional_variant:Nnn
1911 \__zrefclever_is_integer_rgx:n { V } { F , TF }

```

(End of definition for __zrefclever_is_integer_rgx:n.)

```

1912 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomp:nnN #1#2#3
1913 {
1914     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1915     {
1916         \group_begin:
1917         \UseHook { zref-clever/endrange-setup }
1918         \tl_set:Ne \l__zrefclever_tmpa_tl
1919         {
1920             \__zrefclever_extract:nnn
1921             {#1} { \l__zrefclever_ref_property_tl } { }
1922         }
1923         \tl_set:Ne \l__zrefclever_tmpb_tl
1924         {
1925             \__zrefclever_extract:nnn
1926             {#2} { \l__zrefclever_ref_property_tl } { }
1927         }
1928         \bool_set_false:N \l__zrefclever_tmpa_bool
1929         \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1930         {
1931             \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1932             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1933         }
1934         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1935     }

```

```

1936     {
1937         \exp_args:Nee \tl_if_eq:nnTF
1938         { \tl_head:V \l__zrefclever_tmpa_tl }
1939         { \tl_head:V \l__zrefclever_tmpb_tl }
1940         {
1941             \tl_set:Ne \l__zrefclever_tmpa_tl
1942             { \tl_tail:V \l__zrefclever_tmpa_tl }
1943             \tl_set:Ne \l__zrefclever_tmpb_tl
1944             { \tl_tail:V \l__zrefclever_tmpb_tl }
1945             \tl_if_empty:NT \l__zrefclever_tmpb_tl
1946             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1947         }
1948         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1949     }
1950     \exp_args:NNNV
1951     \group_end:
1952     \tl_set:Nn #3 \l__zrefclever_tmpb_tl
1953 }
1954 { \tl_set:Nn #3 { zc@missingproperty } }
1955 }
1956 \cs_generate_variant:Nn \__zrefclever_get_endrange_pagecomp:nnN { VVN }
1957 \cs_new_protected:Npn \__zrefclever_get_endrange_pagecomptwo:nnN #1#2#3
1958 {
1959     \zref@ifrefcontainsprop {#2} { \l__zrefclever_ref_property_tl }
1960     {
1961         \group_begin:
1962         \UseHook { zref-clever/endrange-setup }
1963         \tl_set:Ne \l__zrefclever_tmpa_tl
1964         {
1965             \__zrefclever_extract:nnn
1966             {#1} { \l__zrefclever_ref_property_tl } { }
1967         }
1968         \tl_set:Ne \l__zrefclever_tmpb_tl
1969         {
1970             \__zrefclever_extract:nnn
1971             {#2} { \l__zrefclever_ref_property_tl } { }
1972         }
1973         \bool_set_false:N \l__zrefclever_tmpa_bool
1974         \__zrefclever_is_integer_rgx:VTF \l__zrefclever_tmpa_tl
1975         {
1976             \__zrefclever_is_integer_rgx:VF \l__zrefclever_tmpb_tl
1977             { \bool_set_true:N \l__zrefclever_tmpa_bool }
1978         }
1979         { \bool_set_true:N \l__zrefclever_tmpa_bool }
1980     \bool_until_do:Nn \l__zrefclever_tmpa_bool
1981     {
1982         \exp_args:Nee \tl_if_eq:nnTF
1983         { \tl_head:V \l__zrefclever_tmpa_tl }
1984         { \tl_head:V \l__zrefclever_tmpb_tl }
1985         {
1986             \bool_lazy_or:nnTF
1987             { \int_compare_p:nNn { \l__zrefclever_tmpb_tl } > { 99 } }
1988             {
1989                 \int_compare_p:nNn

```

```

1990         { \tl_head:V \l__zrefclever_tmpb_tl } = { 0 }
1991     }
1992     {
1993         \tl_set:Ne \l__zrefclever_tmpa_tl
1994         { \tl_tail:V \l__zrefclever_tmpa_tl }
1995         \tl_set:Ne \l__zrefclever_tmpb_tl
1996         { \tl_tail:V \l__zrefclever_tmpb_tl }
1997     }
1998     { \bool_set_true:N \l__zrefclever_tmpa_bool }
1999 }
2000 { \bool_set_true:N \l__zrefclever_tmpa_bool }
2001 }
2002 \exp_args:NNNV
2003 \group_end:
2004 \tl_set:Nn #3 \l__zrefclever_tmpb_tl
2005 }
2006 { \tl_set:Nn #3 { zc@missingproperty } }
2007 }
2008 \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`.

```

2009 \bool_new:N \l__zrefclever_typeset_range_bool
2010 \keys_define:nn { zref-clever/reference }
2011 {
2012     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
2013     range .initial:n = false ,
2014     range .default:n = true ,
2015 }

```

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`.

```

2016 \bool_new:N \l__zrefclever_capfirst_bool
2017 \keys_define:nn { zref-clever/reference }
2018 {
2019     capfirst .bool_set:N = \l__zrefclever_capfirst_bool ,
2020     capfirst .initial:n = false ,
2021     capfirst .default:n = true ,
2022 }

```

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`.

```

2023 \bool_new:N \l__zrefclever_noabbrev_first_bool
2024 \keys_define:nn { zref-clever/reference }
2025 {
2026     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,

```

```

2027     noabbrevfirst .initial:n = false ,
2028     noabbrevfirst .default:n = true ,
2029 }

```

S option

```

2030 \keys_define:nm { zref-clever/reference }
2031 {
2032   S .meta:n =
2033     { capfirst = {#1} , noabbrevfirst = {#1} },
2034   S .default:n = true ,
2035 }

```

hyperref option

```

2036 \bool_new:N \l__zrefclever_hyperlink_bool
2037 \bool_new:N \l__zrefclever_hyperref_warn_bool
2038 \keys_define:nm { zref-clever/reference }
2039 {
2040   hyperref .choice: ,
2041   hyperref / auto .code:n =
2042     {
2043       \bool_set_true:N \l__zrefclever_hyperlink_bool
2044       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2045     } ,
2046   hyperref / true .code:n =
2047     {
2048       \bool_set_true:N \l__zrefclever_hyperlink_bool
2049       \bool_set_true:N \l__zrefclever_hyperref_warn_bool
2050     } ,
2051   hyperref / false .code:n =
2052     {
2053       \bool_set_false:N \l__zrefclever_hyperlink_bool
2054       \bool_set_false:N \l__zrefclever_hyperref_warn_bool
2055     } ,
2056   hyperref .initial:n = auto ,
2057   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.

```

2058     nohyperref .meta:n = { hyperref = false } ,
2059     nohyperref .value_forbidden:n = true ,
2060 }
2061 \AddToHook { begindocument }
2062 {
2063   \__zrefclever_if_package_loaded:nTF { hyperref }
2064   {
2065     \bool_if:NT \l__zrefclever_hyperlink_bool
2066     { \RequirePackage { zref-hyperref } }
2067   }
2068   {
2069     \bool_if:NT \l__zrefclever_hyperref_warn_bool

```

```

2070         { \msg_warning:nn { zref-clever } { missing-hyperref } }
2071         \bool_set_false:N \l__zrefclever_hyperlink_bool
2072     }
2073     \keys_define:nn { zref-clever/reference }
2074     {
2075         hyperref .code:n =
2076         { \msg_warning:nn { zref-clever } { hyperref-preamble-only } } ,
2077         nohyperref .code:n =
2078         { \bool_set_false:N \l__zrefclever_hyperlink_bool } ,
2079     }
2080 }

```

nameinlink option

```

2081 \str_new:N \l__zrefclever_nameinlink_str
2082 \keys_define:nn { zref-clever/reference }
2083 {
2084     nameinlink .choice: ,
2085     nameinlink / true .code:n =
2086     { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
2087     nameinlink / false .code:n =
2088     { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
2089     nameinlink / single .code:n =
2090     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
2091     nameinlink / tsingle .code:n =
2092     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
2093     nameinlink .initial:n = tsingle ,
2094     nameinlink .default:n = true ,
2095 }

```

preposinlink option (deprecated)

```

2096 \keys_define:nn { zref-clever/reference }
2097 {
2098     preposinlink .code:n =
2099     {
2100         % NOTE Option deprecated in 2022-01-12 for v0.2.0-alpha.
2101         \msg_warning:nnnn { zref-clever } { option-deprecated }
2102         { preposinlink } { refbounds }
2103     } ,
2104 }

```

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`.

```

2105 \AddToHook { begindocument }
2106 {
2107   \__zrefclever_if_package_loaded:nTF { babel }
2108   {
2109     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
2110     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
2111   }
2112   {
2113     \__zrefclever_if_package_loaded:nTF { polyglossia }
2114     {
2115       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
2116       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
2117     }
2118     {
2119       \tl_set:Nn \l__zrefclever_current_language_tl { english }
2120       \tl_set:Nn \l__zrefclever_main_language_tl { english }
2121     }
2122   }
2123 }
2124 \keys_define:nn { zref-clever/reference }
2125 {
2126   lang .code:n =
2127   {
2128     \AddToHook { begindocument }
2129     {
2130       \str_case:nnF {#1}
2131       {
2132         { current }
2133         {
2134           \tl_set:Nn \l__zrefclever_ref_language_tl
2135             { \l__zrefclever_current_language_tl }
2136         }
2137
2138         { main }
2139         {
2140           \tl_set:Nn \l__zrefclever_ref_language_tl
2141             { \l__zrefclever_main_language_tl }
2142         }
2143       }
2144     {
2145       \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2146       \__zrefclever_language_if_declared:nF {#1}
2147       {
2148         \msg_warning:nnn { zref-clever }
2149           { unknown-language-opt } {#1}
2150       }
2151     }

```



```

2152         \_zrefclever_provide_langfile:e
2153         { \l__zrefclever_ref_language_tl }
2154     }
2155 },
2156 lang .initial:n = current ,
2157 lang .value_required:n = true ,
2158 }
2159 \AddToHook { begindocument / before }
2160 {
2161     \AddToHook { begindocument }
2162     {

```

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.

```

2163     \keys_define:nn { zref-clever/reference }
2164     {
2165         lang .code:n =
2166         {
2167             \str_case:nnF {#1}
2168             {
2169                 { current }
2170                 {
2171                     \tl_set:Nn \l__zrefclever_ref_language_tl
2172                     { \l__zrefclever_current_language_tl }
2173                 }
2174
2175                 { main }
2176                 {
2177                     \tl_set:Nn \l__zrefclever_ref_language_tl
2178                     { \l__zrefclever_main_language_tl }
2179                 }
2180             }
2181             {
2182                 \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
2183                 \_zrefclever_language_if_declared:nF {#1}
2184                 {
2185                     \msg_warning:nnn { zref-clever }
2186                     { unknown-language-opt } {#1}
2187                 }
2188             }
2189         } ,
2190     }
2191 }
2192 }

```

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](https://github.com/frougon/xcref)), have been an insightful source to frame the problem in general terms.

```

2193 \tl_new:N \l__zrefclever_ref_decl_case_tl
2194 \keys_define:nn { zref-clever/reference }
2195 {
2196   d .code:n =
2197     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
2198 }
2199 \AddToHook { begindocument }
2200 {
2201   \keys_define:nn { zref-clever/reference }
2202   {

```

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

```

2203     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
2204     d .value_required:n = true ,
2205   }
2206 }

```

nudge & co. options

```

2207 \bool_new:N \l__zrefclever_nudge_enabled_bool
2208 \bool_new:N \l__zrefclever_nudge_multitype_bool
2209 \bool_new:N \l__zrefclever_nudge_comptosing_bool
2210 \bool_new:N \l__zrefclever_nudge_singular_bool
2211 \bool_new:N \l__zrefclever_nudge_gender_bool
2212 \tl_new:N \l__zrefclever_ref_gender_tl
2213 \keys_define:nn { zref-clever/reference }
2214 {
2215   nudge .choice: ,
2216   nudge / true .code:n =
2217     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
2218   nudge / false .code:n =
2219     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
2220   nudge / ifdraft .code:n =
2221     {
2222       \ifdraft
2223         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2224         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2225     } ,
2226   nudge / iffinal .code:n =
2227     {
2228       \ifoptionfinal
2229         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
2230         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
2231     } ,
2232   nudge .initial:n = false ,
2233   nudge .default:n = true ,
2234   nonudge .meta:n = { nudge = false } ,
2235   nonudge .value_forbidden:n = true ,
2236   nudgeif .code:n =
2237     {
2238       \bool_set_false:N \l__zrefclever_nudge_multitype_bool

```

```

2239     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
2240     \bool_set_false:N \l__zrefclever_nudge_gender_bool
2241     \clist_map_inline:nn {#1}
2242     {
2243         \str_case:nnF {##1}
2244         {
2245             { multitype }
2246             { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
2247             { comptosing }
2248             { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
2249             { gender }
2250             { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
2251             { all }
2252             {
2253                 \bool_set_true:N \l__zrefclever_nudge_multitype_bool
2254                 \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
2255                 \bool_set_true:N \l__zrefclever_nudge_gender_bool
2256             }
2257         }
2258         {
2259             \msg_warning:nnn { zref-clever }
2260             { nudgeif-unknown-value } {##1}
2261         }
2262     }
2263 },
2264 nudgeif .value_required:n = true ,
2265 nudgeif .initial:n = all ,
2266 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
2267 sg .initial:n = false ,
2268 sg .default:n = true ,
2269 g .code:n =
2270 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
2271 }
2272 \AddToHook { begindocument }
2273 {
2274     \keys_define:nn { zref-clever/reference }
2275     {

```

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

```

2276     g .tl_set:N = \l__zrefclever_ref_gender_tl ,
2277     g .value_required:n = true ,
2278 }
2279 }
```

font option

```

2280 \tl_new:N \l__zrefclever_ref_typeset_font_tl
2281 \keys_define:nn { zref-clever/reference }
2282 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

titleref option

```

2283 \keys_define:nn { zref-clever/reference }
2284 {
2285     titleref .code:n =
```

```

2286     {
2287         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2288         \msg_warning:nnee { zref-clever }{ option-deprecated } { titleref }
2289         { \iow_char:N\usepackage\iow_char:N\{zref-titleref\iow_char:N\} }
2290     } ,
2291 }

```

vario option

```

2292 \keys_define:nn { zref-clever/reference }
2293 {
2294     vario .code:n =
2295     {
2296         % NOTE Option deprecated in 2022-04-22 for 0.3.0.
2297         \msg_warning:nnee { zref-clever }{ option-deprecated } { vario }
2298         { \iow_char:N\usepackage\iow_char:N\{zref-vario\iow_char:N\} }
2299     } ,
2300 }

```

note option

```

2301 \tl_new:N \l__zrefclever_zceref_note_tl
2302 \keys_define:nn { zref-clever/reference }
2303 {
2304     note .tl_set:N = \l__zrefclever_zceref_note_tl ,
2305     note .value_required:n = true ,
2306 }

```

check option

Integration with zref-check.

```

2307 \bool_new:N \l__zrefclever_zrefcheck_available_bool
2308 \bool_new:N \l__zrefclever_zceref_with_check_bool
2309 \keys_define:nn { zref-clever/reference }
2310 {
2311     check .code:n =
2312     { \msg_warning:nnn { zref-clever } { option-document-only } { check } } ,
2313 }
2314 \AddToHook { begindocument }
2315 {
2316     \__zrefclever_if_package_loaded:nTF { zref-check }
2317     {
2318         \IfPackageAtLeastTF { zref-check } { 2021-09-16 }
2319         {
2320             \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
2321             \keys_define:nn { zref-clever/reference }
2322             {
2323                 check .code:n =
2324                 {
2325                     \bool_set_true:N \l__zrefclever_zceref_with_check_bool
2326                     \keys_set:nn { zref-check / zcheck } {#1}
2327                 } ,
2328                 check .value_required:n = true ,
2329             }
2330         }
2331     }

```

```

2332     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2333     \keys_define:nn { zref-clever/reference }
2334     {
2335         check .code:n =
2336         {
2337             \msg_warning:nnn { zref-clever }
2338             { zref-check-too-old } { 2021-09-16~v0.2.1 }
2339         } ,
2340     }
2341 }
2342 }
2343 {
2344     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
2345     \keys_define:nn { zref-clever/reference }
2346     {
2347         check .code:n =
2348         { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
2349     }
2350 }
2351 }

```

reftype option

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

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

```

2352 \tl_new:N \l__zrefclever_reftype_override_tl
2353 \keys_define:nn { zref-clever/label }
2354 {
2355     reftype .tl_set:N = \l__zrefclever_reftype_override_tl ,
2356     reftype .default:n = {} ,
2357     reftype .initial:n = {} ,
2358 }

```

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`.

```

2359 \prop_new:N \l__zrefclever_counter_type_prop
2360 \keys_define:nn { zref-clever/label }
2361 {
2362     countertype .code:n =
2363     {
2364         \keyval_parse:nnn
2365         {
2366             \msg_warning:nnnn { zref-clever }
2367             { key-requires-value } { countertype }
2368         }

```

```

2369         {
2370             \__zrefclever_prop_put_non_empty:Nnn
2371             \l__zrefclever_counter_type_prop
2372         }
2373         {#1}
2374     } ,
2375     countertype .value_required:n = true ,
2376     countertype .initial:n =
2377     {
2378         subsection    = section ,
2379         subsubsection = section ,
2380         subparagraph  = paragraph ,
2381         enumi         = item ,
2382         enumii        = item ,
2383         enumiii       = item ,
2384         enumiv        = item ,
2385         mpfootnote    = footnote ,
2386     } ,
2387 }

```

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

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclval` property, and stores the list of counters which are potential “enclosing counters” for other counters.

Note that, as far as L^AT_EX is concerned, a given counter can be reset by *any number of counters*. `\counterwithin` just adds a new “within-counter” for “counter” without removing any other existing ones. However, the data structure of `zref-clever` can only account for *one* enclosing counter. In a way, this is hard to circumvent, because the underlying counter reset behavior works “top-down”, but when looking to a label built from a given counter we need to infer the enclosing counters “bottom-up”. As a result, the reset chain we find is path dependent or, more formally, what `__zrefclever_counter_reset_by:n` returns depends on the order in which it searches the list of `\l__zrefclever_counter_resetters_seq`, since it stops on the first match. This representation mismatch should not be a problem in most cases. But one should be aware of the limits it imposes.

Consider the following case: the `book` class sets, by default `figure` and `table` counters to be reset every `chapter`, `section` is also reset every `chapter`, of course. Suppose

now we say `\counterwithin{figure}{section}`. Technically, `figure` is being reset every `section` and every `chapter`, but since `section` is also reset every `chapter`, the original “`chapter` resets `figure`” behavior is now redundant. Innocuous, but is still there. Now, suppose we want to find which counter is resetting `figure` using `__zrefclever-counter_reset_by:n`. If `chapter` comes before `section` in `\l__zrefclever-counter-resetters_seq`, `chapter` will be returned, and that’s not what we want. That’s the reason `counterresetters` initial value goes bottom-up in the sectioning level, since we’d expect the nesting of the reset chain to *typically* work top-down.

If, despite all this, unexpected results still ensue, users can take care to “clean” redundant resetting settings with `\counterwithout`. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever-counter-resetters_seq` with the `counterresetby` option.

For the above reasons, since order matters, the `counterresetters` option can only be set by the full list of counters. In other words, users wanting to change this should take the initial value as their starting base.

The `zc@enclcnt` `zref` property, not included by default in the `main` property list, is provided for the purpose of easing the debugging of counter reset chains. So, by adding `\zref@addprop{main}{zc@enclcnt}` you can inspect what the values in the `zc@enclval` property correspond to.

```

2388 \seq_new:N \l__zrefclever_counter_resetters_seq
2389 \keys_define:nn { zref-clever/label }
2390 {
2391   counterresetters .code:n =
2392     { \seq_set_from_clist:Nn \l__zrefclever_counter_resetters_seq {#1} } ,
2393   counterresetters .initial:n =
2394     {
2395       subparagraph ,
2396       paragraph ,
2397       subsubsection ,
2398       subsection ,
2399       section ,
2400       chapter ,
2401       part ,
2402     },
2403   counterresetters .value_required:n = true ,
2404 }

```

counterresetby option

`\l__zrefclever-counter_resetby_prop` is used by `__zrefclever-counter_reset_by:n` to populate the `zc@enclval` property, and stores a mapping from counters to the counter which resets each of them. This mapping has precedence in `__zrefclever-counter_reset_by:n` over the search through `\l__zrefclever-counter-resetters_seq`.

```

2405 \prop_new:N \l__zrefclever_counter_resetby_prop
2406 \keys_define:nn { zref-clever/label }
2407 {
2408   counterresetby .code:n =
2409     {
2410       \keyval_parse:nnn
2411     }

```

```

2412         \msg_warning:nnn { zref-clever }
2413         { key-requires-value } { counterresetby }
2414     }
2415     {
2416         \__zrefclever_prop_put_non_empty:Nnn
2417         \l__zrefclever_counter_resetby_prop
2418     }
2419     {#1}
2420 } ,
2421 counterresetby .value_required:n = true ,
2422 counterresetby .initial:n =
2423 {

```

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.

```

2424     enumii = enumi ,
2425     enumiii = enumii ,
2426     enumiv = enumiii ,
2427 } ,
2428 }

```

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.

```

2429 \tl_new:N \l__zrefclever_current_counter_tl
2430 \keys_define:nn { zref-clever/label }
2431 {
2432     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
2433     currentcounter .default:n = \@currentcounter ,
2434     currentcounter .initial:n = \@currentcounter ,
2435 }

```

labelhook option

```

2436 \bool_new:N \l__zrefclever_labelhook_bool
2437 \keys_define:nn { zref-clever/label }
2438 {
2439     labelhook .bool_set:N = \l__zrefclever_labelhook_bool ,
2440     labelhook .initial:n = true ,
2441     labelhook .default:n = true ,
2442 }

```

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>.

```

2443 \AddToHookWithArguments { label }
2444 {
2445     \bool_if:NT \l__zrefclever_labelhook_bool

```



```

2446     { \zref@wrapper@babel \zref@label {#1} }
2447   }

nocompat option

2448 \bool_new:N \g__zrefclever_nocompat_bool
2449 \seq_new:N \g__zrefclever_nocompat_modules_seq
2450 \keys_define:nn { zref-clever/reference }
2451 {
2452   nocompat .code:n =
2453   {
2454     \tl_if_empty:nTF {#1}
2455     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
2456     {
2457       \clist_map_inline:nn {#1}
2458       {
2459         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
2460         {
2461           \seq_gput_right:Nn
2462             \g__zrefclever_nocompat_modules_seq {##1}
2463         }
2464       }
2465     } ,
2466   }
2467 }
2468 \AddToHook { begindocument }
2469 {
2470   \keys_define:nn { zref-clever/reference }
2471   {
2472     nocompat .code:n =
2473     {
2474       \msg_warning:nnn { zref-clever }
2475       { option-preamble-only } { nocompat }
2476     }
2477   }
2478 }
2479 \AtEndOfPackage
2480 {
2481   \AddToHook { begindocument }
2482   {
2483     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
2484     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
2485   }
2486 }

```

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

```

    \__zrefclever_compat_module:nn {<module>} {<code>}
2487 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
2488 {
2489   \AddToHook { begindocument }
2490   {
2491     \bool_if:NF \g__zrefclever_nocompat_bool
2492     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
2493     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
2494   }
2495 }

```

(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.

```

2496 \seq_map_inline:Nn
2497 \g__zrefclever_rf_opts_tl_reference_seq
2498 {
2499   \keys_define:nn { zref-clever/reference }
2500   {
2501     #1 .default:o = \c_novalue_tl ,
2502     #1 .code:n =
2503     {
2504       \tl_if_novalue:nTF {##1}
2505       {
2506         \__zrefclever_opt_tl_unset:c
2507         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2508       }
2509       {
2510         \__zrefclever_opt_tl_set:cn
2511         { \__zrefclever_opt_varname_general:nn {#1} { tl } }
2512         {##1}
2513       }
2514     } ,
2515   }
2516 }
2517 \keys_define:nn { zref-clever/reference }
2518 {
2519   refpre .code:n =
2520   {
2521     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2522     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2523     { refpre } { refbounds }
2524   } ,
2525   refpos .code:n =
2526   {
2527     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2528     \msg_warning:nmmm { zref-clever }{ option-deprecated }
2529     { refpos } { refbounds }
2530   } ,

```

```

2531   preref .code:n =
2532     {
2533       % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2534       \msg_warning:nnnn { zref-clever }{ option-deprecated }
2535       { preref } { refbounds }
2536     } ,
2537   postref .code:n =
2538     {
2539       % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2540       \msg_warning:nnnn { zref-clever }{ option-deprecated }
2541       { postref } { refbounds }
2542     } ,
2543   }
2544 \seq_map_inline:Nn
2545   \g__zrefclever_rf_opts_seq_refbounds_seq
2546   {
2547     \keys_define:nn { zref-clever/reference }
2548     {
2549       #1 .default:o = \c_novalue_tl ,
2550       #1 .code:n =
2551         {
2552           \tl_if_novalue:nTF {##1}
2553             {
2554               \__zrefclever_opt_seq_unset:c
2555               { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2556             }
2557             {
2558               \seq_clear:N \l__zrefclever_tmpa_seq
2559               \__zrefclever_opt_seq_set_clist_split:Nn
2560               \l__zrefclever_tmpa_seq {##1}
2561               \bool_lazy_or:nnTF
2562               { \tl_if_empty_p:n {##1} }
2563               {
2564                 \int_compare_p:nNn
2565                 { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2566               }
2567               {
2568                 \__zrefclever_opt_seq_set_eq:cN
2569                 { \__zrefclever_opt_varname_general:nn {#1} { seq } }
2570                 \l__zrefclever_tmpa_seq
2571               }
2572               {
2573                 \msg_warning:nnee { zref-clever }
2574                 { refbounds-must-be-four }
2575                 {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2576               }
2577             }
2578           } ,
2579         }
2580   }
2581 \seq_map_inline:Nn
2582   \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2583   {
2584     \keys_define:nn { zref-clever/reference }

```

```

2585     {
2586       #1 .choice: ,
2587       #1 / true .code:n =
2588         {
2589           \__zrefclever_opt_bool_set_true:c
2590           { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2591         } ,
2592       #1 / false .code:n =
2593         {
2594           \__zrefclever_opt_bool_set_false:c
2595           { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2596         } ,
2597       #1 / unset .code:n =
2598         {
2599           \__zrefclever_opt_bool_unset:c
2600           { \__zrefclever_opt_varname_general:nn {#1} { bool } }
2601         } ,
2602       #1 .default:n = true ,
2603       no #1 .meta:n = { #1 = false } ,
2604       no #1 .value_forbidden:n = true ,
2605     }
2606 }

```

Package options

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

```

2607 \keys_define:nn { }
2608 {
2609   zref-clever/zcsetup .inherit:n =
2610     {
2611       zref-clever/label ,
2612       zref-clever/reference ,
2613     }
2614 }

```

`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.

```

2615 \bool_lazy_and:nnT
2616 { \tl_if_exist_p:c { opt@ zref-clever.sty } }
2617 { ! \tl_if_empty_p:c { opt@ zref-clever.sty } }
2618 { \msg_warning:nn { zref-clever } { load-time-options } }

```

5 Configuration

5.1 `\zcsetup`

`\zcsetup` Provide `\zcsetup`.

```
\zcsetup{options}  
  
2619 \NewDocumentCommand \zcsetup { m }  
2620 { \__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}  
  
2621 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1  
2622 { \keys_set:nn { zref-clever/zcsetup } {#1} }  
2623 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { e }
```

(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}  
  
2624 \NewDocumentCommand \zcRefTypeSetup { m m }  
2625 {  
2626 \tl_set:Nn \l__zrefclever_setup_type_tl {#1}  
2627 \keys_set:nn { zref-clever/typesetup } {#2}  
2628 \tl_clear:N \l__zrefclever_setup_type_tl  
2629 }  
  
(End of definition for \zcRefTypeSetup.)  
  
2630 \seq_map_inline:Nn  
2631 \g__zrefclever_rf_opts_tl_not_type_specific_seq  
2632 {  
2633 \keys_define:nn { zref-clever/typesetup }  
2634 {  
2635 #1 .code:n =  
2636 {  
2637 \msg_warning:nnn { zref-clever }  
2638 { option-not-type-specific } {#1}  
2639 } ,  
2640 }  
2641 }  
2642 \seq_map_inline:Nn
```

```

2643 \g__zrefclever_rf_opts_tl_typesetup_seq
2644 {
2645   \keys_define:nn { zref-clever/typesetup }
2646   {
2647     #1 .default:o = \c_novalue_tl ,
2648     #1 .code:n =
2649     {
2650       \tl_if_novalue:nTF {##1}
2651       {
2652         \__zrefclever_opt_tl_unset:c
2653         {
2654           \__zrefclever_opt_varname_type:enn
2655           { \l__zrefclever_setup_type_tl } {#1} { t1 }
2656         }
2657       }
2658       {
2659         \__zrefclever_opt_tl_set:cn
2660         {
2661           \__zrefclever_opt_varname_type:enn
2662           { \l__zrefclever_setup_type_tl } {#1} { t1 }
2663         }
2664         {##1}
2665       }
2666     } ,
2667   }
2668 }
2669 \keys_define:nn { zref-clever/typesetup }
2670 {
2671   endrange .code:n =
2672   {
2673     \str_case:nnF {#1}
2674     {
2675       { ref }
2676       {
2677         \__zrefclever_opt_tl_clear:c
2678         {
2679           \__zrefclever_opt_varname_type:enn
2680           { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2681         }
2682         \__zrefclever_opt_tl_clear:c
2683         {
2684           \__zrefclever_opt_varname_type:enn
2685           { \l__zrefclever_setup_type_tl } { endrangeprop } { t1 }
2686         }
2687       }
2688     }
2689     { stripprefix }
2690     {
2691       \__zrefclever_opt_tl_set:cn
2692       {
2693         \__zrefclever_opt_varname_type:enn
2694         { \l__zrefclever_setup_type_tl } { endrangefunc } { t1 }
2695       }
2696       { __zrefclever_get_endrange_stripprefix }

```

```

2697     \_zrefclever_opt_t1_clear:c
2698     {
2699         \_zrefclever_opt_varname_type:enn
2700         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2701     }
2702 }
2703
2704 { pagecomp }
2705 {
2706     \_zrefclever_opt_t1_set:cn
2707     {
2708         \_zrefclever_opt_varname_type:enn
2709         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2710     }
2711     { \_zrefclever_get_endrange_pagecomp }
2712     \_zrefclever_opt_t1_clear:c
2713     {
2714         \_zrefclever_opt_varname_type:enn
2715         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2716     }
2717 }
2718
2719 { pagecomp2 }
2720 {
2721     \_zrefclever_opt_t1_set:cn
2722     {
2723         \_zrefclever_opt_varname_type:enn
2724         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2725     }
2726     { \_zrefclever_get_endrange_pagecomptwo }
2727     \_zrefclever_opt_t1_clear:c
2728     {
2729         \_zrefclever_opt_varname_type:enn
2730         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2731     }
2732 }
2733
2734 { unset }
2735 {
2736     \_zrefclever_opt_t1_unset:c
2737     {
2738         \_zrefclever_opt_varname_type:enn
2739         { \l_zrefclever_setup_type_t1 } { endrangefunc } { t1 }
2740     }
2741     \_zrefclever_opt_t1_unset:c
2742     {
2743         \_zrefclever_opt_varname_type:enn
2744         { \l_zrefclever_setup_type_t1 } { endrangeprop } { t1 }
2745     }
2746 }
2747 }
2748 {
2749     \t1_if_empty:nTF {#1}
2750     {

```

```

2751         \msg_warning:nnn { zref-clever }
2752         { endrange-property-undefined } {#1}
2753     }
2754     {
2755         \zref@ifpropundefined {#1}
2756         {
2757             \msg_warning:nnn { zref-clever }
2758             { endrange-property-undefined } {#1}
2759         }
2760         {
2761             \__zrefclever_opt_tl_set:cn
2762             {
2763                 \__zrefclever_opt_varname_type:enn
2764                 { \l__zrefclever_setup_type_tl }
2765                 { endrangefunc } { tl }
2766             }
2767             { __zrefclever_get_endrange_property }
2768             \__zrefclever_opt_tl_set:cn
2769             {
2770                 \__zrefclever_opt_varname_type:enn
2771                 { \l__zrefclever_setup_type_tl }
2772                 { endrangeprop } { tl }
2773             }
2774             {#1}
2775         }
2776     }
2777 }
2778 } ,
2779 endrange .value_required:n = true ,
2780 }
2781 \keys_define:nn { zref-clever/typesetup }
2782 {
2783     refpre .code:n =
2784     {
2785         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2786         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2787         { refpre } { rebounds }
2788     } ,
2789     refpos .code:n =
2790     {
2791         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
2792         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2793         { refpos } { rebounds }
2794     } ,
2795     preref .code:n =
2796     {
2797         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2798         \msg_warning:nnnn { zref-clever }{ option-deprecated }
2799         { preref } { rebounds }
2800     } ,
2801     postref .code:n =
2802     {
2803         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
2804         \msg_warning:nnnn { zref-clever }{ option-deprecated }

```



```

2805         { postref } { refbounds }
2806     } ,
2807 }
2808 \seq_map_inline:Nn
2809 \g__zrefclever_rf_opts_seq_refbounds_seq
2810 {
2811     \keys_define:nn { zref-clever/typesetup }
2812     {
2813         #1 .default:o = \c_novalue_tl ,
2814         #1 .code:n =
2815         {
2816             \tl_if_novalue:nTF {##1}
2817             {
2818                 \__zrefclever_opt_seq_unset:c
2819                 {
2820                     \__zrefclever_opt_varname_type:enn
2821                     { \l__zrefclever_setup_type_tl } {#1} { seq }
2822                 }
2823             }
2824             {
2825                 \seq_clear:N \l__zrefclever_tmpa_seq
2826                 \__zrefclever_opt_seq_set_clist_split:Nn
2827                 \l__zrefclever_tmpa_seq {##1}
2828                 \bool_lazy_or:nnTF
2829                 { \tl_if_empty_p:n {##1} }
2830                 {
2831                     \int_compare_p:nNn
2832                     { \seq_count:N \l__zrefclever_tmpa_seq } = { 4 }
2833                 }
2834                 {
2835                     \__zrefclever_opt_seq_set_eq:cN
2836                     {
2837                         \__zrefclever_opt_varname_type:enn
2838                         { \l__zrefclever_setup_type_tl } {#1} { seq }
2839                     }
2840                     \l__zrefclever_tmpa_seq
2841                 }
2842                 {
2843                     \msg_warning:nnee { zref-clever }
2844                     { refbounds-must-be-four }
2845                     {#1} { \seq_count:N \l__zrefclever_tmpa_seq }
2846                 }
2847             }
2848         } ,
2849     }
2850 }
2851 \seq_map_inline:Nn
2852 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
2853 {
2854     \keys_define:nn { zref-clever/typesetup }
2855     {
2856         #1 .choice: ,
2857         #1 / true .code:n =
2858         {

```

```

2859         \_zrefclever_opt_bool_set_true:c
2860         {
2861             \_zrefclever_opt_varname_type:enn
2862             { \l__zrefclever_setup_type_t1 }
2863             {#1} { bool }
2864         }
2865     } ,
2866 #1 / false .code:n =
2867 {
2868     \_zrefclever_opt_bool_set_false:c
2869     {
2870         \_zrefclever_opt_varname_type:enn
2871         { \l__zrefclever_setup_type_t1 }
2872         {#1} { bool }
2873     }
2874 } ,
2875 #1 / unset .code:n =
2876 {
2877     \_zrefclever_opt_bool_unset:c
2878     {
2879         \_zrefclever_opt_varname_type:enn
2880         { \l__zrefclever_setup_type_t1 }
2881         {#1} { bool }
2882     }
2883 } ,
2884 #1 .default:n = true ,
2885 no #1 .meta:n = { #1 = false } ,
2886 no #1 .value_forbidden:n = true ,
2887 }
2888 }

```

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 `\langle options \rangle` 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}
2889 \NewDocumentCommand \zcLanguageSetup { m m }
2890 {
2891     \group_begin:
2892     \_zrefclever_language_if_declared:nTF {#1}
2893     {
2894         \tl_clear:N \l__zrefclever_setup_type_t1
2895         \tl_set:Nn \l__zrefclever_setup_language_t1 {#1}
2896         \_zrefclever_opt_seq_get:cNF
2897         {
2898             \_zrefclever_opt_varname_language:nnn
2899             {#1} { declension } { seq }

```

```

2900     }
2901     \l__zrefclever_lang_declension_seq
2902     { \seq_clear:N \l__zrefclever_lang_declension_seq }
2903 \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2904     { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
2905     {
2906       \seq_get_left:NN \l__zrefclever_lang_declension_seq
2907         \l__zrefclever_lang_decl_case_tl
2908     }
2909 \__zrefclever_opt_seq_get:cNF
2910     {
2911       \__zrefclever_opt_varname_language:nmn
2912         {#1} { gender } { seq }
2913     }
2914 \l__zrefclever_lang_gender_seq
2915     { \seq_clear:N \l__zrefclever_lang_gender_seq }
2916 \keys_set:nn { zref-clever/langsetup } {#2}
2917 }
2918 { \msg_warning:nmn { zref-clever } { unknown-language-setup } {#1} }
2919 \group_end:
2920 }
2921 \@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.

```

2922 \keys_define:nn { zref-clever/langsetup }
2923     {
2924       type .code:n =
2925         {
2926           \tl_if_empty:nTF {#1}
2927             { \tl_clear:N \l__zrefclever_setup_type_tl }
2928             { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
2929         } ,
2930
2931       case .code:n =
2932         {
2933           \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
2934             {
2935               \msg_warning:nnee { zref-clever } { language-no-decl-setup }
2936                 { \l__zrefclever_setup_language_tl } {#1}
2937             }
2938             {
2939               \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
2940                 { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
2941                 {
2942                   \msg_warning:nnee { zref-clever } { unknown-decl-case }
2943                     {#1} { \l__zrefclever_setup_language_tl }
2944                   \seq_get_left:NN \l__zrefclever_lang_declension_seq
2945                     \l__zrefclever_lang_decl_case_tl
2946                 }
2947             }
2948         } ,
2949       case .value_required:n = true ,

```

```

2950
2951 gender .value_required:n = true ,
2952 gender .code:n =
2953 {
2954   \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
2955   {
2956     \msg_warning:nnee { zref-clever } { language-no-gender }
2957     { \l__zrefclever_setup_language_tl } { gender } {#1}
2958   }
2959   {
2960     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2961     {
2962       \msg_warning:nnn { zref-clever }
2963       { option-only-type-specific } { gender }
2964     }
2965     {
2966       \seq_clear:N \l__zrefclever_tmpa_seq
2967       \clist_map_inline:nn {#1}
2968       {
2969         \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
2970         { \seq_put_right:Nn \l__zrefclever_tmpa_seq {##1} }
2971         {
2972           \msg_warning:nnee { zref-clever }
2973           { gender-not-declared }
2974           { \l__zrefclever_setup_language_tl } {##1}
2975         }
2976       }
2977       \__zrefclever_opt_seq_gset_eq:cN
2978       {
2979         \__zrefclever_opt_varname_lang_type:een
2980         { \l__zrefclever_setup_language_tl }
2981         { \l__zrefclever_setup_type_tl }
2982         { gender }
2983         { seq }
2984       }
2985       \l__zrefclever_tmpa_seq
2986     }
2987   } ,
2988 }
2989 }
2990 \seq_map_inline:Nn
2991 \g__zrefclever_rf_opts_tl_not_type_specific_seq
2992 {
2993   \keys_define:nn { zref-clever/langsetup }
2994   {
2995     #1 .value_required:n = true ,
2996     #1 .code:n =
2997     {
2998       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
2999       {
3000         \__zrefclever_opt_tl_gset:cn
3001         {
3002           \__zrefclever_opt_varname_lang_default:enn
3003           { \l__zrefclever_setup_language_tl } {#1} { tl }

```

```

3004         }
3005         {##1}
3006     }
3007     {
3008         \msg_warning:nnn { zref-clever }
3009         { option-not-type-specific } {#1}
3010     }
3011 } ,
3012 }
3013 }
3014 \seq_map_inline:Nn
3015 \g__zrefclever_rf_opts_tl_maybe_type_specific_seq
3016 {
3017     \keys_define:nn { zref-clever/langsetup }
3018     {
3019         #1 .value_required:n = true ,
3020         #1 .code:n =
3021         {
3022             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3023             {
3024                 \__zrefclever_opt_tl_gset:cn
3025                 {
3026                     \__zrefclever_opt_varname_lang_default:enn
3027                     { \l__zrefclever_setup_language_tl } {#1} { tl }
3028                 }
3029                 {##1}
3030             }
3031             {
3032                 \__zrefclever_opt_tl_gset:cn
3033                 {
3034                     \__zrefclever_opt_varname_lang_type:eenn
3035                     { \l__zrefclever_setup_language_tl }
3036                     { \l__zrefclever_setup_type_tl }
3037                     {#1} { tl }
3038                 }
3039                 {##1}
3040             }
3041         } ,
3042     }
3043 }
3044 \keys_define:nn { zref-clever/langsetup }
3045 {
3046     endrange .value_required:n = true ,
3047     endrange .code:n =
3048     {
3049         \str_case:nnF {#1}
3050         {
3051             { ref }
3052             {
3053                 \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3054                 {
3055                     \__zrefclever_opt_tl_gclear:c
3056                     {
3057                         \__zrefclever_opt_varname_lang_default:enn

```

```

3058         { \l__zrefclever_setup_language_tl }
3059         { endrangefunc } { tl }
3060     }
3061 \__zrefclever_opt_tl_gclear:c
3062 {
3063     \__zrefclever_opt_varname_lang_default:enn
3064     { \l__zrefclever_setup_language_tl }
3065     { endrangeprop } { tl }
3066 }
3067 }
3068 {
3069     \__zrefclever_opt_tl_gclear:c
3070     {
3071         \__zrefclever_opt_varname_lang_type:eenn
3072         { \l__zrefclever_setup_language_tl }
3073         { \l__zrefclever_setup_type_tl }
3074         { endrangefunc } { tl }
3075     }
3076     \__zrefclever_opt_tl_gclear:c
3077     {
3078         \__zrefclever_opt_varname_lang_type:eenn
3079         { \l__zrefclever_setup_language_tl }
3080         { \l__zrefclever_setup_type_tl }
3081         { endrangeprop } { tl }
3082     }
3083 }
3084 }
3085
3086 { stripprefix }
3087 {
3088     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3089     {
3090         \__zrefclever_opt_tl_gset:cn
3091         {
3092             \__zrefclever_opt_varname_lang_default:enn
3093             { \l__zrefclever_setup_language_tl }
3094             { endrangefunc } { tl }
3095         }
3096         { __zrefclever_get_endrange_stripprefix }
3097         \__zrefclever_opt_tl_gclear:c
3098         {
3099             \__zrefclever_opt_varname_lang_default:enn
3100             { \l__zrefclever_setup_language_tl }
3101             { endrangeprop } { tl }
3102         }
3103     }
3104     {
3105         \__zrefclever_opt_tl_gset:cn
3106         {
3107             \__zrefclever_opt_varname_lang_type:eenn
3108             { \l__zrefclever_setup_language_tl }
3109             { \l__zrefclever_setup_type_tl }
3110             { endrangefunc } { tl }
3111         }

```

```

3112         { __zrefclever_get_endrange_stripprefix }
3113     \__zrefclever_opt_tl_gclear:c
3114     {
3115         \__zrefclever_opt_varname_lang_type:eenn
3116         { \l__zrefclever_setup_language_tl }
3117         { \l__zrefclever_setup_type_tl }
3118         { endrangeprop } { tl }
3119     }
3120 }
3121 }
3122
3123 { pagecomp }
3124 {
3125     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3126     {
3127         \__zrefclever_opt_tl_gset:cn
3128         {
3129             \__zrefclever_opt_varname_lang_default:eenn
3130             { \l__zrefclever_setup_language_tl }
3131             { endrangefunc } { tl }
3132         }
3133         { __zrefclever_get_endrange_pagecomp }
3134         \__zrefclever_opt_tl_gclear:c
3135         {
3136             \__zrefclever_opt_varname_lang_default:eenn
3137             { \l__zrefclever_setup_language_tl }
3138             { endrangeprop } { tl }
3139         }
3140     }
3141     {
3142         \__zrefclever_opt_tl_gset:cn
3143         {
3144             \__zrefclever_opt_varname_lang_type:eenn
3145             { \l__zrefclever_setup_language_tl }
3146             { \l__zrefclever_setup_type_tl }
3147             { endrangefunc } { tl }
3148         }
3149         { __zrefclever_get_endrange_pagecomp }
3150         \__zrefclever_opt_tl_gclear:c
3151         {
3152             \__zrefclever_opt_varname_lang_type:eenn
3153             { \l__zrefclever_setup_language_tl }
3154             { \l__zrefclever_setup_type_tl }
3155             { endrangeprop } { tl }
3156         }
3157     }
3158 }
3159
3160 { pagecomp2 }
3161 {
3162     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3163     {
3164         \__zrefclever_opt_tl_gset:cn
3165         {

```

```

3166         \_zrefclever_opt_varname_lang_default:enn
3167         { \l__zrefclever_setup_language_tl }
3168         { endrangefunc } { tl }
3169     }
3170     { __zrefclever_get_endrange_pagecomptwo }
3171 \_zrefclever_opt_tl_gclear:c
3172     {
3173         \_zrefclever_opt_varname_lang_default:enn
3174         { \l__zrefclever_setup_language_tl }
3175         { endrangeprop } { tl }
3176     }
3177 }
3178 {
3179     \_zrefclever_opt_tl_gset:cn
3180     {
3181         \_zrefclever_opt_varname_lang_type:eenn
3182         { \l__zrefclever_setup_language_tl }
3183         { \l__zrefclever_setup_type_tl }
3184         { endrangefunc } { tl }
3185     }
3186     { __zrefclever_get_endrange_pagecomptwo }
3187 \_zrefclever_opt_tl_gclear:c
3188     {
3189         \_zrefclever_opt_varname_lang_type:eenn
3190         { \l__zrefclever_setup_language_tl }
3191         { \l__zrefclever_setup_type_tl }
3192         { endrangeprop } { tl }
3193     }
3194 }
3195 }
3196 }
3197 {
3198     \tl_if_empty:nTF {#1}
3199     {
3200         \msg_warning:nnn { zref-clever }
3201         { endrange-property-undefined } {#1}
3202     }
3203     {
3204         \zref@ifpropundefined {#1}
3205         {
3206             \msg_warning:nnn { zref-clever }
3207             { endrange-property-undefined } {#1}
3208         }
3209         {
3210             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3211             {
3212                 \_zrefclever_opt_tl_gset:cn
3213                 {
3214                     \_zrefclever_opt_varname_lang_default:enn
3215                     { \l__zrefclever_setup_language_tl }
3216                     { endrangefunc } { tl }
3217                 }
3218                 { __zrefclever_get_endrange_property }
3219                 \_zrefclever_opt_tl_gset:cn

```



```

3220         {
3221             \__zrefclever_opt_varname_lang_default:enn
3222             { \l__zrefclever_setup_language_tl }
3223             { endrangeprop } { t1 }
3224         }
3225         {#1}
3226     }
3227     {
3228         \__zrefclever_opt_tl_gset:cn
3229         {
3230             \__zrefclever_opt_varname_lang_type:eenn
3231             { \l__zrefclever_setup_language_tl }
3232             { \l__zrefclever_setup_type_tl }
3233             { endrangefunc } { t1 }
3234         }
3235         { \__zrefclever_get_endrange_property }
3236         \__zrefclever_opt_tl_gset:cn
3237         {
3238             \__zrefclever_opt_varname_lang_type:eenn
3239             { \l__zrefclever_setup_language_tl }
3240             { \l__zrefclever_setup_type_tl }
3241             { endrangeprop } { t1 }
3242         }
3243         {#1}
3244     }
3245 }
3246 }
3247 }
3248 } ,
3249 }
3250 \keys_define:nn { zref-clever/langsetup }
3251 {
3252     refpre .code:n =
3253     {
3254         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3255         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3256         { refpre } { rebounds }
3257     } ,
3258     refpos .code:n =
3259     {
3260         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
3261         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3262         { refpos } { rebounds }
3263     } ,
3264     preref .code:n =
3265     {
3266         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3267         \msg_warning:nnnn { zref-clever }{ option-deprecated }
3268         { preref } { rebounds }
3269     } ,
3270     postref .code:n =
3271     {
3272         % NOTE Option deprecated in 2022-01-14 for v0.2.0-alpha.
3273         \msg_warning:nnnn { zref-clever }{ option-deprecated }

```

```

3274         { postref } { refbounds }
3275     } ,
3276 }
3277 \seq_map_inline:Nn
3278 \g__zrefclever_rf_opts_tl_type_names_seq
3279 {
3280     \keys_define:nn { zref-clever/langsetup }
3281     {
3282         #1 .value_required:n = true ,
3283         #1 .code:n =
3284         {
3285             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3286             {
3287                 \msg_warning:nnn { zref-clever }
3288                 { option-only-type-specific } {#1}
3289             }
3290             {
3291                 \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
3292                 {
3293                     \__zrefclever_opt_tl_gset:cn
3294                     {
3295                         \__zrefclever_opt_varname_lang_type:een
3296                         { \l__zrefclever_setup_language_tl }
3297                         { \l__zrefclever_setup_type_tl }
3298                         {#1} { tl }
3299                     }
3300                     {##1}
3301                 }
3302                 {
3303                     \__zrefclever_opt_tl_gset:cn
3304                     {
3305                         \__zrefclever_opt_varname_lang_type:een
3306                         { \l__zrefclever_setup_language_tl }
3307                         { \l__zrefclever_setup_type_tl }
3308                         { \l__zrefclever_lang_decl_case_tl - #1 }
3309                         { tl }
3310                     }
3311                     {##1}
3312                 }
3313             }
3314         } ,
3315     }
3316 }
3317 \seq_map_inline:Nn
3318 \g__zrefclever_rf_opts_seq_refbounds_seq
3319 {
3320     \keys_define:nn { zref-clever/langsetup }
3321     {
3322         #1 .value_required:n = true ,
3323         #1 .code:n =
3324         {
3325             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3326             {
3327                 \seq_gclear:N \g__zrefclever_tmpa_seq

```

```

3328     \__zrefclever_opt_seq_gset_clist_split:Nn
3329     \g__zrefclever_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__zrefclever_tmpa_seq } = { 4 }
3335   }
3336   {
3337     \__zrefclever_opt_seq_gset_eq:cN
3338     {
3339       \__zrefclever_opt_varname_lang_default:enn
3340       { \l__zrefclever_setup_language_tl }
3341       {##1} { seq }
3342     }
3343     \g__zrefclever_tmpa_seq
3344   }
3345   {
3346     \msg_warning:nnee { zref-clever }
3347     { refbounds-must-be-four }
3348     {##1} { \seq_count:N \g__zrefclever_tmpa_seq }
3349   }
3350 }
3351 {
3352 \seq_gclear:N \g__zrefclever_tmpa_seq
3353 \__zrefclever_opt_seq_gset_clist_split:Nn
3354 \g__zrefclever_tmpa_seq {##1}
3355 \bool_lazy_or:nnTF
3356   { \tl_if_empty_p:n {##1} }
3357   {
3358     \int_compare_p:nNn
3359     { \seq_count:N \g__zrefclever_tmpa_seq } = { 4 }
3360   }
3361   {
3362     \__zrefclever_opt_seq_gset_eq:cN
3363     {
3364       \__zrefclever_opt_varname_lang_type:enn
3365       { \l__zrefclever_setup_language_tl }
3366       { \l__zrefclever_setup_type_tl } {##1} { seq }
3367     }
3368     \g__zrefclever_tmpa_seq
3369   }
3370   {
3371     \msg_warning:nnee { zref-clever }
3372     { refbounds-must-be-four }
3373     {##1} { \seq_count:N \g__zrefclever_tmpa_seq }
3374   }
3375 }
3376 } ,
3377 }
3378 }
3379 \seq_map_inline:Nn
3380 \g__zrefclever_rf_opts_bool_maybe_type_specific_seq
3381 {

```

```

3382 \keys_define:nn { zref-clever/langsetup }
3383 {
3384   #1 .choice: ,
3385   #1 / true .code:n =
3386   {
3387     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3388     {
3389       \__zrefclever_opt_bool_gset_true:c
3390       {
3391         \__zrefclever_opt_varname_lang_default:enn
3392         { \l__zrefclever_setup_language_tl }
3393         {#1} { bool }
3394       }
3395     }
3396     {
3397       \__zrefclever_opt_bool_gset_true:c
3398       {
3399         \__zrefclever_opt_varname_lang_type:eenn
3400         { \l__zrefclever_setup_language_tl }
3401         { \l__zrefclever_setup_type_tl }
3402         {#1} { bool }
3403       }
3404     }
3405   } ,
3406   #1 / false .code:n =
3407   {
3408     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
3409     {
3410       \__zrefclever_opt_bool_gset_false:c
3411       {
3412         \__zrefclever_opt_varname_lang_default:enn
3413         { \l__zrefclever_setup_language_tl }
3414         {#1} { bool }
3415       }
3416     }
3417     {
3418       \__zrefclever_opt_bool_gset_false:c
3419       {
3420         \__zrefclever_opt_varname_lang_type:eenn
3421         { \l__zrefclever_setup_language_tl }
3422         { \l__zrefclever_setup_type_tl }
3423         {#1} { bool }
3424       }
3425     }
3426   } ,
3427   #1 .default:n = true ,
3428   no #1 .meta:n = { #1 = false } ,
3429   no #1 .value_forbidden:n = true ,
3430 }
3431 }

```

6 User interface

6.1 `\zcref`

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

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

```
3432 \NewDocumentCommand \zcref { s O { } m }
3433 { \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>}
```

```
3434 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
3435 {
3436   \group_begin:
```

Set options.

```
3437   \keys_set:nn { zref-clever/reference } {#3}
```

Store arguments values.

```
3438   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
3439   \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:e` does nothing if the language file is already loaded.

```
3440   \__zrefclever_provide_langfile:e { \l__zrefclever_ref_language_tl }
```

Process language settings.

```
3441   \__zrefclever_process_language_settings:
```

Integration with `zref-check`.

```
3442   \bool_lazy_and:nnT
3443     { \l__zrefclever_zrefcheck_available_bool }
3444     { \l__zrefclever_zcref_with_check_bool }
3445     { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
3446   \bool_lazy_or:nnT
3447     { \l__zrefclever_typeset_sort_bool }
3448     { \l__zrefclever_typeset_range_bool }
3449     { \__zrefclever_sort_labels: }
```

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

```
3450   \group_begin:
3451   \l__zrefclever_ref_typeset_font_tl
3452   \__zrefclever_typeset_refs:
3453   \group_end:
```

Typeset note.

```
3454     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
3455     {
3456         \__zrefclever_get_rf_opt_tl:neeN { noteseq }
3457         { \l__zrefclever_label_type_a_tl }
3458         { \l__zrefclever_ref_language_tl }
3459         \l__zrefclever_tmpa_tl
3460         \l__zrefclever_tmpa_tl
3461         \l__zrefclever_zcref_note_tl
3462     }
```

Integration with zref-check.

```
3463     \bool_lazy_and:nnT
3464     { \l__zrefclever_zrefcheck_available_bool }
3465     { \l__zrefclever_zcref_with_check_bool }
3466     {
3467         \zrefcheck_zcref_end_label_maybe:
3468         \zrefcheck_zcref_run_checks_on_labels:n
3469         { \l__zrefclever_zcref_labels_seq }
3470     }
```

Integration with mathtools.

```
3471     \bool_if:NT \l__zrefclever_mathtools_loaded_bool
3472     {
3473         \__zrefclever_mathtools_showonlyrefs:n
3474         { \l__zrefclever_zcref_labels_seq }
3475     }
3476     \group_end:
3477 }
```

(End of definition for __zrefclever_zcref:nnnn.)

```
\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool
```

```
3478 \seq_new:N \l__zrefclever_zcref_labels_seq
3479 \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>}
```

```
3480 \NewDocumentCommand \zcpageref { s O { } m }
3481 {
3482     \group_begin:
3483     \IfBooleanT {#1}
3484     { \bool_set_false:N \l__zrefclever_hyperlink_bool }
3485     \zcref [#2, ref = page] {#3}
3486     \group_end:
3487 }
```

(End of definition for \zcpageref.)

7 Sorting

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

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

`\l_zrefclever_label_type_b_tl`

`\l_zrefclever_label_enclval_a_tl` 3488 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 3489 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 3490 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 3491 `\tl_new:N \l__zrefclever_label_enclval_b_tl`
 3492 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`
 3493 `\tl_new:N \l__zrefclever_label_extdoc_b_tl`

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

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

3494 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End of definition for `\l__zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

3495 `\int_new:N \l__zrefclever_sort_prior_a_int`
 3496 `\int_new:N \l__zrefclever_sort_prior_b_int`

(End of definition for `\l__zrefclever_sort_prior_a_int` and `\l__zrefclever_sort_prior_b_int`.)

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

3497 `\seq_new:N \l__zrefclever_label_types_seq`

(End of definition for `\l__zrefclever_label_types_seq`.)

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

3498 `\cs_new_protected:Npn __zrefclever_sort_labels:`
 3499 `{`

Store label types sequence.

```

3500 \seq_clear:N \l__zrefclever_label_types_seq
3501 \tl_if_eq:NnF \l__zrefclever_ref_propserity_tl { page }
3502 {
3503   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
3504   \__zrefclever_label_type_put_new_right:n
3505 }

```

Sort.

```

3506 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
3507 {
3508   \zref@ifrefundefined {##1}
3509   {
3510     \zref@ifrefundefined {##2}
3511     {
3512       % Neither label is defined.
3513       \sort_return_same:
3514     }
3515     {
3516       % The second label is defined, but the first isn't, leave the
3517       % undefined first (to be more visible).
3518       \sort_return_same:
3519     }
3520   }
3521   {
3522     \zref@ifrefundefined {##2}
3523     {
3524       % The first label is defined, but the second isn't, bring the
3525       % second forward.
3526       \sort_return_swapped:
3527     }
3528     {
3529       % The interesting case: both labels are defined. References
3530       % to the "default" property or to the "page" are quite
3531       % different with regard to sorting, so we branch them here to
3532       % specialized functions.
3533       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3534       { \__zrefclever_sort_page:n {##1} {##2} }
3535       { \__zrefclever_sort_default:n {##1} {##2} }
3536     }
3537   }
3538 }
3539 }

```

(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}>}
3540 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
3541 {
3542   \__zrefclever_extract_default:Nnnn
3543   \l__zrefclever_label_type_a_tl {#1} { zc@type } { }
3544   \seq_if_in:NVF \l__zrefclever_label_types_seq
3545   \l__zrefclever_label_type_a_tl
3546   {
3547     \seq_put_right:NV \l__zrefclever_label_types_seq
3548     \l__zrefclever_label_type_a_tl
3549   }
3550 }

```

(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>}>
3551 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
3552 {
3553   \__zrefclever_extract_default:Nnnn
3554   \l__zrefclever_label_type_a_tl {#1} { zc@type } { zc@missingtype }
3555   \__zrefclever_extract_default:Nnnn
3556   \l__zrefclever_label_type_b_tl {#2} { zc@type } { zc@missingtype }
3557
3558   \tl_if_eq:NNTF
3559   \l__zrefclever_label_type_a_tl
3560   \l__zrefclever_label_type_b_tl
3561   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
3562   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
3563 }

```

(End of definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn    \__zrefclever_sort_default_same_type:nn <{<label a>} <{<label b>}>
3564 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
3565 {
3566   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
3567   {#1} { zc@enclval } { }
3568   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
3569   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
3570   {#2} { zc@enclval } { }
3571   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
3572   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3573   {#1} { externaldocument } { }
3574   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3575   {#2} { externaldocument } { }
3576
3577   \bool_set_false:N \l__zrefclever_sort_decided_bool

```

```

3578
3579 % First we check if there's any "external document" difference (coming
3580 % from `zref-xr') and, if so, sort based on that.
3581 \tl_if_eq:NNF
3582   \l__zrefclever_label_extdoc_a_tl
3583   \l__zrefclever_label_extdoc_b_tl
3584   {
3585     \bool_if:nTF
3586     {
3587       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3588       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3589     }
3590     {
3591       \bool_set_true:N \l__zrefclever_sort_decided_bool
3592       \sort_return_same:
3593     }
3594     {
3595       \bool_if:nTF
3596       {
3597         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
3598         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
3599       }
3600       {
3601         \bool_set_true:N \l__zrefclever_sort_decided_bool
3602         \sort_return_swapped:
3603       }
3604       {
3605         \bool_set_true:N \l__zrefclever_sort_decided_bool
3606         % Two different "external documents": last resort, sort by the
3607         % document name itself.
3608         \str_compare:eNeTF
3609         { \l__zrefclever_label_extdoc_b_tl } <
3610         { \l__zrefclever_label_extdoc_a_tl }
3611         { \sort_return_swapped: }
3612         { \sort_return_same: }
3613       }
3614     }
3615   }
3616
3617 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
3618 {
3619   \bool_if:nTF
3620   {
3621     % Both are empty: neither label has any (further) "enclosing
3622     % counters" (left).
3623     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
3624     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3625   }
3626   {
3627     \bool_set_true:N \l__zrefclever_sort_decided_bool
3628     \int_compare:nNnTF
3629     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3630     >
3631     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }

```

```

3632     { \sort_return_swapped: }
3633     { \sort_return_same:   }
3634 }
3635 {
3636   \bool_if:nTF
3637   {
3638     % `a' is empty (and `b' is not): `b' may be nested in `a'.
3639     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
3640   }
3641   {
3642     \bool_set_true:N \l__zrefclever_sort_decided_bool
3643     \int_compare:nNnTF
3644       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
3645       >
3646       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3647       { \sort_return_swapped: }
3648       { \sort_return_same:   }
3649   }
3650   {
3651     \bool_if:nTF
3652     {
3653       % `b' is empty (and `a' is not): `a' may be nested in `b'.
3654       \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
3655     }
3656     {
3657       \bool_set_true:N \l__zrefclever_sort_decided_bool
3658       \int_compare:nNnTF
3659         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3660         <
3661         { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
3662         { \sort_return_same:   }
3663         { \sort_return_swapped: }
3664     }
3665     {
3666       % Neither is empty: we can compare the values of the
3667       % current enclosing counter in the loop, if they are
3668       % equal, we are still in the loop, if they are not, a
3669       % sorting decision can be made directly.
3670       \int_compare:nNnTF
3671         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3672         =
3673         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
3674         {
3675           \tl_set:Ne \l__zrefclever_label_enclval_a_tl
3676             { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
3677           \tl_set:Ne \l__zrefclever_label_enclval_b_tl
3678             { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
3679         }
3680         {
3681           \bool_set_true:N \l__zrefclever_sort_decided_bool
3682           \int_compare:nNnTF
3683             { \tl_head:N \l__zrefclever_label_enclval_a_tl }
3684             >
3685             { \tl_head:N \l__zrefclever_label_enclval_b_tl }

```

```

3686             { \sort_return_swapped: }
3687             { \sort_return_same:   }
3688         }
3689     }
3690 }
3691 }
3692 }
3693 }

```

(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>}

```

```

3694 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
3695 {

```

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

```

3696     \int_zero:N \l__zrefclever_sort_prior_a_int
3697     \int_zero:N \l__zrefclever_sort_prior_b_int
3698     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
3699     {
3700         \tl_if_eq:nnTF {##2} {{othertypes}}
3701         {
3702             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
3703             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3704             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
3705             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3706         }
3707         {
3708             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
3709             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
3710             {
3711                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
3712                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
3713             }
3714         }
3715     }

```

Then do the actual sorting.

```

3716     \bool_if:nTF
3717     {
3718         \int_compare_p:nNn
3719         { \l__zrefclever_sort_prior_a_int } <
3720         { \l__zrefclever_sort_prior_b_int }
3721     }
3722     { \sort_return_same: }
3723     {
3724         \bool_if:nTF
3725         {
3726             \int_compare_p:nNn
3727             { \l__zrefclever_sort_prior_a_int } >
3728             { \l__zrefclever_sort_prior_b_int }
3729         }

```

```

3730     { \sort_return_swapped: }
3731     {
3732     % Sort priorities are equal: the type that occurs first in
3733     % `labels', as given by the user, is kept (or brought) forward.
3734     \seq_map_inline:Nn \l__zrefclever_label_types_seq
3735     {
3736         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
3737         { \seq_map_break:n { \sort_return_same: } }
3738         {
3739             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
3740             { \seq_map_break:n { \sort_return_swapped: } }
3741         }
3742     }
3743 }
3744 }
3745 }

```

(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)}
3746 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
3747 {
3748     \int_compare:nNnTF
3749     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
3750     >
3751     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
3752     { \sort_return_swapped: }
3753     { \sort_return_same: }
3754 }

```

(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` Auxiliary variables for `_zrefclever_typeset_refs`: main stack control.

`\l_zrefclever_typeset_last_bool` 3755 `\seq_new:N \l__zrefclever_typeset_labels_seq`

`\l_zrefclever_last_of_type_bool` 3756 `\bool_new:N \l__zrefclever_typeset_last_bool`

3757 `\bool_new:N \l__zrefclever_last_of_type_bool`

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

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

`\l_zrefclever_label_count_int` 3758 `\int_new:N \l__zrefclever_type_count_int`

`\l__zrefclever_ref_count_int` 3759 `\int_new:N \l__zrefclever_label_count_int`

3760 `\int_new:N \l__zrefclever_ref_count_int`

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

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

`\l__zrefclever_label_b_tl` 3761 `\tl_new:N \l__zrefclever_label_a_tl`

`\l_zrefclever_typeset_queue_prev_tl` 3762 `\tl_new:N \l__zrefclever_label_b_tl`

`\l_zrefclever_typeset_queue_curr_tl` 3763 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`

`\l_zrefclever_type_first_label_tl` 3764 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`

`\l__zrefclever_type_first_label_type_tl` 3765 `\tl_new:N \l__zrefclever_type_first_label_tl`

3766 `\tl_new:N \l__zrefclever_type_first_label_type_tl`

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

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

`\l_zrefclever_name_in_link_bool` 3767 `\tl_new:N \l__zrefclever_type_name_tl`

`\l_zrefclever_type_name_missing_bool` 3768 `\bool_new:N \l__zrefclever_name_in_link_bool`

`\l_zrefclever_name_format_tl` 3769 `\bool_new:N \l__zrefclever_type_name_missing_bool`

`\l_zrefclever_name_format_fallback_tl` 3770 `\tl_new:N \l__zrefclever_name_format_tl`

`\l_zrefclever_type_name_gender_seq` 3771 `\tl_new:N \l__zrefclever_name_format_fallback_tl`

3772 `\seq_new:N \l__zrefclever_type_name_gender_seq`

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

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

`\l_zrefclever_range_same_count_int` 3773 `\int_new:N \l__zrefclever_range_count_int`

`\l_zrefclever_range_beg_label_tl` 3774 `\int_new:N \l__zrefclever_range_same_count_int`

`\l_zrefclever_range_beg_is_first_bool` 3775 `\tl_new:N \l__zrefclever_range_beg_label_tl`

`\l_zrefclever_range_end_ref_tl` 3776 `\bool_new:N \l__zrefclever_range_beg_is_first_bool`

`\l_zrefclever_next_maybe_range_bool` 3777 `\tl_new:N \l__zrefclever_range_end_ref_tl`

`\l_zrefclever_next_is_same_bool` 3778 `\bool_new:N \l__zrefclever_next_maybe_range_bool`

3779 `\bool_new:N \l__zrefclever_next_is_same_bool`

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

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

```

\l__zrefclever_tlistsep_tl 3780 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_tlastsep_tl 3781 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_namesep_tl 3782 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_pairsep_tl 3783 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_listsep_tl 3784 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_lastsep_tl 3785 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_rangeseq_tl 3786 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_namefont_tl 3787 \tl_new:N \l__zrefclever_rangeseq_tl
\l__zrefclever_reffont_tl 3788 \tl_new:N \l__zrefclever_namefont_tl
  \l__zrefclever_endrangefunc_tl 3789 \tl_new:N \l__zrefclever_reffont_tl
  \l__zrefclever_endrangeprop_tl 3790 \tl_new:N \l__zrefclever_endrangefunc_tl
\l__zrefclever_cap_bool 3791 \tl_new:N \l__zrefclever_endrangeprop_tl
\l__zrefclever_abbrev_bool 3792 \bool_new:N \l__zrefclever_cap_bool
  \l__zrefclever_rangetopair_bool 3793 \bool_new:N \l__zrefclever_abbrev_bool
  3794 \bool_new:N \l__zrefclever_rangetopair_bool

```

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

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

```

\l__zrefclever_refbounds_first_sg_seq 3795 \seq_new:N \l__zrefclever_refbounds_first_seq
\l__zrefclever_refbounds_first_pb_seq 3796 \seq_new:N \l__zrefclever_refbounds_first_sg_seq
\l__zrefclever_refbounds_first_rb_seq 3797 \seq_new:N \l__zrefclever_refbounds_first_pb_seq
  \l__zrefclever_refbounds_mid_seq 3798 \seq_new:N \l__zrefclever_refbounds_first_rb_seq
  \l__zrefclever_refbounds_mid_re_seq 3799 \seq_new:N \l__zrefclever_refbounds_mid_seq
  \l__zrefclever_refbounds_last_seq 3800 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
  \l__zrefclever_refbounds_last_pe_seq 3801 \seq_new:N \l__zrefclever_refbounds_last_seq
  \l__zrefclever_refbounds_last_re_seq 3802 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_type_first_refbounds_seq 3803 \seq_new:N \l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_set_bool 3804 \seq_new:N \l__zrefclever_type_first_refbounds_seq
  3805 \seq_new:N \l__zrefclever_type_first_refbounds_seq
  3806 \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`.

```
3807 \bool_new:N \l__zrefclever_verbose_testing_bool
```

(End of definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```

3808 \cs_new_protected:Npn \__zrefclever_typeset_refs:
3809 {
3810   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
3811   \l__zrefclever_zcref_labels_seq
3812   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
3813   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
3814   \tl_clear:N \l__zrefclever_type_first_label_tl

```



```

3815 \tl_clear:N \l__zrefclever_type_first_label_type_tl
3816 \tl_clear:N \l__zrefclever_range_beg_label_tl
3817 \tl_clear:N \l__zrefclever_range_end_ref_tl
3818 \int_zero:N \l__zrefclever_label_count_int
3819 \int_zero:N \l__zrefclever_type_count_int
3820 \int_zero:N \l__zrefclever_ref_count_int
3821 \int_zero:N \l__zrefclever_range_count_int
3822 \int_zero:N \l__zrefclever_range_same_count_int
3823 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
3824 \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
3825
3826 % Get type block options (not type-specific).
3827 \__zrefclever_get_rf_opt_tl:neeN { tpairsep }
3828   { \l__zrefclever_label_type_a_tl }
3829   { \l__zrefclever_ref_language_tl }
3830   \l__zrefclever_tpairsep_tl
3831 \__zrefclever_get_rf_opt_tl:neeN { tlistsep }
3832   { \l__zrefclever_label_type_a_tl }
3833   { \l__zrefclever_ref_language_tl }
3834   \l__zrefclever_tlistsep_tl
3835 \__zrefclever_get_rf_opt_tl:neeN { tlastsep }
3836   { \l__zrefclever_label_type_a_tl }
3837   { \l__zrefclever_ref_language_tl }
3838   \l__zrefclever_tlastsep_tl
3839
3840 % Process label stack.
3841 \bool_set_false:N \l__zrefclever_typeset_last_bool
3842 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
3843   {
3844     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
3845     \l__zrefclever_label_a_tl
3846     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
3847       {
3848         \tl_clear:N \l__zrefclever_label_b_tl
3849         \bool_set_true:N \l__zrefclever_typeset_last_bool
3850       }
3851       {
3852         \seq_get_left:NN \l__zrefclever_typeset_labels_seq
3853         \l__zrefclever_label_b_tl
3854       }
3855
3856     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3857       {
3858         \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
3859         \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
3860       }
3861       {
3862         \__zrefclever_extract_default:NVnn
3863         \l__zrefclever_label_type_a_tl
3864         \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
3865         \__zrefclever_extract_default:NVnn
3866         \l__zrefclever_label_type_b_tl
3867         \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
3868       }

```

```

3869
3870 % First, we establish whether the "current label" (i.e. `a') is the
3871 % last one of its type. This can happen because the "next label"
3872 % (i.e. `b') is of a different type (or different definition status),
3873 % or because we are at the end of the list.
3874 \bool_if:NTF \l__zrefclever_typeset_last_bool
3875 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3876 {
3877   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3878   {
3879     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3880     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3881     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3882   }
3883   {
3884     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
3885     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3886     {
3887       % Neither is undefined, we must check the types.
3888       \tl_if_eq:NNTF
3889         \l__zrefclever_label_type_a_tl
3890         \l__zrefclever_label_type_b_tl
3891         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
3892         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
3893     }
3894   }
3895 }
3896
3897 % Handle warnings in case of reference or type undefined.
3898 % Test: `zc-typeset01.lvt': "Typeset refs: warn ref undefined"
3899 \zref@refused { \l__zrefclever_label_a_tl }
3900 % Test: `zc-typeset01.lvt': "Typeset refs: warn missing type"
3901 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
3902 {
3903   {
3904     \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
3905     {
3906       \msg_warning:nne { zref-clever } { missing-type }
3907       { \l__zrefclever_label_a_tl }
3908     }
3909     \zref@ifrefcontainsprop
3910     { \l__zrefclever_label_a_tl }
3911     { \l__zrefclever_ref_property_tl }
3912     { }
3913     {
3914       \msg_warning:nnee { zref-clever } { missing-property }
3915       { \l__zrefclever_ref_property_tl }
3916       { \l__zrefclever_label_a_tl }
3917     }
3918   }
3919 }
3920 % Get possibly type-specific separators, refbounds, font and other
3921 % options, once per type.
3922 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```

```

3923 {
3924   \__zrefclever_get_rf_opt_tl:neeN { namesep }
3925     { \l__zrefclever_label_type_a_tl }
3926     { \l__zrefclever_ref_language_tl }
3927     \l__zrefclever_namesep_tl
3928   \__zrefclever_get_rf_opt_tl:neeN { pairsep }
3929     { \l__zrefclever_label_type_a_tl }
3930     { \l__zrefclever_ref_language_tl }
3931     \l__zrefclever_pairsep_tl
3932   \__zrefclever_get_rf_opt_tl:neeN { listsep }
3933     { \l__zrefclever_label_type_a_tl }
3934     { \l__zrefclever_ref_language_tl }
3935     \l__zrefclever_listsep_tl
3936   \__zrefclever_get_rf_opt_tl:neeN { lastsep }
3937     { \l__zrefclever_label_type_a_tl }
3938     { \l__zrefclever_ref_language_tl }
3939     \l__zrefclever_lastsep_tl
3940   \__zrefclever_get_rf_opt_tl:neeN { rangesep }
3941     { \l__zrefclever_label_type_a_tl }
3942     { \l__zrefclever_ref_language_tl }
3943     \l__zrefclever_rangesep_tl
3944   \__zrefclever_get_rf_opt_tl:neeN { namefont }
3945     { \l__zrefclever_label_type_a_tl }
3946     { \l__zrefclever_ref_language_tl }
3947     \l__zrefclever_namefont_tl
3948   \__zrefclever_get_rf_opt_tl:neeN { reffont }
3949     { \l__zrefclever_label_type_a_tl }
3950     { \l__zrefclever_ref_language_tl }
3951     \l__zrefclever_reffont_tl
3952   \__zrefclever_get_rf_opt_tl:neeN { endrangefunc }
3953     { \l__zrefclever_label_type_a_tl }
3954     { \l__zrefclever_ref_language_tl }
3955     \l__zrefclever_endrangefunc_tl
3956   \__zrefclever_get_rf_opt_tl:neeN { endrangeprop }
3957     { \l__zrefclever_label_type_a_tl }
3958     { \l__zrefclever_ref_language_tl }
3959     \l__zrefclever_endrangeprop_tl
3960   \__zrefclever_get_rf_opt_bool:nneeN { cap } { false }
3961     { \l__zrefclever_label_type_a_tl }
3962     { \l__zrefclever_ref_language_tl }
3963     \l__zrefclever_cap_bool
3964   \__zrefclever_get_rf_opt_bool:nneeN { abbrev } { false }
3965     { \l__zrefclever_label_type_a_tl }
3966     { \l__zrefclever_ref_language_tl }
3967     \l__zrefclever_abbrev_bool
3968   \__zrefclever_get_rf_opt_bool:nneeN { rangetopair } { true }
3969     { \l__zrefclever_label_type_a_tl }
3970     { \l__zrefclever_ref_language_tl }
3971     \l__zrefclever_rangetopair_bool
3972   \__zrefclever_get_rf_opt_seq:neeN { refbounds-first }
3973     { \l__zrefclever_label_type_a_tl }
3974     { \l__zrefclever_ref_language_tl }
3975     \l__zrefclever_refbounds_first_seq
3976   \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-sg }

```

```

3977         { \l__zrefclever_label_type_a_tl }
3978         { \l__zrefclever_ref_language_tl }
3979         \l__zrefclever_refbounds_first_sg_seq
3980     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-pb }
3981         { \l__zrefclever_label_type_a_tl }
3982         { \l__zrefclever_ref_language_tl }
3983         \l__zrefclever_refbounds_first_pb_seq
3984     \__zrefclever_get_rf_opt_seq:neeN { refbounds-first-rb }
3985         { \l__zrefclever_label_type_a_tl }
3986         { \l__zrefclever_ref_language_tl }
3987         \l__zrefclever_refbounds_first_rb_seq
3988     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid }
3989         { \l__zrefclever_label_type_a_tl }
3990         { \l__zrefclever_ref_language_tl }
3991         \l__zrefclever_refbounds_mid_seq
3992     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-rb }
3993         { \l__zrefclever_label_type_a_tl }
3994         { \l__zrefclever_ref_language_tl }
3995         \l__zrefclever_refbounds_mid_rb_seq
3996     \__zrefclever_get_rf_opt_seq:neeN { refbounds-mid-re }
3997         { \l__zrefclever_label_type_a_tl }
3998         { \l__zrefclever_ref_language_tl }
3999         \l__zrefclever_refbounds_mid_re_seq
4000     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last }
4001         { \l__zrefclever_label_type_a_tl }
4002         { \l__zrefclever_ref_language_tl }
4003         \l__zrefclever_refbounds_last_seq
4004     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-pe }
4005         { \l__zrefclever_label_type_a_tl }
4006         { \l__zrefclever_ref_language_tl }
4007         \l__zrefclever_refbounds_last_pe_seq
4008     \__zrefclever_get_rf_opt_seq:neeN { refbounds-last-re }
4009         { \l__zrefclever_label_type_a_tl }
4010         { \l__zrefclever_ref_language_tl }
4011         \l__zrefclever_refbounds_last_re_seq
4012     }
4013
4014     % Here we send this to a couple of auxiliary functions.
4015     \bool_if:NTF \l__zrefclever_last_of_type_bool
4016     % There exists no next label of the same type as the current.
4017     { \__zrefclever_typeset_refs_last_of_type: }
4018     % There exists a next label of the same type as the current.
4019     { \__zrefclever_typeset_refs_not_last_of_type: }
4020 }
4021 }

```

(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.

```

4022 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
4023   {
4024     % Process the current label to the current queue.
4025     \int_case:nnF { \l__zrefclever_label_count_int }
4026     {
4027       % It is the last label of its type, but also the first one, and that's
4028       % what matters here: just store it.
4029       % Test: `zc-typeset01.lvt': "Last of type: single"
4030       { 0 }
4031       {
4032         \tl_set:NV \l__zrefclever_type_first_label_tl
4033           \l__zrefclever_label_a_tl
4034         \tl_set:NV \l__zrefclever_type_first_label_type_tl
4035           \l__zrefclever_label_type_a_tl
4036         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4037           \l__zrefclever_refbounds_first_sg_seq
4038         \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4039       }
4040
4041       % The last is the second: we have a pair (if not repeated).
4042       % Test: `zc-typeset01.lvt': "Last of type: pair"
4043       { 1 }
4044       {
4045         \int_compare:nNnTF { \l__zrefclever_range_same_count_int } = { 1 }
4046         {
4047           \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4048             \l__zrefclever_refbounds_first_sg_seq
4049           \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4050         }
4051         {
4052           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4053             {
4054               \exp_not:V \l__zrefclever_pairsep_tl
4055               \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4056               \l__zrefclever_refbounds_last_pe_seq
4057             }
4058           \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4059             \l__zrefclever_refbounds_first_pb_seq
4060           \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4061         }
4062       }
4063     }
4064     % Last is third or more of its type: without repetition, we'd have the
4065     % last element on a list, but control for possible repetition.
4066     {
4067       \int_case:nnF { \l__zrefclever_range_count_int }
4068       {
4069         % There was no range going on.
4070         % Test: `zc-typeset01.lvt': "Last of type: not range"
4071         { 0 }

```

```

4072 {
4073   \int_compare:nNnTF { \l__zrefclever_ref_count_int } < { 2 }
4074   {
4075     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4076     {
4077       \exp_not:V \l__zrefclever_pairsep_tl
4078       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4079       \l__zrefclever_refbounds_last_pe_seq
4080     }
4081   }
4082   {
4083     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4084     {
4085       \exp_not:V \l__zrefclever_lastsep_tl
4086       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4087       \l__zrefclever_refbounds_last_seq
4088     }
4089   }
4090 }
4091 % Last in the range is also the second in it.
4092 % Test: `zc-typeset01.lvt': "Last of type: pair in sequence"
4093 { 1 }
4094 {
4095   \int_compare:nNnTF
4096   { \l__zrefclever_range_same_count_int } = { 1 }
4097   {
4098     % We know `range_beg_is_first_bool' is false, since this is
4099     % the second element in the range, but the third or more in
4100     % the type list.
4101     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4102     {
4103       \exp_not:V \l__zrefclever_pairsep_tl
4104       \__zrefclever_get_ref:VN
4105       \l__zrefclever_range_beg_label_tl
4106       \l__zrefclever_refbounds_last_pe_seq
4107     }
4108     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4109     \l__zrefclever_refbounds_first_pb_seq
4110     \bool_set_true:N
4111     \l__zrefclever_type_first_refbounds_set_bool
4112   }
4113   {
4114     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4115     {
4116       \exp_not:V \l__zrefclever_listsep_tl
4117       \__zrefclever_get_ref:VN
4118       \l__zrefclever_range_beg_label_tl
4119       \l__zrefclever_refbounds_mid_seq
4120       \exp_not:V \l__zrefclever_lastsep_tl
4121       \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4122       \l__zrefclever_refbounds_last_seq
4123     }
4124   }
4125 }

```

```

4126 }
4127 % Last in the range is third or more in it.
4128 {
4129   \int_case:nnF
4130   {
4131     \l__zrefclever_range_count_int -
4132     \l__zrefclever_range_same_count_int
4133   }
4134   {
4135     % Repetition, not a range.
4136     % Test: `zc-typeset01.lvt': "Last of type: range to one"
4137     { 0 }
4138     {
4139       % If `range_beg_is_first_bool' is true, it means it was also
4140       % the first of the type, and hence its typesetting was
4141       % already handled, and we just have to set refbounds.
4142       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4143       {
4144         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4145         \l__zrefclever_refbounds_first_sg_seq
4146         \bool_set_true:N
4147         \l__zrefclever_type_first_refbounds_set_bool
4148       }
4149       {
4150         \int_compare:nNnTF
4151         { \l__zrefclever_ref_count_int } < { 2 }
4152         {
4153           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4154           {
4155             \exp_not:V \l__zrefclever_pairsep_tl
4156             \__zrefclever_get_ref:VN
4157             \l__zrefclever_range_beg_label_tl
4158             \l__zrefclever_refbounds_last_pe_seq
4159           }
4160         }
4161         {
4162           \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4163           {
4164             \exp_not:V \l__zrefclever_lastsep_tl
4165             \__zrefclever_get_ref:VN
4166             \l__zrefclever_range_beg_label_tl
4167             \l__zrefclever_refbounds_last_seq
4168           }
4169         }
4170       }
4171     }
4172     % A `range', but with no skipped value, treat as pair if range
4173     % started with first of type, otherwise as list.
4174     % Test: `zc-typeset01.lvt': "Last of type: range to pair"
4175     { 1 }
4176     {
4177       % Ditto.
4178       \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4179       {

```

```

4180         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4181         \l__zrefclever_refbounds_first_pb_seq
4182     \bool_set_true:N
4183     \l__zrefclever_type_first_refbounds_set_bool
4184     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4185     {
4186         \exp_not:V \l__zrefclever_pairsep_tl
4187         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4188         \l__zrefclever_refbounds_last_pe_seq
4189     }
4190 }
4191 {
4192     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4193     {
4194         \exp_not:V \l__zrefclever_listsep_tl
4195         \__zrefclever_get_ref:VN
4196         \l__zrefclever_range_beg_label_tl
4197         \l__zrefclever_refbounds_mid_seq
4198     }
4199     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4200     {
4201         \exp_not:V \l__zrefclever_lastsep_tl
4202         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4203         \l__zrefclever_refbounds_last_seq
4204     }
4205 }
4206 }
4207 }
4208 {
4209     % An actual range.
4210     % Test: `zc-typeset01.lvt': "Last of type: range"
4211     % Ditto.
4212     \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4213     {
4214         \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4215         \l__zrefclever_refbounds_first_rb_seq
4216         \bool_set_true:N
4217         \l__zrefclever_type_first_refbounds_set_bool
4218     }
4219     {
4220         \int_compare:nNnTF
4221         { \l__zrefclever_ref_count_int } < { 2 }
4222         {
4223             \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4224             {
4225                 \exp_not:V \l__zrefclever_pairsep_tl
4226                 \__zrefclever_get_ref:VN
4227                 \l__zrefclever_range_beg_label_tl
4228                 \l__zrefclever_refbounds_mid_rb_seq
4229             }
4230             \seq_set_eq:NN
4231             \l__zrefclever_type_first_refbounds_seq
4232             \l__zrefclever_refbounds_first_pb_seq
4233             \bool_set_true:N

```



```

4234         \l__zrefclever_type_first_refbounds_set_bool
4235     }
4236     {
4237     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4238     {
4239         \exp_not:V \l__zrefclever_lastsep_tl
4240         \__zrefclever_get_ref:VN
4241         \l__zrefclever_range_beg_label_tl
4242         \l__zrefclever_refbounds_mid_rb_seq
4243     }
4244     }
4245 }
4246 \bool_lazy_and:nnTF
4247 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4248 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4249 {
4250     \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4251     \l__zrefclever_range_beg_label_tl
4252     \l__zrefclever_label_a_tl
4253     \l__zrefclever_range_end_ref_tl
4254     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4255     {
4256         \exp_not:V \l__zrefclever_rangesep_tl
4257         \__zrefclever_get_ref_endrange:VVN
4258         \l__zrefclever_label_a_tl
4259         \l__zrefclever_range_end_ref_tl
4260         \l__zrefclever_refbounds_last_re_seq
4261     }
4262     }
4263     {
4264     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4265     {
4266         \exp_not:V \l__zrefclever_rangesep_tl
4267         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4268         \l__zrefclever_refbounds_last_re_seq
4269     }
4270     }
4271 }
4272 }
4273 }
4274
4275 % Handle "range" option. The idea is simple: if the queue is not empty,
4276 % we replace it with the end of the range (or pair). We can still
4277 % retrieve the end of the range from `label_a' since we know to be
4278 % processing the last label of its type at this point.
4279 \bool_if:NT \l__zrefclever_typeset_range_bool
4280 {
4281     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4282     {
4283         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4284         { }
4285         {
4286             \msg_warning:nne { zref-clever } { single-element-range }
4287             { \l__zrefclever_type_first_label_type_tl }

```

```

4288     }
4289   }
4290   {
4291     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4292     \bool_if:NT \l__zrefclever_rangetopair_bool
4293     {
4294       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4295       { }
4296       {
4297         \__zrefclever_labels_in_sequence:nn
4298         { \l__zrefclever_type_first_label_tl }
4299         { \l__zrefclever_label_a_tl }
4300       }
4301     }
4302     % Test: `zc-typeset01.lvt': "Last of type: option range"
4303     % Test: `zc-typeset01.lvt': "Last of type: option range to pair"
4304     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4305     {
4306       \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4307       {
4308         \exp_not:V \l__zrefclever_pairsep_tl
4309         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4310         \l__zrefclever_refbounds_last_pe_seq
4311       }
4312       \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4313       \l__zrefclever_refbounds_first_pb_seq
4314       \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4315     }
4316     {
4317       \bool_lazy_and:nnTF
4318       { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4319       { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VVN } }
4320       {
4321         % We must get `type_first_label_tl' instead of
4322         % `range_beg_label_tl' here, since it is not necessary
4323         % that the first of type was actually starting a range for
4324         % the `range' option to be used.
4325         \use:c { \l__zrefclever_endrangefunc_tl :VVN }
4326         \l__zrefclever_type_first_label_tl
4327         \l__zrefclever_label_a_tl
4328         \l__zrefclever_range_end_ref_tl
4329         \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4330         {
4331           \exp_not:V \l__zrefclever_rangesep_tl
4332           \__zrefclever_get_ref_endrange:VVN
4333           \l__zrefclever_label_a_tl
4334           \l__zrefclever_range_end_ref_tl
4335           \l__zrefclever_refbounds_last_re_seq
4336         }
4337       }
4338     }
4339     \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4340     {
4341       \exp_not:V \l__zrefclever_rangesep_tl

```

```

4342         \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4343         \l__zrefclever_refbounds_last_re_seq
4344     }
4345 }
4346 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4347 \l__zrefclever_refbounds_first_rb_seq
4348 \bool_set_true:N \l__zrefclever_type_first_refbounds_set_bool
4349 }
4350 }
4351 }
4352
4353 % If none of the special cases for the first of type refbounds have been
4354 % set, do it.
4355 \bool_if:NF \l__zrefclever_type_first_refbounds_set_bool
4356 {
4357     \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4358     \l__zrefclever_refbounds_first_seq
4359 }
4360
4361 % Now that the type block is finished, we can add the name and the first
4362 % ref to the queue. Also, if "typeset" option is not "both", handle it
4363 % here as well.
4364 \__zrefclever_type_name_setup:
4365 \bool_if:nTF
4366 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
4367 {
4368     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4369     { \__zrefclever_get_ref_first: }
4370 }
4371 {
4372     \bool_if:NTF \l__zrefclever_typeset_ref_bool
4373     {
4374         % Test: `zc-typeset01.lvt': "Last of type: option typeset ref"
4375         \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4376         {
4377             \__zrefclever_get_ref:VN \l__zrefclever_type_first_label_tl
4378             \l__zrefclever_type_first_refbounds_seq
4379         }
4380     }
4381     {
4382         \bool_if:NTF \l__zrefclever_typeset_name_bool
4383         {
4384             % Test: `zc-typeset01.lvt': "Last of type: option typeset name"
4385             \tl_set:Ne \l__zrefclever_typeset_queue_curr_tl
4386             {
4387                 \bool_if:NTF \l__zrefclever_name_in_link_bool
4388                 {
4389                     \exp_not:N \group_begin:
4390                     \exp_not:V \l__zrefclever_namefont_tl
4391                     \__zrefclever_hyperlink:nnn
4392                     {
4393                         \__zrefclever_extract_url_unexp:V
4394                         \l__zrefclever_type_first_label_tl
4395                     }
4396                 }
4397             }
4398         }
4399     }

```

```

4396         {
4397             \__zrefclever_extract_unexp:Vnn
4398             \l__zrefclever_type_first_label_tl
4399             { anchor } { }
4400         }
4401         { \exp_not:V \l__zrefclever_type_name_tl }
4402         \exp_not:N \group_end:
4403     }
4404     {
4405         \exp_not:N \group_begin:
4406         \exp_not:V \l__zrefclever_namefont_tl
4407         \exp_not:V \l__zrefclever_type_name_tl
4408         \exp_not:N \group_end:
4409     }
4410 }
4411 }
4412 {
4413     % Logically, this case would correspond to "typeset=none", but
4414     % it should not occur, given that the options are set up to
4415     % typeset either "ref" or "name". Still, leave here a
4416     % sensible fallback, equal to the behavior of "both".
4417     % Test: `zc-typeset01.lvt': "Last of type: option typeset none"
4418     \tl_put_left:Ne \l__zrefclever_typeset_queue_curr_tl
4419     { \__zrefclever_get_ref_first: }
4420 }
4421 }
4422 }
4423
4424 % Typeset the previous type block, if there is one.
4425 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
4426 {
4427     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
4428     { \l__zrefclever_tlistsep_tl }
4429     \l__zrefclever_typeset_queue_prev_tl
4430 }
4431
4432 % Extra log for testing.
4433 \bool_if:NT \l__zrefclever_verbose_testing_bool
4434 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
4435
4436 % Wrap up loop, or prepare for next iteration.
4437 \bool_if:NTF \l__zrefclever_typeset_last_bool
4438 {
4439     % We are finishing, typeset the current queue.
4440     \int_case:nnF { \l__zrefclever_type_count_int }
4441     {
4442         % Single type.
4443         % Test: `zc-typeset01.lvt': "Last of type: single type"
4444         { 0 }
4445         { \l__zrefclever_typeset_queue_curr_tl }
4446         % Pair of types.
4447         % Test: `zc-typeset01.lvt': "Last of type: pair of types"
4448         { 1 }
4449         {

```

```

4450         \l__zrefclever_tpairsep_tl
4451         \l__zrefclever_typeset_queue_curr_tl
4452     }
4453 }
4454 {
4455     % Last in list of types.
4456     % Test: `zc-typeset01.lvt': "Last of type: list of types"
4457     \l__zrefclever_tlastsep_tl
4458     \l__zrefclever_typeset_queue_curr_tl
4459 }
4460 % And nudge in case of multitype reference.
4461 \bool_lazy_all:nT
4462 {
4463     { \l__zrefclever_nudge_enabled_bool }
4464     { \l__zrefclever_nudge_multitype_bool }
4465     { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
4466 }
4467 { \msg_warning:nn { zref-clever } { nudge-multitype } }
4468 }
4469 {
4470     % There are further labels, set variables for next iteration.
4471     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
4472         \l__zrefclever_typeset_queue_curr_tl
4473     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
4474     \tl_clear:N \l__zrefclever_type_first_label_tl
4475     \tl_clear:N \l__zrefclever_type_first_label_type_tl
4476     \tl_clear:N \l__zrefclever_range_beg_label_tl
4477     \tl_clear:N \l__zrefclever_range_end_ref_tl
4478     \int_zero:N \l__zrefclever_label_count_int
4479     \int_zero:N \l__zrefclever_ref_count_int
4480     \int_incr:N \l__zrefclever_type_count_int
4481     \int_zero:N \l__zrefclever_range_count_int
4482     \int_zero:N \l__zrefclever_range_same_count_int
4483     \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4484     \bool_set_false:N \l__zrefclever_type_first_refbounds_set_bool
4485 }
4486 }

```

(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.

```

4487 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
4488 {
4489     % Signal if next label may form a range with the current one (only
4490     % considered if compression is enabled in the first place).
4491     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
4492     \bool_set_false:N \l__zrefclever_next_is_same_bool
4493     \bool_if:NT \l__zrefclever_typeset_compress_bool
4494     {
4495         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
4496         { }
4497         {
4498             \__zrefclever_labels_in_sequence:nn
4499             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }

```

```

4500     }
4501 }
4502
4503 % Process the current label to the current queue.
4504 \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
4505 {
4506     % Current label is the first of its type (also not the last, but it
4507     % doesn't matter here): just store the label.
4508     \tl_set:NV \l__zrefclever_type_first_label_tl
4509         \l__zrefclever_label_a_tl
4510     \tl_set:NV \l__zrefclever_type_first_label_type_tl
4511         \l__zrefclever_label_type_a_tl
4512     \int_incr:N \l__zrefclever_ref_count_int
4513
4514     % If the next label may be part of a range, signal it (we deal with it
4515     % as the "first", and must do it there, to handle hyperlinking), but
4516     % also step the range counters.
4517     % Test: `zc-typeset01.lvt': "Not last of type: first is range"
4518     \bool_if:NT \l__zrefclever_next_maybe_range_bool
4519     {
4520         \bool_set_true:N \l__zrefclever_range_beg_is_first_bool
4521         \tl_set:NV \l__zrefclever_range_beg_label_tl
4522             \l__zrefclever_label_a_tl
4523         \tl_clear:N \l__zrefclever_range_end_ref_tl
4524         \int_incr:N \l__zrefclever_range_count_int
4525         \bool_if:NT \l__zrefclever_next_is_same_bool
4526             { \int_incr:N \l__zrefclever_range_same_count_int }
4527     }
4528 }
4529 {
4530     % Current label is neither the first (nor the last) of its type.
4531     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
4532     {
4533         % Starting, or continuing a range.
4534         \int_compare:nNnTF
4535             { \l__zrefclever_range_count_int } = { 0 }
4536         {
4537             % There was no range going, we are starting one.
4538             \tl_set:NV \l__zrefclever_range_beg_label_tl
4539                 \l__zrefclever_label_a_tl
4540             \tl_clear:N \l__zrefclever_range_end_ref_tl
4541             \int_incr:N \l__zrefclever_range_count_int
4542             \bool_if:NT \l__zrefclever_next_is_same_bool
4543                 { \int_incr:N \l__zrefclever_range_same_count_int }
4544         }
4545         {
4546             % Second or more in the range, but not the last.
4547             \int_incr:N \l__zrefclever_range_count_int
4548             \bool_if:NT \l__zrefclever_next_is_same_bool
4549                 { \int_incr:N \l__zrefclever_range_same_count_int }
4550         }
4551     }
4552 }
4553     % Next element is not in sequence: there was no range, or we are

```

```

4554 % closing one.
4555 \int_case:nNF { \l__zrefclever_range_count_int }
4556 {
4557 % There was no range going on.
4558 % Test: `zc-typeset01.lvt': "Not last of type: no range"
4559 { 0 }
4560 {
4561 \int_incr:N \l__zrefclever_ref_count_int
4562 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4563 {
4564 \exp_not:V \l__zrefclever_listsep_tl
4565 \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4566 \l__zrefclever_refbounds_mid_seq
4567 }
4568 }
4569 % Last is second in the range: if `range_same_count' is also
4570 % `1', it's a repetition (drop it), otherwise, it's a "pair
4571 % within a list", treat as list.
4572 % Test: `zc-typeset01.lvt': "Not last of type: range pair to one"
4573 % Test: `zc-typeset01.lvt':"Not last of type: range pair"
4574 { 1 }
4575 {
4576 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4577 {
4578 \seq_set_eq:NN \l__zrefclever_type_first_refbounds_seq
4579 \l__zrefclever_refbounds_first_seq
4580 \bool_set_true:N
4581 \l__zrefclever_type_first_refbounds_set_bool
4582 }
4583 {
4584 \int_incr:N \l__zrefclever_ref_count_int
4585 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4586 {
4587 \exp_not:V \l__zrefclever_listsep_tl
4588 \__zrefclever_get_ref:VN
4589 \l__zrefclever_range_beg_label_tl
4590 \l__zrefclever_refbounds_mid_seq
4591 }
4592 }
4593 \int_compare:nNnF
4594 { \l__zrefclever_range_same_count_int } = { 1 }
4595 {
4596 \int_incr:N \l__zrefclever_ref_count_int
4597 \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4598 {
4599 \exp_not:V \l__zrefclever_listsep_tl
4600 \__zrefclever_get_ref:VN
4601 \l__zrefclever_label_a_tl
4602 \l__zrefclever_refbounds_mid_seq
4603 }
4604 }
4605 }
4606 }
4607 {

```

```

4608 % Last is third or more in the range: if `range_count' and
4609 % `range_same_count' are the same, its a repetition (drop it),
4610 % if they differ by `1', its a list, if they differ by more,
4611 % it is a real range.
4612 \int_case:nnF
4613 {
4614   \l__zrefclever_range_count_int -
4615   \l__zrefclever_range_same_count_int
4616 }
4617 {
4618 % Test: `zc-typeset01.lvt': "Not last of type: range to one"
4619 { 0 }
4620 {
4621   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4622   {
4623     \seq_set_eq:NN
4624     \l__zrefclever_type_first_refbounds_seq
4625     \l__zrefclever_refbounds_first_seq
4626     \bool_set_true:N
4627     \l__zrefclever_type_first_refbounds_set_bool
4628   }
4629   {
4630     \int_incr:N \l__zrefclever_ref_count_int
4631     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4632     {
4633       \exp_not:V \l__zrefclever_listsep_tl
4634       \__zrefclever_get_ref:VN
4635       \l__zrefclever_range_beg_label_tl
4636       \l__zrefclever_refbounds_mid_seq
4637     }
4638   }
4639 }
4640 % Test: `zc-typeset01.lvt': "Not last of type: range to pair"
4641 { 1 }
4642 {
4643   \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4644   {
4645     \seq_set_eq:NN
4646     \l__zrefclever_type_first_refbounds_seq
4647     \l__zrefclever_refbounds_first_seq
4648     \bool_set_true:N
4649     \l__zrefclever_type_first_refbounds_set_bool
4650   }
4651   {
4652     \int_incr:N \l__zrefclever_ref_count_int
4653     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4654     {
4655       \exp_not:V \l__zrefclever_listsep_tl
4656       \__zrefclever_get_ref:VN
4657       \l__zrefclever_range_beg_label_tl
4658       \l__zrefclever_refbounds_mid_seq
4659     }
4660   }
4661   \int_incr:N \l__zrefclever_ref_count_int

```



```

4662         \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4663         {
4664             \exp_not:V \l__zrefclever_listsep_tl
4665             \__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4666             \l__zrefclever_refbounds_mid_seq
4667         }
4668     }
4669 }
4670 {
4671 % Test: `zc-typeset01.lvt': "Not last of type: range"
4672 \bool_if:NTF \l__zrefclever_range_beg_is_first_bool
4673 {
4674     \seq_set_eq:NN
4675     \l__zrefclever_type_first_refbounds_seq
4676     \l__zrefclever_refbounds_first_rb_seq
4677     \bool_set_true:N
4678     \l__zrefclever_type_first_refbounds_set_bool
4679 }
4680 {
4681     \int_incr:N \l__zrefclever_ref_count_int
4682     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4683     {
4684         \exp_not:V \l__zrefclever_listsep_tl
4685         \__zrefclever_get_ref:VN
4686         \l__zrefclever_range_beg_label_tl
4687         \l__zrefclever_refbounds_mid_rb_seq
4688     }
4689 }
4690 % For the purposes of the serial comma, and thus for the
4691 % distinction of `lastsep' and `pairsep', a "range" counts
4692 % as one. Since `range_beg' has already been counted
4693 % (here or with the first of type), we refrain from
4694 % incrementing `ref_count_int'.
4695 \bool_lazy_and:nnTF
4696 { ! \tl_if_empty_p:N \l__zrefclever_endrangefunc_tl }
4697 { \cs_if_exist_p:c { \l__zrefclever_endrangefunc_tl :VWN } }
4698 {
4699     \use:c { \l__zrefclever_endrangefunc_tl :VWN }
4700     \l__zrefclever_range_beg_label_tl
4701     \l__zrefclever_label_a_tl
4702     \l__zrefclever_range_end_ref_tl
4703     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4704     {
4705         \exp_not:V \l__zrefclever_rangesep_tl
4706         \__zrefclever_get_ref_endrange:VWN
4707         \l__zrefclever_label_a_tl
4708         \l__zrefclever_range_end_ref_tl
4709         \l__zrefclever_refbounds_mid_re_seq
4710     }
4711 }
4712 {
4713     \tl_put_right:Ne \l__zrefclever_typeset_queue_curr_tl
4714     {
4715         \exp_not:V \l__zrefclever_rangesep_tl

```

```

4716             \l__zrefclever_get_ref:VN \l__zrefclever_label_a_tl
4717             \l__zrefclever_refbounds_mid_re_seq
4718         }
4719     }
4720 }
4721 }
4722 % We just closed a range, reset `range_beg_is_first' in case a
4723 % second range for the same type occurs, in which case its
4724 % `range_beg' will no longer be `first'.
4725 \bool_set_false:N \l__zrefclever_range_beg_is_first_bool
4726 % Reset counters.
4727 \int_zero:N \l__zrefclever_range_count_int
4728 \int_zero:N \l__zrefclever_range_same_count_int
4729 }
4730 }
4731 % Step label counter for next iteration.
4732 \int_incr:N \l__zrefclever_label_count_int
4733 }

```

(End of definition for `__zrefclever_typeset_refs_not_last_of_type:`)

Auxiliary functions

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

`__zrefclever_ref_default:` Default values for undefined references and undefined type names, respectively. We are 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.

```

4734 \cs_new_protected:Npn \__zrefclever_ref_default:
4735   { \zref@default }
4736 \cs_new_protected:Npn \__zrefclever_name_default:
4737   { \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>}
4738 \cs_new:Npn \__zrefclever_get_ref:nN #1#2
4739 {
4740   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
4741   {
4742     \bool_if:nTF
4743     {
4744       \l__zrefclever_hyperlink_bool &&
4745       ! \l__zrefclever_link_star_bool
4746     }
4747     {
4748       \seq_item:Nn #2 { 1 }
4749       \__zrefclever_hyperlink:nnn
4750       { \__zrefclever_extract_url_unexp:n {#1} }
4751       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4752       {
4753         \seq_item:Nn #2 { 2 }
4754         \exp_not:N \group_begin:
4755         \exp_not:V \l__zrefclever_reffont_tl
4756         \__zrefclever_extract_unexp:nvn {#1}
4757         { \l__zrefclever_ref_property_tl } { }
4758         \exp_not:N \group_end:
4759         \seq_item:Nn #2 { 3 }
4760       }
4761       \seq_item:Nn #2 { 4 }
4762     }
4763     {
4764       \seq_item:Nn #2 { 1 }
4765       \seq_item:Nn #2 { 2 }
4766       \exp_not:N \group_begin:
4767       \exp_not:V \l__zrefclever_reffont_tl
4768       \__zrefclever_extract_unexp:nvn {#1}
4769       { \l__zrefclever_ref_property_tl } { }
4770       \exp_not:N \group_end:
4771       \seq_item:Nn #2 { 3 }
4772       \seq_item:Nn #2 { 4 }
4773     }
4774   }
4775   { \__zrefclever_ref_default: }
4776 }
4777 \cs_generate_variant:Nn \__zrefclever_get_ref:nN { VN }

```

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

```

\__zrefclever_get_ref_endrange:nnN {<label>} {<reference>} {<refbounds>}
4778 \cs_new:Npn \__zrefclever_get_ref_endrange:nnN #1#2#3

```

```

4779 {
4780   \str_if_eq:nnTF {#2} { zc@missingproperty }
4781   { \__zrefclever_ref_default: }
4782   {
4783     \bool_if:nTF
4784     {
4785       \l__zrefclever_hyperlink_bool &&
4786       ! \l__zrefclever_link_star_bool
4787     }
4788     {
4789       \seq_item:Nn #3 { 1 }
4790       \__zrefclever_hyperlink:nnn
4791       { \__zrefclever_extract_url_unexp:n {#1} }
4792       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
4793       {
4794         \seq_item:Nn #3 { 2 }
4795         \exp_not:N \group_begin:
4796         \exp_not:V \l__zrefclever_reffont_tl
4797         \exp_not:n {#2}
4798         \exp_not:N \group_end:
4799         \seq_item:Nn #3 { 3 }
4800       }
4801       \seq_item:Nn #3 { 4 }
4802     }
4803     {
4804       \seq_item:Nn #3 { 1 }
4805       \seq_item:Nn #3 { 2 }
4806       \exp_not:N \group_begin:
4807       \exp_not:V \l__zrefclever_reffont_tl
4808       \exp_not:n {#2}
4809       \exp_not:N \group_end:
4810       \seq_item:Nn #3 { 3 }
4811       \seq_item:Nn #3 { 4 }
4812     }
4813   }
4814 }
4815 \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.

```

4816 \cs_new:Npn \__zrefclever_get_ref_first:
4817 {
4818   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4819   { \__zrefclever_ref_default: }
4820   {
4821     \bool_if:NTF \l__zrefclever_name_in_link_bool

```

```

4822 {
4823   \zref@ifrefcontainsprop
4824     { \l__zrefclever_type_first_label_tl }
4825     { \l__zrefclever_ref_property_tl }
4826     {
4827       \__zrefclever_hyperlink:nnn
4828       {
4829         \__zrefclever_extract_url_unexp:V
4830         \l__zrefclever_type_first_label_tl
4831       }
4832       {
4833         \__zrefclever_extract_unexp:Vnn
4834         \l__zrefclever_type_first_label_tl { anchor } { }
4835       }
4836       {
4837         \exp_not:N \group_begin:
4838         \exp_not:V \l__zrefclever_namefont_tl
4839         \exp_not:V \l__zrefclever_type_name_tl
4840         \exp_not:N \group_end:
4841         \exp_not:V \l__zrefclever_namesep_tl
4842         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4843         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4844         \exp_not:N \group_begin:
4845         \exp_not:V \l__zrefclever_reffont_tl
4846         \__zrefclever_extract_unexp:Vvn
4847         \l__zrefclever_type_first_label_tl
4848         { \l__zrefclever_ref_property_tl } { }
4849         \exp_not:N \group_end:
4850         \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4851       }
4852       \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4853     }
4854     {
4855       \exp_not:N \group_begin:
4856       \exp_not:V \l__zrefclever_namefont_tl
4857       \exp_not:V \l__zrefclever_type_name_tl
4858       \exp_not:N \group_end:
4859       \exp_not:V \l__zrefclever_namesep_tl
4860       \__zrefclever_ref_default:
4861     }
4862   }
4863 {
4864   \bool_if:nTF \l__zrefclever_type_name_missing_bool
4865     {
4866       \__zrefclever_name_default:
4867       \exp_not:V \l__zrefclever_namesep_tl
4868     }
4869     {
4870       \exp_not:N \group_begin:
4871       \exp_not:V \l__zrefclever_namefont_tl
4872       \exp_not:V \l__zrefclever_type_name_tl
4873       \exp_not:N \group_end:
4874       \tl_if_empty:NF \l__zrefclever_type_name_tl
4875       { \exp_not:V \l__zrefclever_namesep_tl }

```

```

4876     }
4877 \zref@ifrefcontainsprop
4878 { \l__zrefclever_type_first_label_tl }
4879 { \l__zrefclever_ref_property_tl }
4880 {
4881   \bool_if:nTF
4882   {
4883     \l__zrefclever_hyperlink_bool &&
4884     ! \l__zrefclever_link_star_bool
4885   }
4886   {
4887     \seq_item:Nn
4888     \l__zrefclever_type_first_refbounds_seq { 1 }
4889     \__zrefclever_hyperlink:nnn
4890     {
4891       \__zrefclever_extract_url_unexp:V
4892       \l__zrefclever_type_first_label_tl
4893     }
4894     {
4895       \__zrefclever_extract_unexp:Vnn
4896       \l__zrefclever_type_first_label_tl { anchor } { }
4897     }
4898     {
4899       \seq_item:Nn
4900       \l__zrefclever_type_first_refbounds_seq { 2 }
4901       \exp_not:N \group_begin:
4902       \exp_not:V \l__zrefclever_reffont_tl
4903       \__zrefclever_extract_unexp:Vvn
4904       \l__zrefclever_type_first_label_tl
4905       { \l__zrefclever_ref_property_tl } { }
4906       \exp_not:N \group_end:
4907       \seq_item:Nn
4908       \l__zrefclever_type_first_refbounds_seq { 3 }
4909     }
4910     \seq_item:Nn
4911     \l__zrefclever_type_first_refbounds_seq { 4 }
4912   }
4913   {
4914     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 1 }
4915     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 2 }
4916     \exp_not:N \group_begin:
4917     \exp_not:V \l__zrefclever_reffont_tl
4918     \__zrefclever_extract_unexp:Vvn
4919     \l__zrefclever_type_first_label_tl
4920     { \l__zrefclever_ref_property_tl } { }
4921     \exp_not:N \group_end:
4922     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 3 }
4923     \seq_item:Nn \l__zrefclever_type_first_refbounds_seq { 4 }
4924   }
4925 }
4926 { \__zrefclever_ref_default: }
4927 }
4928 }
4929 }

```

(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`, `\l__zrefclever_name_in_link_bool`, and `\l__zrefclever_type_name_missing_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`.

```
4930 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4931   {
4932     \bool_if:nTF
4933       { \l__zrefclever_typeset_ref_bool && ! \l__zrefclever_typeset_name_bool }
4934       {
4935         % `typeset=ref' / `noname' option
4936         % Probably redundant, since in this case the type name is not being
4937         % typeset. But, for completeness sake:
4938         \tl_clear:N \l__zrefclever_type_name_tl
4939         \bool_set_false:N \l__zrefclever_name_in_link_bool
4940         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4941       }
4942     {
4943       \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4944       {
4945         \tl_clear:N \l__zrefclever_type_name_tl
4946         \bool_set_true:N \l__zrefclever_type_name_missing_bool
4947       }
4948       {
4949         \tl_if_eq:NnTF
4950           \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4951           {
4952             \tl_clear:N \l__zrefclever_type_name_tl
4953             \bool_set_true:N \l__zrefclever_type_name_missing_bool
4954           }
4955           {
4956             % Determine whether we should use capitalization,
4957             % abbreviation, and plural.
4958             \bool_lazy_or:nnTF
4959               { \l__zrefclever_cap_bool }
4960               {
4961                 \l__zrefclever_capfirst_bool &&
4962                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4963               }
4964             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4965             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4966             % If the queue is empty, we have a singular, otherwise,
4967             % plural.
4968             \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
```

```

4969         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4970         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4971 \bool_lazy_and:nnTF
4972   { \l__zrefclever_abbrev_bool }
4973   {
4974     ! \int_compare_p:nNn
4975       { \l__zrefclever_type_count_int } = { 0 } ||
4976     ! \l__zrefclever_noabbrev_first_bool
4977   }
4978   {
4979     \tl_set:NV \l__zrefclever_name_format_fallback_tl
4980       \l__zrefclever_name_format_tl
4981     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
4982   }
4983   { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4984
4985 % Handle number and gender nudges.
4986 % Note that these nudges get disabled for `typeset=ref' /
4987 % `noname' option, but in this case they are not really
4988 % meaningful anyway.
4989 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4990 {
4991   \bool_if:NTF \l__zrefclever_nudge_singular_bool
4992   {
4993     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4994     {
4995       \msg_warning:nne { zref-clever }
4996       { nudge-plural-when-sg }
4997       { \l__zrefclever_type_first_label_type_tl }
4998     }
4999   }
5000   {
5001     \bool_lazy_all:nT
5002     {
5003       { \l__zrefclever_nudge_comptosing_bool }
5004       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
5005       {
5006         \int_compare_p:nNn
5007           { \l__zrefclever_label_count_int } > { 0 }
5008       }
5009     }
5010     {
5011       \msg_warning:nne { zref-clever }
5012       { nudge-comptosing }
5013       { \l__zrefclever_type_first_label_type_tl }
5014     }
5015   }
5016 \bool_lazy_and:nnT
5017   { \l__zrefclever_nudge_gender_bool }
5018   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
5019   {
5020     \__zrefclever_get_rf_opt_seq:neeN { gender }
5021     { \l__zrefclever_type_first_label_type_tl }
5022     { \l__zrefclever_ref_language_tl }

```



```

5023         \l__zrefclever_type_name_gender_seq
5024     \seq_if_in:NVF
5025         \l__zrefclever_type_name_gender_seq
5026         \l__zrefclever_ref_gender_tl
5027     {
5028         \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
5029         {
5030             \msg_warning:nneee { zref-clever }
5031             { nudge-gender-not-declared-for-type }
5032             { \l__zrefclever_ref_gender_tl }
5033             { \l__zrefclever_type_first_label_type_tl }
5034             { \l__zrefclever_ref_language_tl }
5035         }
5036         {
5037             \msg_warning:nneeee { zref-clever }
5038             { nudge-gender-mismatch }
5039             { \l__zrefclever_type_first_label_type_tl }
5040             { \l__zrefclever_ref_gender_tl }
5041             {
5042                 \seq_use:Nn
5043                 \l__zrefclever_type_name_gender_seq { ,~ }
5044             }
5045             { \l__zrefclever_ref_language_tl }
5046         }
5047     }
5048 }
5049 }
5050
5051 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5052 {
5053     \__zrefclever_opt_tl_get:cNF
5054     {
5055         \__zrefclever_opt_varname_type:een
5056         { \l__zrefclever_type_first_label_type_tl }
5057         { \l__zrefclever_name_format_tl }
5058         { tl }
5059     }
5060     \l__zrefclever_type_name_tl
5061     {
5062         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5063         {
5064             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5065             \tl_put_left:NV \l__zrefclever_name_format_tl
5066             \l__zrefclever_ref_decl_case_tl
5067         }
5068         \__zrefclever_opt_tl_get:cNF
5069         {
5070             \__zrefclever_opt_varname_lang_type:eeen
5071             { \l__zrefclever_ref_language_tl }
5072             { \l__zrefclever_type_first_label_type_tl }
5073             { \l__zrefclever_name_format_tl }
5074             { tl }
5075         }
5076     }
5077     \l__zrefclever_type_name_tl

```

```

5077         {
5078             \tl_clear:N \l__zrefclever_type_name_tl
5079             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5080             \msg_warning:nnee { zref-clever } { missing-name }
5081             { \l__zrefclever_name_format_tl }
5082             { \l__zrefclever_type_first_label_type_tl }
5083         }
5084     }
5085 }
5086 {
5087     \__zrefclever_opt_tl_get:cNF
5088     {
5089         \__zrefclever_opt_varname_type:een
5090         { \l__zrefclever_type_first_label_type_tl }
5091         { \l__zrefclever_name_format_tl }
5092         { tl }
5093     }
5094     \l__zrefclever_type_name_tl
5095     {
5096         \__zrefclever_opt_tl_get:cNF
5097         {
5098             \__zrefclever_opt_varname_type:een
5099             { \l__zrefclever_type_first_label_type_tl }
5100             { \l__zrefclever_name_format_fallback_tl }
5101             { tl }
5102         }
5103         \l__zrefclever_type_name_tl
5104         {
5105             \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5106             {
5107                 \tl_put_left:Nn
5108                 \l__zrefclever_name_format_tl { - }
5109                 \tl_put_left:NV \l__zrefclever_name_format_tl
5110                 \l__zrefclever_ref_decl_case_tl
5111                 \tl_put_left:Nn
5112                 \l__zrefclever_name_format_fallback_tl { - }
5113                 \tl_put_left:NV
5114                 \l__zrefclever_name_format_fallback_tl
5115                 \l__zrefclever_ref_decl_case_tl
5116             }
5117             \__zrefclever_opt_tl_get:cNF
5118             {
5119                 \__zrefclever_opt_varname_lang_type:een
5120                 { \l__zrefclever_ref_language_tl }
5121                 { \l__zrefclever_type_first_label_type_tl }
5122                 { \l__zrefclever_name_format_tl }
5123                 { tl }
5124             }
5125             \l__zrefclever_type_name_tl
5126             {
5127                 \__zrefclever_opt_tl_get:cNF
5128                 {
5129                     \__zrefclever_opt_varname_lang_type:een
5130                     { \l__zrefclever_ref_language_tl }

```

```

5131         { \l__zrefclever_type_first_label_type_tl }
5132         { \l__zrefclever_name_format_fallback_tl }
5133         { tl }
5134     }
5135     \l__zrefclever_type_name_tl
5136     {
5137         \tl_clear:N \l__zrefclever_type_name_tl
5138         \bool_set_true:N
5139             \l__zrefclever_type_name_missing_bool
5140         \msg_warning:nnee { zref-clever }
5141             { missing-name }
5142             { \l__zrefclever_name_format_tl }
5143             { \l__zrefclever_type_first_label_type_tl }
5144     }
5145 }
5146 }
5147 }
5148 }
5149 }
5150 }
5151
5152 % Signal whether the type name is to be included in the hyperlink or
5153 % not.
5154 \bool_lazy_any:nTF
5155 {
5156     { ! \l__zrefclever_hyperlink_bool }
5157     { \l__zrefclever_link_star_bool }
5158     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5159     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5160 }
5161 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5162 {
5163     \bool_lazy_any:nTF
5164     {
5165         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5166         {
5167             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5168             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5169         }
5170         {
5171             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5172             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5173             \l__zrefclever_typeset_last_bool &&
5174             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5175         }
5176     }
5177     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5178     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5179 }
5180 }
5181 }

```

(End of definition for __zrefclever_type_name_setup:.)

__zrefclever_hyperlink:nmn This avoids using the internal \hyper@@link, using only public hyperref commands

(see <https://github.com/latex3/hyperref/issues/229#issuecomment-1093870142>, thanks Ulrike Fischer).

```

    \__zrefclever_hyperlink:nnn {<url/file>} {<anchor>} {<text>}
5182 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5183 {
5184   \tl_if_empty:nTF {#1}
5185     { \hyperlink {#2} {#3} }
5186     { \hyper@linkfile {#3} {#1} {#2} }
5187 }

```

(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`.

```

5188 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5189 {
5190   \zref@ifpropundefined { urluse }
5191     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5192     {
5193       \zref@ifrefcontainsprop {#1} { urluse }
5194         { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5195         { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5196     }
5197 }
5198 \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 `<label b>` comes in immediate sequence from `<label a>`. And sets both `\l__zrefclever_next_maybe_range_bool` and `\l__zrefclever_next_is_same_bool` to true if the two labels are the “same” (that is, have the same counter value). These two boolean variables are the basis for all range and compression handling inside `__zrefclever_typeset_refs_not_last_of_type:`, so this function is expected to be called at its beginning, if compression is enabled.

```

    \__zrefclever_labels_in_sequence:nn {<label a>} {<label b>}
5199 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5200 {
5201   \exp_args:Nee \tl_if_eq:nnT
5202     { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5203     { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5204   {
5205     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5206     {
5207       \exp_args:Nee \tl_if_eq:nnT
5208         { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5209         { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5210       {
5211         \int_compare:nNnTF

```

```

5212     { \_zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5213     =
5214     { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5215     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5216     {
5217       \int_compare:nNnT
5218         { \_zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5219         =
5220         { \_zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5221         {
5222           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5223           \bool_set_true:N \l__zrefclever_next_is_same_bool
5224         }
5225     }
5226   }
5227 }
5228 {
5229   \exp_args:Nee \tl_if_eq:nnT
5230   { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5231   { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5232   {
5233     \exp_args:Nee \tl_if_eq:nnT
5234     { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5235     { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5236     {
5237       \int_compare:nNnTF
5238         { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5239         =
5240         { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5241         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5242         {
5243           \int_compare:nNnT
5244             { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5245             =
5246             { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5247             {

```

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.

```

5248     \exp_args:Nee \tl_if_eq:nnT
5249     {
5250       \_zrefclever_extract_unexp:nvn {#1}
5251       { l__zrefclever_ref_property_tl } { }
5252     }
5253     {
5254       \_zrefclever_extract_unexp:nvn {#2}
5255       { l__zrefclever_ref_property_tl } { }
5256     }
5257     {
5258       \bool_set_true:N
5259       \l__zrefclever_next_maybe_range_bool

```

```

5260                                     \bool_set_true:N
5261                                     \l__zrefclever_next_is_same_bool
5262                                     }
5263                                 }
5264                             }
5265                         }
5266                     }
5267                 }
5268             }
5269         }

```

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

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

```

\__zrefclever_get_rf_opt_tl:nnnN
\__zrefclever_get_rf_opt_tl:nnnN {<option>}
  {<ref type>} {<language>} {<tl variable>}
5270 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5271   {
5272     % First attempt: general options.
5273     \__zrefclever_opt_tl_get:cNF
5274     { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5275     #4
5276     {
5277       % If not found, try type specific options.
5278       \__zrefclever_opt_tl_get:cNF
5279       { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5280       #4
5281       {
5282         % If not found, try type- and language-specific.
5283         \__zrefclever_opt_tl_get:cNF
5284         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5285         #4
5286         {
5287           % If not found, try language-specific default.
5288           \__zrefclever_opt_tl_get:cNF
5289           { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5290           #4
5291           {
5292             % If not found, try fallback.
5293             \__zrefclever_opt_tl_get:cNF
5294             { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5295             #4
5296             { \tl_clear:N #4 }
5297           }
5298         }
5299       }
5300     }
5301   }
5302 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { neeN }

```

(End of definition for `__zrefclever_get_rf_opt_tl:nnnN`.)

```

5303 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN {<option>}
5304   {<ref type>} {<language>} {<seq variable>}
5305   {
5306     % First attempt: general options.
5307     \__zrefclever_opt_seq_get:cNF
5308     { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5309     #4
5310     {
5311       % If not found, try type specific options.
5312       \__zrefclever_opt_seq_get:cNF
5313       { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5314       #4
5315       {
5316         % If not found, try type- and language-specific.
5317         \__zrefclever_opt_seq_get:cNF
5318         { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5319         #4
5320         {
5321           % If not found, try language-specific default.
5322           \__zrefclever_opt_seq_get:cNF
5323           { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5324           #4
5325           {
5326             % If not found, try fallback.
5327             \__zrefclever_opt_seq_get:cNF
5328             { \__zrefclever_opt_varname_fallback:nn {#1} { seq } }
5329             #4
5330             { \seq_clear:N #4 }
5331           }
5332         }
5333       }
5334     }
5335 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_seq:nnnN { neeN }

```

(End of definition for __zrefclever_get_rf_opt_seq:nnnN.)

```

5336 \cs_new_protected:Npn \__zrefclever_get_rf_opt_bool:nnnnN {<option>} {<default>}
5337   {<ref type>} {<language>} {<bool variable>}
5338   {
5339     % First attempt: general options.
5340     \__zrefclever_opt_bool_get:cNF
5341     { \__zrefclever_opt_varname_general:nn {#1} { bool } }
5342     #5
5343     {
5344       % If not found, try type specific options.
5345       \__zrefclever_opt_bool_get:cNF
5346       { \__zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5347       #5
5348       {
5349         % If not found, try type- and language-specific.
5350         \__zrefclever_opt_bool_get:cNF

```

```

5350         { \__zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5351         #5
5352         {
5353             % If not found, try language-specific default.
5354             \__zrefclever_opt_bool_get:cNF
5355             { \__zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5356             #5
5357             {
5358                 % If not found, try fallback.
5359                 \__zrefclever_opt_bool_get:cNF
5360                 { \__zrefclever_opt_varname_fallback:nn {#1} { bool } }
5361                 #5
5362                 { \use:c { bool_set_ #2 :N } #5 }
5363             }
5364         }
5365     }
5366 }
5367 }
5368 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_bool:nnnnN { nneeN }

```

(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.

For the record, <https://tex.stackexchange.com/a/724742> is of interest.

```

5369 \__zrefclever_compat_module:nm { appendix }
5370 {
5371     \newcounter { zc@appendix }
5372     \cs_if_exist:cTF { chapter }
5373     {

```



```

5374     \_zrefclever_zcsetup:e
5375     {
5376         counterresetby =
5377         {

```

In case someone did something like `\counterwithin{chapter}{part}`. Harmless otherwise.

```

5378             zc@appendix = \_zrefclever_counter_reset_by:n { chapter } ,
5379             chapter = zc@appendix ,
5380         } ,
5381     }
5382 }
5383 {
5384     \cs_if_exist:cT { section }
5385     {
5386         \_zrefclever_zcsetup:e
5387         {
5388             counterresetby =
5389             {
5390                 zc@appendix = \_zrefclever_counter_reset_by:n { section } ,
5391                 section = zc@appendix ,
5392             } ,
5393         }
5394     }
5395 }
5396 \AddToHook { cmd / appendix / before }
5397 {
5398     \setcounter { zc@appendix } { 1 }
5399     \_zrefclever_zcsetup:n
5400     {
5401         countertype =
5402         {
5403             chapter      = appendix ,
5404             section      = appendix ,
5405             subsection   = appendix ,
5406             subsubsection = appendix ,
5407             paragraph    = appendix ,
5408             subparagraph = appendix ,
5409         }
5410     }
5411 }
5412 }

```

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

9.2 appendices

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

```
5413 \__zrefclever_compat_module:nn { appendices }
5414 {
5415   \__zrefclever_if_package_loaded:nT { appendix }
5416   {
5417     \AddToHook { env / appendices / begin }
5418     {
```

Technically, the `appendices` environment can be called multiple times. By default, successive calls keep track of numbering and start where the previous one left off. Which means just setting the `zc@appendix` counter to 1 is enough for things to work, since the distinction between the calls and the sorting of their respective references will depend on the underlying sectioning. `appendix`’s documentation however, provides a way to restart from A at each call (by redefining `\restoreapp` to do nothing). In this case, the references inside different calls to `appendices` get to be identical in every way, including printed form, counter value, enclosing counters, etc., despite being different. We could keep track of different calls to `appendices` by having the `zc@appendix` counter be “stepped” at each call. Doing so would mean though that `\zcref` would distinguish things which are typeset identically, granting some arguably weird results. True, the user *can* change the printed form for each `appendices` call, e.g. redefining `\thechapter`, but in this case, they are responsible for keeping track of this.

```
5419       \setcounter { zc@appendix } { 1 }
5420       \__zrefclever_zcsetup:n
5421       {
5422         countertype =
5423         {
5424           chapter      = appendix ,
5425           section      = appendix ,
5426           subsection   = appendix ,
5427           subsubsection = appendix ,
5428           paragraph    = appendix ,
5429           subparagraph = appendix ,
5430         }
5431       }
5432     }
5433     \AddToHook { env / appendices / end }
5434     { \setcounter { zc@appendix } { 0 } }
5435     \newcounter { zc@subappendix }
5436     \cs_if_exist:cTF { chapter }
5437     {
5438       \__zrefclever_zcsetup:e
5439       {
5440         counterresetby =
5441         {
5442           zc@subappendix = \__zrefclever_counter_reset_by:n { section } ,
5443           section = zc@subappendix ,
5444         } ,
5445       }
5446     }
5447     {
```

```

5448     \__zrefclever_zcsetup:e
5449     {
5450         counterresetby =
5451         {
5452             zc@subappendix = \__zrefclever_counter_reset_by:n { subsection } ,
5453             subsection = zc@subappendix ,
5454         } ,
5455     }
5456 }
5457 \AddToHook { env / subappendices / begin }
5458 {

```

The `subappendices` environment, on the other hand, appears not to support multiple calls inside the same chapter/section (the counter is reset by default). Either way, the same reasoning applies.

```

5459     \setcounter { zc@subappendix } { 1 }
5460     \__zrefclever_zcsetup:n
5461     {
5462         countertype =
5463         {
5464             section      = appendix ,
5465             subsection   = appendix ,
5466             subsubsection = appendix ,
5467             paragraph    = appendix ,
5468             subparagraph = appendix ,
5469         } ,
5470     }
5471 }
5472 \AddToHook { env / subappendices / end }
5473 { \setcounter { zc@subappendix } { 0 } }
5474 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5475 }
5476 }

```

9.3 memoir

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

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

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

```

5477 \__zrefclever_compat_module:nn { memoir }
5478 {

```

```

5479 \__zrefclever_if_class_loaded:nT { memoir }
5480 {

```

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.

```

5481 \__zrefclever_zcsetup:n
5482 {
5483   countertype =
5484   {
5485     subfigure = figure ,
5486     subtable = table ,
5487     poemline = line ,
5488   } ,
5489   counterresetby =
5490   {
5491     subfigure = figure ,
5492     subtable = table ,
5493   } ,
5494 }

```

Support for subcaption references.

```

5495 \zref@newprop { subcaption }
5496 { \cs_if_exist_use:c { @@thesub \@capttype } }
5497 \AddToHook{ memoir/subcaption/aftercounter }
5498 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

Support for `\sidefootnote` and `\pagenote`.

```

5499 \__zrefclever_zcsetup:n
5500 {
5501   countertype =
5502   {
5503     sidefootnote = footnote ,
5504     pagenote = endnote ,
5505   } ,
5506 }
5507 \msg_info:nnn { zref-clever } { compat-class } { memoir }
5508 }
5509 }

```

9.4 amsmath

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

```

5510 \__zrefclever_compat_module:nn { amsmath }
5511 {
5512   \__zrefclever_if_package_loaded:nT { amsmath }
5513   {

```

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).

```

5514     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5515     \AddToHook { env / subequations / begin }
5516     {
5517         \__zrefclever_zcsetup:e
5518         {
5519             counterresetby =
5520             {
5521                 parentequation =
5522                 \__zrefclever_counter_reset_by:n { equation } ,
5523                 equation = parentequation ,
5524             } ,
5525             currentcounter = parentequation ,
5526             countertype = { parentequation = equation } ,
5527         }
5528     \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5529 }

```

`amsmath` does use `\refstepcounter` for the `equation` counter throughout and supposedly sets `\@currentcounter` for `\tags` (I'm not sure if it works in all environments, though. Once I tried to remove the explicit `currentcounter` setting and several labels to `\tags` ended up with type `section`. But I didn't investigate this further). 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.

```

5530     \zref@newprop { subeq } { \alph { equation } }
5531     \clist_map_inline:nn
5532     {
5533         equation ,
5534         equation* ,
5535         align ,
5536         align* ,
5537         alignat ,
5538         alignat* ,
5539         flalign ,
5540         flalign* ,
5541         xalignat ,
5542         xalignat* ,
5543         gather ,
5544         gather* ,

```

```

5545     multiline ,
5546     multiline* ,
5547   }
5548   {
5549     \AddToHook { env / #1 / begin }
5550     {
5551       \__zrefclever_zcsetup:n { currentcounter = equation }
5552       \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5553         { \zref@localaddprop \ZREF@mainlist { subeq } }
5554     }
5555   }
5556   \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5557 }
5558 }

```

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.

Note that this support comes at a little cost. `showonlyrefs` works by setting a special `\MT@newlabel` for each label referenced with `\eqref`. Now, `\eqref` is a specialized reference command, only used to refer to equations, so it sets `\MT@newlabel` unconditionally on the first run. `\zcref`, on the other hand, is a general purpose reference command, used to reference labels of any type. But we wouldn't want to set `\MT@newlabel` indiscriminately for all referenced labels in the document, so we need to test for its type. Alas, the label must exist before its type can be tested, thus we cannot set `\MT@newlabel` on the first run, only on the second. In sum, since `\eqref` requires 3 runs to work, `\zcref` needs 4.

```

5559 \bool_new:N \l__zrefclever_mathtools_loaded_bool
5560 \__zrefclever_compat_module:nn { mathtools }
5561   {
5562     \__zrefclever_if_package_loaded:nT { mathtools }
5563     {
5564       \bool_set_true:N \l__zrefclever_mathtools_loaded_bool
5565       \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5566         {
5567           \seq_map_inline:Nn #1
5568             {
5569               \tl_set:Ne \l__zrefclever_tmpa_tl
5570                 { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5571               \bool_lazy_or:nnT
5572                 { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { equation } }
5573                 { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { parentequation } }
5574                 { \noeqref {##1} }
5575             }

```

```

5576     }
5577     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5578   }
5579 }

```

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.

```

5580 \__zrefclever_compat_module:nn { breqn }
5581   {
5582     \__zrefclever_if_package_loaded:nT { breqn }
5583   }

```

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

```

5584     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5585     \AddToHook { env / dgroup / begin }
5586     {
5587       \__zrefclever_zcsetup:e
5588       {
5589         counterresetby =
5590         {
5591           parentequation =
5592           \__zrefclever_counter_reset_by:n { equation } ,
5593           equation = parentequation ,
5594         } ,
5595         currentcounter = parentequation ,
5596         countertype = { parentequation = equation } ,
5597       }
5598     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5599   }
5600 \zref@ifpropundefined { subeq }
5601   { \zref@newprop { subeq } { \alph { equation } } }
5602   { }
5603 \clist_map_inline:nn
5604   {
5605     dmath ,
5606     dseries ,
5607     darray ,
5608   }
5609   {
5610     \AddToHook { env / #1 / begin }
5611     {
5612       \__zrefclever_zcsetup:n { currentcounter = equation }
5613       \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5614         { \zref@localaddprop \ZREF@mainlist { subeq } }
5615     }

```

```

5616     }
5617     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5618   }
5619 }

```

9.7 listings

```

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

```

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.

```

5633   \lst@AddToHook { Init }
5634   { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5635   \msg_info:nnn { zref-clever } { compat-package } { listings }
5636 }
5637 }

```

9.8 enumitem

The procedure below will “see” any changes made to the `enumerate` environment (made with `enumitem`’s `\renewlist`) as long as it is done in the preamble. Though, technically, `\renewlist` can be issued anywhere in the document, this should be more than enough for the purpose at hand. Besides, trying to retrieve this information “on the fly” would be much overkill.

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

```

5638 \__zrefclever_compat_module:nn { enumitem }
5639 {
5640   \__zrefclever_if_package_loaded:nT { enumitem }

```



```

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

```

9.9 subcaption

```

5665 \__zrefclever_compat_module:nn { subcaption }
5666 {
5667   \__zrefclever_if_package_loaded:nT { subcaption }
5668   {
5669     \__zrefclever_zcsetup:n
5670     {
5671       countertype =
5672       {
5673         subfigure = figure ,
5674         subtable = table ,
5675       } ,
5676       counterresetby =
5677       {
5678         subfigure = figure ,
5679         subtable = table ,
5680       } ,
5681     }

```

Support for subref reference.

```

5682   \zref@newprop { subref }
5683   { \cs_if_exist_use:c { thesub \@capytype } }
5684   \tl_put_right:Nn \caption@subtypehook
5685   { \zref@localaddprop \ZREF@mainlist { subref } }
5686 }
5687 }

```

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.

```
5688 \__zrefclever_compat_module:nn { subfig }
5689   {
5690     \__zrefclever_if_package_loaded:nT { subfig }
5691     {
5692       \__zrefclever_zcsetup:n
5693       {
5694         countertype =
5695         {
5696           subfigure = figure ,
5697           subtable = table ,
5698         } ,
5699         counterresetby =
5700         {
5701           subfigure = figure ,
5702           subtable = table ,
5703         } ,
5704       }
5705     }
5706   }
5707 </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 `refbounds`, and also some abbreviated name forms. In the case of abbreviated name forms, it is usual and desirable to provide some, but they should be used sparingly, only for cases where the abbreviation is a common and well established tradition for the language. The reason is that `abbrev=true` is quite a common use case, and it is easier to provide an occasional wanted abbreviated form, if the language file didn’t include it, than it is to disable several unwanted ones, if the language file includes too many of them. What should be aimed at is to provide a good default abbreviations set. Unusual or disputable abbreviations should be avoided. In particular, there is no need at all to provide the same set of abbreviations for each language. It is not because English has them for a given type that some other language has to have them, and it is not because English lacks them for another type, that other languages shouldn’t have them. Still, with regard to abbreviated forms, it is better to be conservative than opinionated.

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

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

Precedence rule for options in the language files: Any option given twice or more times has to have some precedence rule. Normally, the language files should not contain options in duplicity, but they may happen when setting some “group” `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.

```

5708 \*package)
5709 \zcDeclareLanguage { english }
5710 \zcDeclareLanguageAlias { american } { english }
5711 \zcDeclareLanguageAlias { australian } { english }
5712 \zcDeclareLanguageAlias { british } { english }
5713 \zcDeclareLanguageAlias { canadian } { english }
5714 \zcDeclareLanguageAlias { newzealand } { english }
5715 \zcDeclareLanguageAlias { UKenglish } { english }
5716 \zcDeclareLanguageAlias { USenglish } { english }
5717 \*package)

5718 \*lang-english)

5719 namesep = {\nobreakspace} ,
5720 pairsep = {\and\nobreakspace} ,
5721 listsep = {,~} ,
5722 lastsep = {\and\nobreakspace} ,
5723 tpairsep = {\and\nobreakspace} ,
5724 tlistsep = {,~} ,
5725 tlastsep = {,~\and\nobreakspace} ,
5726 notesep = {~} ,
5727 rangesep = {\to\nobreakspace} ,
5728
5729 type = book ,
5730 Name-sg = Book ,
5731 name-sg = book ,
5732 Name-pl = Books ,
5733 name-pl = books ,
5734
5735 type = part ,
5736 Name-sg = Part ,
5737 name-sg = part ,
5738 Name-pl = Parts ,
5739 name-pl = parts ,
5740
5741 type = chapter ,
5742 Name-sg = Chapter ,
5743 name-sg = chapter ,
5744 Name-pl = Chapters ,
5745 name-pl = chapters ,
5746
5747 type = section ,

```

```

5748 Name-sg = Section ,
5749 name-sg = section ,
5750 Name-pl = Sections ,
5751 name-pl = sections ,
5752
5753 type = paragraph ,
5754 Name-sg = Paragraph ,
5755 name-sg = paragraph ,
5756 Name-pl = Paragraphs ,
5757 name-pl = paragraphs ,
5758 Name-sg-ab = Par. ,
5759 name-sg-ab = par. ,
5760 Name-pl-ab = Par. ,
5761 name-pl-ab = par. ,
5762
5763 type = appendix ,
5764 Name-sg = Appendix ,
5765 name-sg = appendix ,
5766 Name-pl = Appendices ,
5767 name-pl = appendices ,
5768
5769 type = page ,
5770 Name-sg = Page ,
5771 name-sg = page ,
5772 Name-pl = Pages ,
5773 name-pl = pages ,
5774 rangeseq = {\textendash} ,
5775 rangetopair = false ,
5776
5777 type = line ,
5778 Name-sg = Line ,
5779 name-sg = line ,
5780 Name-pl = Lines ,
5781 name-pl = lines ,
5782
5783 type = figure ,
5784 Name-sg = Figure ,
5785 name-sg = figure ,
5786 Name-pl = Figures ,
5787 name-pl = figures ,
5788 Name-sg-ab = Fig. ,
5789 name-sg-ab = fig. ,
5790 Name-pl-ab = Figs. ,
5791 name-pl-ab = figs. ,
5792
5793 type = table ,
5794 Name-sg = Table ,
5795 name-sg = table ,
5796 Name-pl = Tables ,
5797 name-pl = tables ,
5798
5799 type = item ,
5800 Name-sg = Item ,
5801 name-sg = item ,

```

```

5802 Name-pl = Items ,
5803 name-pl = items ,
5804
5805 type = footnote ,
5806 Name-sg = Footnote ,
5807 name-sg = footnote ,
5808 Name-pl = Footnotes ,
5809 name-pl = footnotes ,
5810
5811 type = endnote ,
5812 Name-sg = Note ,
5813 name-sg = note ,
5814 Name-pl = Notes ,
5815 name-pl = notes ,
5816
5817 type = note ,
5818 Name-sg = Note ,
5819 name-sg = note ,
5820 Name-pl = Notes ,
5821 name-pl = notes ,
5822
5823 type = equation ,
5824 Name-sg = Equation ,
5825 name-sg = equation ,
5826 Name-pl = Equations ,
5827 name-pl = equations ,
5828 Name-sg-ab = Eq. ,
5829 name-sg-ab = eq. ,
5830 Name-pl-ab = Eqs. ,
5831 name-pl-ab = eqs. ,
5832 refbounds-first-sg = {,(,)}, ,
5833 refbounds = {(,,)} ,
5834
5835 type = theorem ,
5836 Name-sg = Theorem ,
5837 name-sg = theorem ,
5838 Name-pl = Theorems ,
5839 name-pl = theorems ,
5840
5841 type = lemma ,
5842 Name-sg = Lemma ,
5843 name-sg = lemma ,
5844 Name-pl = Lemmas ,
5845 name-pl = lemmas ,
5846
5847 type = corollary ,
5848 Name-sg = Corollary ,
5849 name-sg = corollary ,
5850 Name-pl = Corollaries ,
5851 name-pl = corollaries ,
5852
5853 type = proposition ,
5854 Name-sg = Proposition ,
5855 name-sg = proposition ,

```

```

5856 Name-pl = Propositions ,
5857 name-pl = propositions ,
5858
5859 type = definition ,
5860 Name-sg = Definition ,
5861 name-sg = definition ,
5862 Name-pl = Definitions ,
5863 name-pl = definitions ,
5864
5865 type = proof ,
5866 Name-sg = Proof ,
5867 name-sg = proof ,
5868 Name-pl = Proofs ,
5869 name-pl = proofs ,
5870
5871 type = result ,
5872 Name-sg = Result ,
5873 name-sg = result ,
5874 Name-pl = Results ,
5875 name-pl = results ,
5876
5877 type = remark ,
5878 Name-sg = Remark ,
5879 name-sg = remark ,
5880 Name-pl = Remarks ,
5881 name-pl = remarks ,
5882
5883 type = example ,
5884 Name-sg = Example ,
5885 name-sg = example ,
5886 Name-pl = Examples ,
5887 name-pl = examples ,
5888
5889 type = algorithm ,
5890 Name-sg = Algorithm ,
5891 name-sg = algorithm ,
5892 Name-pl = Algorithms ,
5893 name-pl = algorithms ,
5894
5895 type = listing ,
5896 Name-sg = Listing ,
5897 name-sg = listing ,
5898 Name-pl = Listings ,
5899 name-pl = listings ,
5900
5901 type = exercise ,
5902 Name-sg = Exercise ,
5903 name-sg = exercise ,
5904 Name-pl = Exercises ,
5905 name-pl = exercises ,
5906
5907 type = solution ,
5908 Name-sg = Solution ,
5909 name-sg = solution ,

```



```

5910   Name-pl = Solutions ,
5911   name-pl = solutions ,
5912 </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`.

```

5913 (*package)
5914 \zcDeclareLanguage
5915 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5916 { german }
5917 \zcDeclareLanguageAlias { ngerman      } { german }
5918 \zcDeclareLanguageAlias { austrian    } { german }
5919 \zcDeclareLanguageAlias { naustrian   } { german }
5920 \zcDeclareLanguageAlias { swissgerman } { german }
5921 \zcDeclareLanguageAlias { nswissgerman } { german }
5922 </package>
5923 (*lang-german)
5924 namesep = {\nobreakspace} ,
5925 pairsep  = {\~und\nobreakspace} ,
5926 listsep  = {,~} ,
5927 lastsep  = {\~und\nobreakspace} ,
5928 tpairsep = {\~und\nobreakspace} ,
5929 tlistsep = {,~} ,
5930 tlastsep = {\~und\nobreakspace} ,
5931 notesep  = {\~} ,
5932 rangeseq = {\~bis\nobreakspace} ,
5933
5934 type = book ,
5935 gender = n ,
5936 case = N ,
5937   Name-sg = Buch ,
5938   Name-pl = Bücher ,
5939 case = A ,
5940   Name-sg = Buch ,
5941   Name-pl = Bücher ,
5942 case = D ,
5943   Name-sg = Buch ,
5944   Name-pl = Büchern ,
5945 case = G ,
5946   Name-sg = Buches ,
5947   Name-pl = Bücher ,
5948
5949 type = part ,
5950 gender = m ,
5951 case = N ,
5952   Name-sg = Teil ,
5953   Name-pl = Teile ,
5954 case = A ,
5955   Name-sg = Teil ,

```

```

5956     Name-pl = Teile ,
5957     case = D ,
5958     Name-sg = Teil ,
5959     Name-pl = Teilen ,
5960     case = G ,
5961     Name-sg = Teiles ,
5962     Name-pl = Teile ,
5963
5964 type = chapter ,
5965     gender = n ,
5966     case = N ,
5967     Name-sg = Kapitel ,
5968     Name-pl = Kapitel ,
5969     case = A ,
5970     Name-sg = Kapitel ,
5971     Name-pl = Kapitel ,
5972     case = D ,
5973     Name-sg = Kapitel ,
5974     Name-pl = Kapiteln ,
5975     case = G ,
5976     Name-sg = Kapitels ,
5977     Name-pl = Kapitel ,
5978
5979 type = section ,
5980     gender = m ,
5981     case = N ,
5982     Name-sg = Abschnitt ,
5983     Name-pl = Abschnitte ,
5984     case = A ,
5985     Name-sg = Abschnitt ,
5986     Name-pl = Abschnitte ,
5987     case = D ,
5988     Name-sg = Abschnitt ,
5989     Name-pl = Abschnitten ,
5990     case = G ,
5991     Name-sg = Abschnitts ,
5992     Name-pl = Abschnitte ,
5993
5994 type = paragraph ,
5995     gender = m ,
5996     case = N ,
5997     Name-sg = Absatz ,
5998     Name-pl = Absätze ,
5999     case = A ,
6000     Name-sg = Absatz ,
6001     Name-pl = Absätze ,
6002     case = D ,
6003     Name-sg = Absatz ,
6004     Name-pl = Absätzen ,
6005     case = G ,
6006     Name-sg = Absatzes ,
6007     Name-pl = Absätze ,
6008
6009 type = appendix ,

```

```

6010 gender = m ,
6011 case = N ,
6012     Name-sg = Anhang ,
6013     Name-pl = Anhänge ,
6014 case = A ,
6015     Name-sg = Anhang ,
6016     Name-pl = Anhänge ,
6017 case = D ,
6018     Name-sg = Anhang ,
6019     Name-pl = Anhängen ,
6020 case = G ,
6021     Name-sg = Anhangs ,
6022     Name-pl = Anhänge ,
6023
6024 type = page ,
6025 gender = f ,
6026 case = N ,
6027     Name-sg = Seite ,
6028     Name-pl = Seiten ,
6029 case = A ,
6030     Name-sg = Seite ,
6031     Name-pl = Seiten ,
6032 case = D ,
6033     Name-sg = Seite ,
6034     Name-pl = Seiten ,
6035 case = G ,
6036     Name-sg = Seite ,
6037     Name-pl = Seiten ,
6038 rangeseq = {\textendash} ,
6039 rangetopair = false ,
6040
6041 type = line ,
6042 gender = f ,
6043 case = N ,
6044     Name-sg = Zeile ,
6045     Name-pl = Zeilen ,
6046 case = A ,
6047     Name-sg = Zeile ,
6048     Name-pl = Zeilen ,
6049 case = D ,
6050     Name-sg = Zeile ,
6051     Name-pl = Zeilen ,
6052 case = G ,
6053     Name-sg = Zeile ,
6054     Name-pl = Zeilen ,
6055
6056 type = figure ,
6057 gender = f ,
6058 case = N ,
6059     Name-sg = Abbildung ,
6060     Name-pl = Abbildungen ,
6061     Name-sg-ab = Abb. ,
6062     Name-pl-ab = Abb. ,
6063 case = A ,

```

```

6064     Name-sg = Abbildung ,
6065     Name-pl = Abbildungen ,
6066     Name-sg-ab = Abb. ,
6067     Name-pl-ab = Abb. ,
6068 case = D ,
6069     Name-sg = Abbildung ,
6070     Name-pl = Abbildungen ,
6071     Name-sg-ab = Abb. ,
6072     Name-pl-ab = Abb. ,
6073 case = G ,
6074     Name-sg = Abbildung ,
6075     Name-pl = Abbildungen ,
6076     Name-sg-ab = Abb. ,
6077     Name-pl-ab = Abb. ,
6078
6079 type = table ,
6080     gender = f ,
6081     case = N ,
6082     Name-sg = Tabelle ,
6083     Name-pl = Tabellen ,
6084 case = A ,
6085     Name-sg = Tabelle ,
6086     Name-pl = Tabellen ,
6087 case = D ,
6088     Name-sg = Tabelle ,
6089     Name-pl = Tabellen ,
6090 case = G ,
6091     Name-sg = Tabelle ,
6092     Name-pl = Tabellen ,
6093
6094 type = item ,
6095     gender = m ,
6096     case = N ,
6097     Name-sg = Punkt ,
6098     Name-pl = Punkte ,
6099 case = A ,
6100     Name-sg = Punkt ,
6101     Name-pl = Punkte ,
6102 case = D ,
6103     Name-sg = Punkt ,
6104     Name-pl = Punkten ,
6105 case = G ,
6106     Name-sg = Punktes ,
6107     Name-pl = Punkte ,
6108
6109 type = footnote ,
6110     gender = f ,
6111     case = N ,
6112     Name-sg = Fußnote ,
6113     Name-pl = Fußnoten ,
6114 case = A ,
6115     Name-sg = Fußnote ,
6116     Name-pl = Fußnoten ,
6117 case = D ,

```

```

6118     Name-sg = Fußnote ,
6119     Name-pl = Fußnoten ,
6120     case = G ,
6121     Name-sg = Fußnote ,
6122     Name-pl = Fußnoten ,
6123
6124 type = endnote ,
6125     gender = f ,
6126     case = N ,
6127     Name-sg = Endnote ,
6128     Name-pl = Endnoten ,
6129     case = A ,
6130     Name-sg = Endnote ,
6131     Name-pl = Endnoten ,
6132     case = D ,
6133     Name-sg = Endnote ,
6134     Name-pl = Endnoten ,
6135     case = G ,
6136     Name-sg = Endnote ,
6137     Name-pl = Endnoten ,
6138
6139 type = note ,
6140     gender = f ,
6141     case = N ,
6142     Name-sg = Anmerkung ,
6143     Name-pl = Anmerkungen ,
6144     case = A ,
6145     Name-sg = Anmerkung ,
6146     Name-pl = Anmerkungen ,
6147     case = D ,
6148     Name-sg = Anmerkung ,
6149     Name-pl = Anmerkungen ,
6150     case = G ,
6151     Name-sg = Anmerkung ,
6152     Name-pl = Anmerkungen ,
6153
6154 type = equation ,
6155     gender = f ,
6156     case = N ,
6157     Name-sg = Gleichung ,
6158     Name-pl = Gleichungen ,
6159     case = A ,
6160     Name-sg = Gleichung ,
6161     Name-pl = Gleichungen ,
6162     case = D ,
6163     Name-sg = Gleichung ,
6164     Name-pl = Gleichungen ,
6165     case = G ,
6166     Name-sg = Gleichung ,
6167     Name-pl = Gleichungen ,
6168     refbounds-first-sg = {,(,)}, ,
6169     refbounds = {(,,)} ,
6170
6171 type = theorem ,

```

```

6172 gender = n ,
6173 case = N ,
6174     Name-sg = Theorem ,
6175     Name-pl = Theoreme ,
6176 case = A ,
6177     Name-sg = Theorem ,
6178     Name-pl = Theoreme ,
6179 case = D ,
6180     Name-sg = Theorem ,
6181     Name-pl = Theoremen ,
6182 case = G ,
6183     Name-sg = Theorems ,
6184     Name-pl = Theoreme ,
6185
6186 type = lemma ,
6187 gender = n ,
6188 case = N ,
6189     Name-sg = Lemma ,
6190     Name-pl = Lemmata ,
6191 case = A ,
6192     Name-sg = Lemma ,
6193     Name-pl = Lemmata ,
6194 case = D ,
6195     Name-sg = Lemma ,
6196     Name-pl = Lemmata ,
6197 case = G ,
6198     Name-sg = Lemmas ,
6199     Name-pl = Lemmata ,
6200
6201 type = corollary ,
6202 gender = n ,
6203 case = N ,
6204     Name-sg = Korollar ,
6205     Name-pl = Korollare ,
6206 case = A ,
6207     Name-sg = Korollar ,
6208     Name-pl = Korollare ,
6209 case = D ,
6210     Name-sg = Korollar ,
6211     Name-pl = Korollaren ,
6212 case = G ,
6213     Name-sg = Korollars ,
6214     Name-pl = Korollare ,
6215
6216 type = proposition ,
6217 gender = m ,
6218 case = N ,
6219     Name-sg = Satz ,
6220     Name-pl = Sätze ,
6221 case = A ,
6222     Name-sg = Satz ,
6223     Name-pl = Sätze ,
6224 case = D ,
6225     Name-sg = Satz ,

```

```

6226     Name-pl = Sätzen ,
6227     case = G ,
6228     Name-sg = Satzes ,
6229     Name-pl = Sätze ,
6230
6231 type = definition ,
6232     gender = f ,
6233     case = N ,
6234     Name-sg = Definition ,
6235     Name-pl = Definitionen ,
6236     case = A ,
6237     Name-sg = Definition ,
6238     Name-pl = Definitionen ,
6239     case = D ,
6240     Name-sg = Definition ,
6241     Name-pl = Definitionen ,
6242     case = G ,
6243     Name-sg = Definition ,
6244     Name-pl = Definitionen ,
6245
6246 type = proof ,
6247     gender = m ,
6248     case = N ,
6249     Name-sg = Beweis ,
6250     Name-pl = Beweise ,
6251     case = A ,
6252     Name-sg = Beweis ,
6253     Name-pl = Beweise ,
6254     case = D ,
6255     Name-sg = Beweis ,
6256     Name-pl = Beweisen ,
6257     case = G ,
6258     Name-sg = Beweises ,
6259     Name-pl = Beweise ,
6260
6261 type = result ,
6262     gender = n ,
6263     case = N ,
6264     Name-sg = Ergebnis ,
6265     Name-pl = Ergebnisse ,
6266     case = A ,
6267     Name-sg = Ergebnis ,
6268     Name-pl = Ergebnisse ,
6269     case = D ,
6270     Name-sg = Ergebnis ,
6271     Name-pl = Ergebnissen ,
6272     case = G ,
6273     Name-sg = Ergebnisses ,
6274     Name-pl = Ergebnisse ,
6275
6276 type = remark ,
6277     gender = f ,
6278     case = N ,
6279     Name-sg = Bemerkung ,

```

```

6280     Name-pl = Bemerkungen ,
6281     case = A ,
6282     Name-sg = Bemerkung ,
6283     Name-pl = Bemerkungen ,
6284     case = D ,
6285     Name-sg = Bemerkung ,
6286     Name-pl = Bemerkungen ,
6287     case = G ,
6288     Name-sg = Bemerkung ,
6289     Name-pl = Bemerkungen ,
6290
6291 type = example ,
6292     gender = n ,
6293     case = N ,
6294     Name-sg = Beispiel ,
6295     Name-pl = Beispiele ,
6296     case = A ,
6297     Name-sg = Beispiel ,
6298     Name-pl = Beispiele ,
6299     case = D ,
6300     Name-sg = Beispiel ,
6301     Name-pl = Beispielen ,
6302     case = G ,
6303     Name-sg = Beispiels ,
6304     Name-pl = Beispiele ,
6305
6306 type = algorithm ,
6307     gender = m ,
6308     case = N ,
6309     Name-sg = Algorithmus ,
6310     Name-pl = Algorithmen ,
6311     case = A ,
6312     Name-sg = Algorithmus ,
6313     Name-pl = Algorithmen ,
6314     case = D ,
6315     Name-sg = Algorithmus ,
6316     Name-pl = Algorithmen ,
6317     case = G ,
6318     Name-sg = Algorithmus ,
6319     Name-pl = Algorithmen ,
6320
6321 type = listing ,
6322     gender = n ,
6323     case = N ,
6324     Name-sg = Listing ,
6325     Name-pl = Listings ,
6326     case = A ,
6327     Name-sg = Listing ,
6328     Name-pl = Listings ,
6329     case = D ,
6330     Name-sg = Listing ,
6331     Name-pl = Listings ,
6332     case = G ,
6333     Name-sg = Listings ,

```



```

6334     Name-pl = Listings ,
6335
6336 type = exercise ,
6337     gender = f ,
6338     case = N ,
6339     Name-sg = Übungsaufgabe ,
6340     Name-pl = Übungsaufgaben ,
6341     case = A ,
6342     Name-sg = Übungsaufgabe ,
6343     Name-pl = Übungsaufgaben ,
6344     case = D ,
6345     Name-sg = Übungsaufgabe ,
6346     Name-pl = Übungsaufgaben ,
6347     case = G ,
6348     Name-sg = Übungsaufgabe ,
6349     Name-pl = Übungsaufgaben ,
6350
6351 type = solution ,
6352     gender = f ,
6353     case = N ,
6354     Name-sg = Lösung ,
6355     Name-pl = Lösungen ,
6356     case = A ,
6357     Name-sg = Lösung ,
6358     Name-pl = Lösungen ,
6359     case = D ,
6360     Name-sg = Lösung ,
6361     Name-pl = Lösungen ,
6362     case = G ,
6363     Name-sg = Lösung ,
6364     Name-pl = Lösungen ,
6365 </lang-german>

```

10.4 French

French language file has been initially provided by the author, and has been improved thanks to Denis Bitouzé and François Lagarde (at issue [#1](#)) and participants of the Groupe francophone des Utilisateurs de T_EX (GUTenberg) (at https://groups.google.com/g/gut_fr/c/rNlm6weGcyg) and the fr.comp.text.tex (at <https://groups.google.com/g/fr.comp.text.tex/c/Fa11Tf6MFFs>) mailing lists.

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

```

6366 <*package>
6367 \zcDeclareLanguage [ gender = { f , m } ] { french }
6368 \zcDeclareLanguageAlias { acadian } { french }
6369 </package>
6370 <*lang-french>
6371 namesep = {\nobreakspace} ,
6372 pairsep = {\et\nobreakspace} ,
6373 listsep = {,~} ,
6374 lastsep = {\et\nobreakspace} ,
6375 tpairsep = {\et\nobreakspace} ,

```

```

6376 tlistsep = {,~} ,
6377 tlastsep = {~et\nobreakspace} ,
6378 notesep = {~} ,
6379 rangesep = {~à\nobreakspace} ,
6380
6381 type = book ,
6382     gender = m ,
6383     Name-sg = Livre ,
6384     name-sg = livre ,
6385     Name-pl = Livres ,
6386     name-pl = livres ,
6387
6388 type = part ,
6389     gender = f ,
6390     Name-sg = Partie ,
6391     name-sg = partie ,
6392     Name-pl = Parties ,
6393     name-pl = parties ,
6394
6395 type = chapter ,
6396     gender = m ,
6397     Name-sg = Chapitre ,
6398     name-sg = chapitre ,
6399     Name-pl = Chapitres ,
6400     name-pl = chapitres ,
6401
6402 type = section ,
6403     gender = f ,
6404     Name-sg = Section ,
6405     name-sg = section ,
6406     Name-pl = Sections ,
6407     name-pl = sections ,
6408
6409 type = paragraph ,
6410     gender = m ,
6411     Name-sg = Paragraphe ,
6412     name-sg = paragraphe ,
6413     Name-pl = Paragraphes ,
6414     name-pl = paragraphes ,
6415
6416 type = appendix ,
6417     gender = f ,
6418     Name-sg = Annexe ,
6419     name-sg = annexe ,
6420     Name-pl = Annexes ,
6421     name-pl = annexes ,
6422
6423 type = page ,
6424     gender = f ,
6425     Name-sg = Page ,
6426     name-sg = page ,
6427     Name-pl = Pages ,
6428     name-pl = pages ,
6429     rangesep = {-} ,

```

```

6430   rangetopair = false ,
6431
6432   type = line ,
6433     gender = f ,
6434     Name-sg = Ligne ,
6435     name-sg = ligne ,
6436     Name-pl = Lignes ,
6437     name-pl = lignes ,
6438
6439   type = figure ,
6440     gender = f ,
6441     Name-sg = Figure ,
6442     name-sg = figure ,
6443     Name-pl = Figures ,
6444     name-pl = figures ,
6445
6446   type = table ,
6447     gender = f ,
6448     Name-sg = Table ,
6449     name-sg = table ,
6450     Name-pl = Tables ,
6451     name-pl = tables ,
6452
6453   type = item ,
6454     gender = m ,
6455     Name-sg = Point ,
6456     name-sg = point ,
6457     Name-pl = Points ,
6458     name-pl = points ,
6459
6460   type = footnote ,
6461     gender = f ,
6462     Name-sg = Note ,
6463     name-sg = note ,
6464     Name-pl = Notes ,
6465     name-pl = notes ,
6466
6467   type = endnote ,
6468     gender = f ,
6469     Name-sg = Note ,
6470     name-sg = note ,
6471     Name-pl = Notes ,
6472     name-pl = notes ,
6473
6474   type = note ,
6475     gender = f ,
6476     Name-sg = Note ,
6477     name-sg = note ,
6478     Name-pl = Notes ,
6479     name-pl = notes ,
6480
6481   type = equation ,
6482     gender = f ,
6483     Name-sg = Équation ,

```

```

6484 name-sg = équation ,
6485 Name-pl = Équations ,
6486 name-pl = équations ,
6487 refbounds-first-sg = {,(,)}, ,
6488 refbounds = {(,,)} ,
6489
6490 type = theorem ,
6491 gender = m ,
6492 Name-sg = Théorème ,
6493 name-sg = théorème ,
6494 Name-pl = Théorèmes ,
6495 name-pl = théorèmes ,
6496
6497 type = lemma ,
6498 gender = m ,
6499 Name-sg = Lemme ,
6500 name-sg = lemme ,
6501 Name-pl = Lemmes ,
6502 name-pl = lemmes ,
6503
6504 type = corollary ,
6505 gender = m ,
6506 Name-sg = Corollaire ,
6507 name-sg = corollaire ,
6508 Name-pl = Corollaires ,
6509 name-pl = corollaires ,
6510
6511 type = proposition ,
6512 gender = f ,
6513 Name-sg = Proposition ,
6514 name-sg = proposition ,
6515 Name-pl = Propositions ,
6516 name-pl = propositions ,
6517
6518 type = definition ,
6519 gender = f ,
6520 Name-sg = Définition ,
6521 name-sg = définition ,
6522 Name-pl = Définitions ,
6523 name-pl = définitions ,
6524
6525 type = proof ,
6526 gender = f ,
6527 Name-sg = Démonstration ,
6528 name-sg = démonstration ,
6529 Name-pl = Démonstrations ,
6530 name-pl = démonstrations ,
6531
6532 type = result ,
6533 gender = m ,
6534 Name-sg = Résultat ,
6535 name-sg = résultat ,
6536 Name-pl = Résultats ,
6537 name-pl = résultats ,

```

```

6538
6539 type = remark ,
6540   gender = f ,
6541   Name-sg = Remarque ,
6542   name-sg = remarque ,
6543   Name-pl = Remarques ,
6544   name-pl = remarques ,
6545
6546 type = example ,
6547   gender = m ,
6548   Name-sg = Exemple ,
6549   name-sg = exemple ,
6550   Name-pl = Exemples ,
6551   name-pl = exemples ,
6552
6553 type = algorithm ,
6554   gender = m ,
6555   Name-sg = Algorithmme ,
6556   name-sg = algorithmme ,
6557   Name-pl = Algorithmmes ,
6558   name-pl = algorithmmes ,
6559
6560 type = listing ,
6561   gender = m ,
6562   Name-sg = Listing ,
6563   name-sg = listing ,
6564   Name-pl = Listings ,
6565   name-pl = listings ,
6566
6567 type = exercise ,
6568   gender = m ,
6569   Name-sg = Exercice ,
6570   name-sg = exercice ,
6571   Name-pl = Exercices ,
6572   name-pl = exercices ,
6573
6574 type = solution ,
6575   gender = f ,
6576   Name-sg = Solution ,
6577   name-sg = solution ,
6578   Name-pl = Solutions ,
6579   name-pl = solutions ,
6580 </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.

```

6581 (*package)
6582 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6583 \zcDeclareLanguageAlias { brazilian } { portuguese }
6584 \zcDeclareLanguageAlias { brazil   } { portuguese }

```

```

6585 \zcDeclareLanguageAlias { portugues } { portuguese }
6586 \end{package}
6587 \begin{lang-portuguese}
6588 namesep = {\nobreakspace} ,
6589 pairsep = {\e\nobreakspace} ,
6590 listsep = {,~} ,
6591 lastsep = {\e\nobreakspace} ,
6592 tpairsep = {\e\nobreakspace} ,
6593 tlistsep = {,~} ,
6594 tlastsep = {\e\nobreakspace} ,
6595 notesep = {~} ,
6596 rangesep = {\a\nobreakspace} ,
6597
6598 type = book ,
6599   gender = m ,
6600   Name-sg = Livro ,
6601   name-sg = livro ,
6602   Name-pl = Livros ,
6603   name-pl = livros ,
6604
6605 type = part ,
6606   gender = f ,
6607   Name-sg = Parte ,
6608   name-sg = parte ,
6609   Name-pl = Partes ,
6610   name-pl = partes ,
6611
6612 type = chapter ,
6613   gender = m ,
6614   Name-sg = Capítulo ,
6615   name-sg = capítulo ,
6616   Name-pl = Capítulos ,
6617   name-pl = capítulos ,
6618
6619 type = section ,
6620   gender = f ,
6621   Name-sg = Seção ,
6622   name-sg = seção ,
6623   Name-pl = Seções ,
6624   name-pl = seções ,
6625
6626 type = paragraph ,
6627   gender = m ,
6628   Name-sg = Parágrafo ,
6629   name-sg = parágrafo ,
6630   Name-pl = Parágrafos ,
6631   name-pl = parágrafos ,
6632   Name-sg-ab = Par. ,
6633   name-sg-ab = par. ,
6634   Name-pl-ab = Par. ,
6635   name-pl-ab = par. ,
6636
6637 type = appendix ,

```

```

6638     gender = m ,
6639     Name-sg = Apêndice ,
6640     name-sg = apêndice ,
6641     Name-pl = Apêndices ,
6642     name-pl = apêndices ,
6643
6644     type = page ,
6645     gender = f ,
6646     Name-sg = Página ,
6647     name-sg = página ,
6648     Name-pl = Páginas ,
6649     name-pl = páginas ,
6650     rangesep = {\textendash} ,
6651     rangetopair = false ,
6652
6653     type = line ,
6654     gender = f ,
6655     Name-sg = Linha ,
6656     name-sg = linha ,
6657     Name-pl = Linhas ,
6658     name-pl = linhas ,
6659
6660     type = figure ,
6661     gender = f ,
6662     Name-sg = Figura ,
6663     name-sg = figura ,
6664     Name-pl = Figuras ,
6665     name-pl = figuras ,
6666     Name-sg-ab = Fig. ,
6667     name-sg-ab = fig. ,
6668     Name-pl-ab = Figs. ,
6669     name-pl-ab = figs. ,
6670
6671     type = table ,
6672     gender = f ,
6673     Name-sg = Tabela ,
6674     name-sg = tabela ,
6675     Name-pl = Tabelas ,
6676     name-pl = tabelas ,
6677
6678     type = item ,
6679     gender = m ,
6680     Name-sg = Item ,
6681     name-sg = item ,
6682     Name-pl = Itens ,
6683     name-pl = itens ,
6684
6685     type = footnote ,
6686     gender = f ,
6687     Name-sg = Nota ,
6688     name-sg = nota ,
6689     Name-pl = Notas ,
6690     name-pl = notas ,
6691

```

```

6692 type = endnote ,
6693     gender = f ,
6694     Name-sg = Nota ,
6695     name-sg = nota ,
6696     Name-pl = Notas ,
6697     name-pl = notas ,
6698
6699 type = note ,
6700     gender = f ,
6701     Name-sg = Nota ,
6702     name-sg = nota ,
6703     Name-pl = Notas ,
6704     name-pl = notas ,
6705
6706 type = equation ,
6707     gender = f ,
6708     Name-sg = Equação ,
6709     name-sg = equação ,
6710     Name-pl = Equações ,
6711     name-pl = equações ,
6712     Name-sg-ab = Eq. ,
6713     name-sg-ab = eq. ,
6714     Name-pl-ab = Eqs. ,
6715     name-pl-ab = eqs. ,
6716     refbounds-first-sg = {,(,)}, ,
6717     refbounds = {(,,)} ,
6718
6719 type = theorem ,
6720     gender = m ,
6721     Name-sg = Teorema ,
6722     name-sg = teorema ,
6723     Name-pl = Teoremas ,
6724     name-pl = teoremas ,
6725
6726 type = lemma ,
6727     gender = m ,
6728     Name-sg = Lema ,
6729     name-sg = lema ,
6730     Name-pl = Lemas ,
6731     name-pl = lemas ,
6732
6733 type = corollary ,
6734     gender = m ,
6735     Name-sg = Corolário ,
6736     name-sg = corolário ,
6737     Name-pl = Corolários ,
6738     name-pl = corolários ,
6739
6740 type = proposition ,
6741     gender = f ,
6742     Name-sg = Proposição ,
6743     name-sg = proposição ,
6744     Name-pl = Proposições ,
6745     name-pl = proposições ,

```



```

6746
6747 type = definition ,
6748     gender = f ,
6749     Name-sg = Definição ,
6750     name-sg = definição ,
6751     Name-pl = Definições ,
6752     name-pl = definições ,
6753
6754 type = proof ,
6755     gender = f ,
6756     Name-sg = Demonstração ,
6757     name-sg = demonstração ,
6758     Name-pl = Demonstrações ,
6759     name-pl = demonstrações ,
6760
6761 type = result ,
6762     gender = m ,
6763     Name-sg = Resultado ,
6764     name-sg = resultado ,
6765     Name-pl = Resultados ,
6766     name-pl = resultados ,
6767
6768 type = remark ,
6769     gender = f ,
6770     Name-sg = Observação ,
6771     name-sg = observação ,
6772     Name-pl = Observações ,
6773     name-pl = observações ,
6774
6775 type = example ,
6776     gender = m ,
6777     Name-sg = Exemplo ,
6778     name-sg = exemplo ,
6779     Name-pl = Exemplos ,
6780     name-pl = exemplos ,
6781
6782 type = algorithm ,
6783     gender = m ,
6784     Name-sg = Algoritmo ,
6785     name-sg = algoritmo ,
6786     Name-pl = Algoritmos ,
6787     name-pl = algoritmos ,
6788
6789 type = listing ,
6790     gender = f ,
6791     Name-sg = Listagem ,
6792     name-sg = listagem ,
6793     Name-pl = Listagens ,
6794     name-pl = listagens ,
6795
6796 type = exercise ,
6797     gender = m ,
6798     Name-sg = Exercício ,
6799     name-sg = exercício ,

```

```

6800 Name-pl = Exercícios ,
6801 name-pl = exercícios ,
6802
6803 type = solution ,
6804 gender = f ,
6805 Name-sg = Solução ,
6806 name-sg = solução ,
6807 Name-pl = Soluções ,
6808 name-pl = soluções ,
6809 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```

6810 <*package>
6811 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6812 </package>
6813 <*lang-spanish>
6814 namesep = {\nobreakspace} ,
6815 pairsep = {\~y\nobreakspace} ,
6816 listsep = {,~} ,
6817 lastsep = {\~y\nobreakspace} ,
6818 tpairsep = {\~y\nobreakspace} ,
6819 tlistsep = {,~} ,
6820 tlastsep = {\~y\nobreakspace} ,
6821 notesep = {\~} ,
6822 rangesep = {\~a\nobreakspace} ,
6823
6824 type = book ,
6825 gender = m ,
6826 Name-sg = Libro ,
6827 name-sg = libro ,
6828 Name-pl = Libros ,
6829 name-pl = libros ,
6830
6831 type = part ,
6832 gender = f ,
6833 Name-sg = Parte ,
6834 name-sg = parte ,
6835 Name-pl = Partes ,
6836 name-pl = partes ,
6837
6838 type = chapter ,
6839 gender = m ,
6840 Name-sg = Capítulo ,
6841 name-sg = capítulo ,
6842 Name-pl = Capítulos ,
6843 name-pl = capítulos ,
6844
6845 type = section ,
6846 gender = f ,
6847 Name-sg = Sección ,

```

```

6848 name-sg = sección ,
6849 Name-pl = Secciones ,
6850 name-pl = secciones ,
6851
6852 type = paragraph ,
6853 gender = m ,
6854 Name-sg = Párrafo ,
6855 name-sg = párrafo ,
6856 Name-pl = Párrafos ,
6857 name-pl = párrafos ,
6858
6859 type = appendix ,
6860 gender = m ,
6861 Name-sg = Apéndice ,
6862 name-sg = apéndice ,
6863 Name-pl = Apéndices ,
6864 name-pl = apéndices ,
6865
6866 type = page ,
6867 gender = f ,
6868 Name-sg = Página ,
6869 name-sg = página ,
6870 Name-pl = Páginas ,
6871 name-pl = páginas ,
6872 rangeseq = {\textendash} ,
6873 rangetopair = false ,
6874
6875 type = line ,
6876 gender = f ,
6877 Name-sg = Línea ,
6878 name-sg = línea ,
6879 Name-pl = Líneas ,
6880 name-pl = líneas ,
6881
6882 type = figure ,
6883 gender = f ,
6884 Name-sg = Figura ,
6885 name-sg = figura ,
6886 Name-pl = Figuras ,
6887 name-pl = figuras ,
6888
6889 type = table ,
6890 gender = m ,
6891 Name-sg = Cuadro ,
6892 name-sg = cuadro ,
6893 Name-pl = Cuadros ,
6894 name-pl = cuadros ,
6895
6896 type = item ,
6897 gender = m ,
6898 Name-sg = Punto ,
6899 name-sg = punto ,
6900 Name-pl = Puntos ,
6901 name-pl = puntos ,

```

```

6902
6903 type = footnote ,
6904     gender = f ,
6905     Name-sg = Nota ,
6906     name-sg = nota ,
6907     Name-pl = Notas ,
6908     name-pl = notas ,
6909
6910 type = endnote ,
6911     gender = f ,
6912     Name-sg = Nota ,
6913     name-sg = nota ,
6914     Name-pl = Notas ,
6915     name-pl = notas ,
6916
6917 type = note ,
6918     gender = f ,
6919     Name-sg = Nota ,
6920     name-sg = nota ,
6921     Name-pl = Notas ,
6922     name-pl = notas ,
6923
6924 type = equation ,
6925     gender = f ,
6926     Name-sg = Ecuación ,
6927     name-sg = ecuación ,
6928     Name-pl = Ecuaciones ,
6929     name-pl = ecuaciones ,
6930     refbounds-first-sg = {,(,)}, ,
6931     refbounds = {(,,)} ,
6932
6933 type = theorem ,
6934     gender = m ,
6935     Name-sg = Teorema ,
6936     name-sg = teorema ,
6937     Name-pl = Teoremas ,
6938     name-pl = teoremas ,
6939
6940 type = lemma ,
6941     gender = m ,
6942     Name-sg = Lema ,
6943     name-sg = lema ,
6944     Name-pl = Lemas ,
6945     name-pl = lemas ,
6946
6947 type = corollary ,
6948     gender = m ,
6949     Name-sg = Corolario ,
6950     name-sg = corolario ,
6951     Name-pl = Corolarios ,
6952     name-pl = corolarios ,
6953
6954 type = proposition ,
6955     gender = f ,

```

```

6956 Name-sg = Proposición ,
6957 name-sg = proposición ,
6958 Name-pl = Proposiciones ,
6959 name-pl = proposiciones ,
6960
6961 type = definition ,
6962   gender = f ,
6963   Name-sg = Definición ,
6964   name-sg = definición ,
6965   Name-pl = Definiciones ,
6966   name-pl = definiciones ,
6967
6968 type = proof ,
6969   gender = f ,
6970   Name-sg = Demostración ,
6971   name-sg = demostración ,
6972   Name-pl = Demostraciones ,
6973   name-pl = demostraciones ,
6974
6975 type = result ,
6976   gender = m ,
6977   Name-sg = Resultado ,
6978   name-sg = resultado ,
6979   Name-pl = Resultados ,
6980   name-pl = resultados ,
6981
6982 type = remark ,
6983   gender = f ,
6984   Name-sg = Observación ,
6985   name-sg = observación ,
6986   Name-pl = Observaciones ,
6987   name-pl = observaciones ,
6988
6989 type = example ,
6990   gender = m ,
6991   Name-sg = Ejemplo ,
6992   name-sg = ejemplo ,
6993   Name-pl = Ejemplos ,
6994   name-pl = ejemplos ,
6995
6996 type = algorithm ,
6997   gender = m ,
6998   Name-sg = Algoritmo ,
6999   name-sg = algoritmo ,
7000   Name-pl = Algoritmos ,
7001   name-pl = algoritmos ,
7002
7003 type = listing ,
7004   gender = m ,
7005   Name-sg = Listado ,
7006   name-sg = listado ,
7007   Name-pl = Listados ,
7008   name-pl = listados ,
7009

```

```

7010 type = exercise ,
7011   gender = m ,
7012   Name-sg = Ejercicio ,
7013   name-sg = ejercicio ,
7014   Name-pl = Ejercicios ,
7015   name-pl = ejercicios ,
7016
7017 type = solution ,
7018   gender = f ,
7019   Name-sg = Solución ,
7020   name-sg = solución ,
7021   Name-pl = Soluciones ,
7022   name-pl = soluciones ,
7023 </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.

```

7024 <*package>
7025 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7026 </package>
7027 <*lang-dutch>
7028 namesep = {\nobreakspace} ,
7029 pairsep = {\~en\nobreakspace} ,
7030 listsep = { ,~ } ,
7031 lastsep = {\~en\nobreakspace} ,
7032 tpairsep = {\~en\nobreakspace} ,
7033 tlistsep = { ,~ } ,
7034 tlastsep = { ,~en\nobreakspace} ,
7035 notesep = {~} ,
7036 rangesep = {\~t/m\nobreakspace} ,
7037
7038 type = book ,
7039   gender = n ,
7040   Name-sg = Boek ,
7041   name-sg = boek ,
7042   Name-pl = Boeken ,
7043   name-pl = boeken ,
7044
7045 type = part ,
7046   gender = n ,
7047   Name-sg = Deel ,
7048   name-sg = deel ,
7049   Name-pl = Delen ,
7050   name-pl = delen ,
7051
7052 type = chapter ,
7053   gender = n ,
7054   Name-sg = Hoofdstuk ,
7055   name-sg = hoofdstuk ,
7056   Name-pl = Hoofdstukken ,

```

```

7057     name-pl = hoofdstukken ,
7058
7059 type = section ,
7060     gender = m ,
7061     Name-sg = Paragraaf ,
7062     name-sg = paragraaf ,
7063     Name-pl = Paragrafen ,
7064     name-pl = paragrafen ,
7065
7066 type = paragraph ,
7067     gender = f ,
7068     Name-sg = Alinea ,
7069     name-sg = alinea ,
7070     Name-pl = Alinea's ,
7071     name-pl = alinea's ,
7072

```

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.

```

7073 type = appendix ,
7074     gender = { f , m } ,
7075     Name-sg = Bijlage ,
7076     name-sg = bijlage ,
7077     Name-pl = Bijlagen ,
7078     name-pl = bijlagen ,
7079
7080 type = page ,
7081     gender = { f , m } ,
7082     Name-sg = Pagina ,
7083     name-sg = pagina ,
7084     Name-pl = Pagina's ,
7085     name-pl = pagina's ,
7086     rangesep = {\textendash} ,
7087     rangetopair = false ,
7088
7089 type = line ,
7090     gender = m ,
7091     Name-sg = Regel ,
7092     name-sg = regel ,
7093     Name-pl = Regels ,
7094     name-pl = regels ,
7095
7096 type = figure ,
7097     gender = { n , f , m } ,
7098     Name-sg = Figuur ,
7099     name-sg = figuur ,
7100     Name-pl = Figuren ,
7101     name-pl = figuren ,
7102
7103 type = table ,
7104     gender = { f , m } ,
7105     Name-sg = Tabel ,
7106     name-sg = tabel ,

```

```

7107 Name-pl = Tabellen ,
7108 name-pl = tabellen ,
7109
7110 type = item ,
7111 gender = n ,
7112 Name-sg = Punt ,
7113 name-sg = punt ,
7114 Name-pl = Punten ,
7115 name-pl = punten ,
7116
7117 type = footnote ,
7118 gender = { f , m } ,
7119 Name-sg = Voetnoot ,
7120 name-sg = voetnoot ,
7121 Name-pl = Voetnoten ,
7122 name-pl = voetnoten ,
7123
7124 type = endnote ,
7125 gender = { f , m } ,
7126 Name-sg = Eindnoot ,
7127 name-sg = eindnoot ,
7128 Name-pl = Eindnoten ,
7129 name-pl = eindnoten ,
7130
7131 type = note ,
7132 gender = f ,
7133 Name-sg = Opmerking ,
7134 name-sg = opmerking ,
7135 Name-pl = Opmerkingen ,
7136 name-pl = opmerkingen ,
7137
7138 type = equation ,
7139 gender = f ,
7140 Name-sg = Vergelijking ,
7141 name-sg = vergelijking ,
7142 Name-pl = Vergelijkingen ,
7143 name-pl = vergelijkingen ,
7144 Name-sg-ab = Vgl. ,
7145 name-sg-ab = vgl. ,
7146 Name-pl-ab = Vgl.'s ,
7147 name-pl-ab = vgl.'s ,
7148 refbounds-first-sg = {,(,)}, ,
7149 refbounds = {(,,)} ,
7150
7151 type = theorem ,
7152 gender = f ,
7153 Name-sg = Stelling ,
7154 name-sg = stelling ,
7155 Name-pl = Stellingen ,
7156 name-pl = stellingen ,
7157

```

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”.

```
7158 type = lemma ,
7159     gender = n ,
7160     Name-sg = Lemma ,
7161     name-sg = lemma ,
7162     Name-pl = Lemma's ,
7163     name-pl = lemma's ,
7164
7165 type = corollary ,
7166     gender = n ,
7167     Name-sg = Gevolg ,
7168     name-sg = gevolg ,
7169     Name-pl = Gevolgen ,
7170     name-pl = gevolgen ,
7171
7172 type = proposition ,
7173     gender = f ,
7174     Name-sg = Propositie ,
7175     name-sg = propositie ,
7176     Name-pl = Propositions ,
7177     name-pl = proposities ,
7178
7179 type = definition ,
7180     gender = f ,
7181     Name-sg = Definitie ,
7182     name-sg = definitie ,
7183     Name-pl = Definities ,
7184     name-pl = definities ,
7185
7186 type = proof ,
7187     gender = n ,
7188     Name-sg = Bewijs ,
7189     name-sg = bewijs ,
7190     Name-pl = Bewijzen ,
7191     name-pl = bewijzen ,
7192
7193 type = result ,
7194     gender = n ,
7195     Name-sg = Resultaat ,
7196     name-sg = resultaat ,
7197     Name-pl = Resultaten ,
7198     name-pl = resultaten ,
7199
7200 type = remark ,
7201     gender = f ,
7202     Name-sg = Opmerking ,
7203     name-sg = opmerking ,
7204     Name-pl = Opmerkingen ,
7205     name-pl = opmerkingen ,
7206
7207 type = example ,
7208     gender = n ,
7209     Name-sg = Voorbeeld ,
7210     name-sg = voorbeeld ,
```

```

7211 Name-pl = Voorbeelden ,
7212 name-pl = voorbeelden ,
7213

```

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”.

```

7214 type = algorithm ,
7215 gender = { n , f , m } ,
7216 Name-sg = Algoritme ,
7217 name-sg = algoritme ,
7218 Name-pl = Algoritmen ,
7219 name-pl = algoritmen ,
7220

```

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

```

7221 type = listing ,
7222 gender = m ,
7223 Name-sg = Listing ,
7224 name-sg = listing ,
7225 Name-pl = Listings ,
7226 name-pl = listings ,
7227
7228 type = exercise ,
7229 gender = { f , m } ,
7230 Name-sg = Opgave ,
7231 name-sg = opgave ,
7232 Name-pl = Opgaven ,
7233 name-pl = opgaven ,
7234
7235 type = solution ,
7236 gender = f ,
7237 Name-sg = Oplossing ,
7238 name-sg = oplossing ,
7239 Name-pl = Oplossingen ,
7240 name-pl = oplossingen ,
7241 </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->

```

7242 <*package>
7243 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7244 </package>
7245 <*lang-italian>
7246 namesep = {\nobreakspace} ,
7247 pairsep = {\nobreakspace} ,
7248 listsep = {,~} ,
7249 lastsep = {\nobreakspace} ,
7250 tpairsep = {\nobreakspace} ,
7251 tlistsep = {,~} ,

```

```

7252 tlastsep = {,~e\nobreakspace} ,
7253 notesep = {~} ,
7254 rangesep = {~a\nobreakspace} ,
7255 +refbounds-rb = {da\nobreakspace,,,} ,
7256
7257 type = book ,
7258     gender = m ,
7259     Name-sg = Libro ,
7260     name-sg = libro ,
7261     Name-pl = Libri ,
7262     name-pl = libri ,
7263
7264 type = part ,
7265     gender = f ,
7266     Name-sg = Parte ,
7267     name-sg = parte ,
7268     Name-pl = Parti ,
7269     name-pl = parti ,
7270
7271 type = chapter ,
7272     gender = m ,
7273     Name-sg = Capitolo ,
7274     name-sg = capitolo ,
7275     Name-pl = Capitoli ,
7276     name-pl = capitoli ,
7277
7278 type = section ,
7279     gender = m ,
7280     Name-sg = Paragrafo ,
7281     name-sg = paragrafo ,
7282     Name-pl = Paragrafi ,
7283     name-pl = paragrafi ,
7284
7285 type = paragraph ,
7286     gender = m ,
7287     Name-sg = Capoverso ,
7288     name-sg = capoverso ,
7289     Name-pl = Capoversi ,
7290     name-pl = capoversi ,
7291
7292 type = appendix ,
7293     gender = f ,
7294     Name-sg = Appendice ,
7295     name-sg = appendice ,
7296     Name-pl = Appendici ,
7297     name-pl = appendici ,
7298
7299 type = page ,
7300     gender = f ,
7301     Name-sg = Pagina ,
7302     name-sg = pagina ,
7303     Name-pl = Pagine ,
7304     name-pl = pagine ,
7305     Name-sg-ab = Pag. ,

```

```

7306 name-sg-ab = pag. ,
7307 Name-pl-ab = Pag. ,
7308 name-pl-ab = pag. ,
7309 rangesep = {\textendash} ,
7310 rangetopair = false ,
7311 +refbounds-rb = {,,} ,
7312
7313 type = line ,
7314 gender = f ,
7315 Name-sg = Riga ,
7316 name-sg = riga ,
7317 Name-pl = Righe ,
7318 name-pl = righe ,
7319
7320 type = figure ,
7321 gender = f ,
7322 Name-sg = Figura ,
7323 name-sg = figura ,
7324 Name-pl = Figure ,
7325 name-pl = figure ,
7326 Name-sg-ab = Fig. ,
7327 name-sg-ab = fig. ,
7328 Name-pl-ab = Fig. ,
7329 name-pl-ab = fig. ,
7330
7331 type = table ,
7332 gender = f ,
7333 Name-sg = Tabella ,
7334 name-sg = tabella ,
7335 Name-pl = Tabelle ,
7336 name-pl = tabelle ,
7337 Name-sg-ab = Tab. ,
7338 name-sg-ab = tab. ,
7339 Name-pl-ab = Tab. ,
7340 name-pl-ab = tab. ,
7341
7342 type = item ,
7343 gender = m ,
7344 Name-sg = Punto ,
7345 name-sg = punto ,
7346 Name-pl = Punti ,
7347 name-pl = punti ,
7348
7349 type = footnote ,
7350 gender = f ,
7351 Name-sg = Nota ,
7352 name-sg = nota ,
7353 Name-pl = Note ,
7354 name-pl = note ,
7355
7356 type = endnote ,
7357 gender = f ,
7358 Name-sg = Nota ,
7359 name-sg = nota ,

```

```

7360 Name-pl = Note ,
7361 name-pl = note ,
7362
7363 type = note ,
7364 gender = f ,
7365 Name-sg = Nota ,
7366 name-sg = nota ,
7367 Name-pl = Note ,
7368 name-pl = note ,
7369
7370 type = equation ,
7371 gender = f ,
7372 Name-sg = Equazione ,
7373 name-sg = equazione ,
7374 Name-pl = Equazioni ,
7375 name-pl = equazioni ,
7376 Name-sg-ab = Eq. ,
7377 name-sg-ab = eq. ,
7378 Name-pl-ab = Eq. ,
7379 name-pl-ab = eq. ,
7380 +refbounds-rb = {da\nobreakspace(,,)} ,
7381 refbounds-first-sg = {(,)} ,
7382 refbounds = {(,,)} ,
7383
7384 type = theorem ,
7385 gender = m ,
7386 Name-sg = Teorema ,
7387 name-sg = teorema ,
7388 Name-pl = Teoremi ,
7389 name-pl = teoremi ,
7390
7391 type = lemma ,
7392 gender = m ,
7393 Name-sg = Lemma ,
7394 name-sg = lemma ,
7395 Name-pl = Lemmi ,
7396 name-pl = lemmi ,
7397
7398 type = corollary ,
7399 gender = m ,
7400 Name-sg = Corollario ,
7401 name-sg = corollario ,
7402 Name-pl = Corollari ,
7403 name-pl = corollari ,
7404
7405 type = proposition ,
7406 gender = f ,
7407 Name-sg = Proposizione ,
7408 name-sg = proposizione ,
7409 Name-pl = Proposizioni ,
7410 name-pl = proposizioni ,
7411
7412 type = definition ,
7413 gender = f ,

```

```

7414 Name-sg = Definizione ,
7415 name-sg = definizione ,
7416 Name-pl = Definizioni ,
7417 name-pl = definizioni ,
7418
7419 type = proof ,
7420 gender = f ,
7421 Name-sg = Dimostrazione ,
7422 name-sg = dimostrazione ,
7423 Name-pl = Dimostrazioni ,
7424 name-pl = dimostrazioni ,
7425
7426 type = result ,
7427 gender = m ,
7428 Name-sg = Risultato ,
7429 name-sg = risultato ,
7430 Name-pl = Risultati ,
7431 name-pl = risultati ,
7432
7433 type = remark ,
7434 gender = f ,
7435 Name-sg = Osservazione ,
7436 name-sg = osservazione ,
7437 Name-pl = Osservazioni ,
7438 name-pl = osservazioni ,
7439
7440 type = example ,
7441 gender = m ,
7442 Name-sg = Esempio ,
7443 name-sg = esempio ,
7444 Name-pl = Esempi ,
7445 name-pl = esempi ,
7446
7447 type = algorithm ,
7448 gender = m ,
7449 Name-sg = Algoritmo ,
7450 name-sg = algoritmo ,
7451 Name-pl = Algoritmi ,
7452 name-pl = algoritmi ,
7453
7454 type = listing ,
7455 gender = m ,
7456 Name-sg = Listato ,
7457 name-sg = listato ,
7458 Name-pl = Listati ,
7459 name-pl = listati ,
7460
7461 type = exercise ,
7462 gender = m ,
7463 Name-sg = Esercizio ,
7464 name-sg = esercizio ,
7465 Name-pl = Esercizi ,
7466 name-pl = esercizi ,
7467

```

```

7468 type = solution ,
7469   gender = f ,
7470   Name-sg = Soluzione ,
7471   name-sg = soluzione ,
7472   Name-pl = Soluzioni ,
7473   name-pl = soluzioni ,
7474 </lang-italian>

```

10.9 Russian

Russian language file initially contributed by Sergey Slyusarev ‘jemmybutton’ (PR #29). Russian localization is consistent with that of cleveref, with the following exceptions: “equation” is translated as “уравнение”, rather than “formula”, “proposition” is translated as “предложение”, rather than “утверждение”; several abbreviations are replaced with more common ones, e.g. abbreviated plural of “item” is “шт.”, not “п.п.”.

```

7475 (*package)
7476 \zcDeclareLanguage
7477   [ declension = { n , a , g , d , i , p } , gender = { f , m , n } ]
7478   { russian }
7479 </package>
7480 (*lang-russian)
7481 namesep = {\nobreakspace} ,
7482 pairsep = {\~\nobreakspace} ,
7483 listsep = {,~} ,
7484 lastsep = {\~\nobreakspace} ,
7485 tpairsep = {\~\nobreakspace} ,
7486 tlistsep = {,~} ,
7487 tlastsep = {,~\nobreakspace} ,
7488 notesep = {\~} ,
7489 rangesep = {\~\nobreakspace} ,
7490 +refbounds-rb = {c\nobreakspace,,,} ,
7491
7492 type = book ,
7493   gender = f ,
7494   case = n ,
7495     Name-sg = Книга ,
7496     name-sg = книга ,
7497     Name-pl = Книги ,
7498     name-pl = книги ,
7499   case = a ,
7500     Name-sg = Книгу ,
7501     name-sg = книгу ,
7502     Name-pl = Книги ,
7503     name-pl = книги ,
7504   case = g ,
7505     Name-sg = Книги ,
7506     name-sg = книги ,
7507     Name-pl = Книг ,
7508     name-pl = книг ,
7509   case = d ,
7510     Name-sg = Книге ,
7511     name-sg = книге ,

```

7512 Name-pl = Книгам ,
7513 name-pl = книгам ,
7514 case = i ,
7515 Name-sg = Книгой ,
7516 name-sg = книгой ,
7517 Name-pl = Книгами ,
7518 name-pl = книгами ,
7519 case = p ,
7520 Name-sg = Книге ,
7521 name-sg = книге ,
7522 Name-pl = Книгах ,
7523 name-pl = книгах ,
7524
7525 type = part ,
7526 gender = f ,
7527 case = n ,
7528 Name-sg = Часть ,
7529 name-sg = часть ,
7530 Name-pl = Части ,
7531 name-pl = части ,
7532 Name-sg-ab = Ч. ,
7533 name-sg-ab = ч. ,
7534 Name-pl-ab = Чч. ,
7535 name-pl-ab = чч. ,
7536 case = a ,
7537 Name-sg = Часть ,
7538 name-sg = часть ,
7539 Name-pl = Части ,
7540 name-pl = части ,
7541 Name-sg-ab = Ч. ,
7542 name-sg-ab = ч. ,
7543 Name-pl-ab = Чч. ,
7544 name-pl-ab = чч. ,
7545 case = g ,
7546 Name-sg = Части ,
7547 name-sg = части ,
7548 Name-pl = Частей ,
7549 name-pl = частей ,
7550 Name-sg-ab = Ч. ,
7551 name-sg-ab = ч. ,
7552 Name-pl-ab = Чч. ,
7553 name-pl-ab = чч. ,
7554 case = d ,
7555 Name-sg = Части ,
7556 name-sg = части ,
7557 Name-pl = Частям ,
7558 name-pl = частям ,
7559 Name-sg-ab = Ч. ,
7560 name-sg-ab = ч. ,
7561 Name-pl-ab = Чч. ,
7562 name-pl-ab = чч. ,
7563 case = i ,
7564 Name-sg = Частью ,
7565 name-sg = частью ,

7566 Name-pl = Частями ,
7567 name-pl = частями ,
7568 Name-sg-ab = Ч. ,
7569 name-sg-ab = ч. ,
7570 Name-pl-ab = Чч. ,
7571 name-pl-ab = чч. ,
7572 case = p ,
7573 Name-sg = Части ,
7574 name-sg = части ,
7575 Name-pl = Частях ,
7576 name-pl = частях ,
7577 Name-sg-ab = Ч. ,
7578 name-sg-ab = ч. ,
7579 Name-pl-ab = Чч. ,
7580 name-pl-ab = чч. ,
7581
7582 type = chapter ,
7583 gender = f ,
7584 case = n ,
7585 Name-sg = Глава ,
7586 name-sg = глава ,
7587 Name-pl = Главы ,
7588 name-pl = главы ,
7589 Name-sg-ab = Гл. ,
7590 name-sg-ab = гл. ,
7591 Name-pl-ab = Гл. ,
7592 name-pl-ab = гл. ,
7593 case = a ,
7594 Name-sg = Главу ,
7595 name-sg = главу ,
7596 Name-pl = Главы ,
7597 name-pl = главы ,
7598 Name-sg-ab = Гл. ,
7599 name-sg-ab = гл. ,
7600 Name-pl-ab = Гл. ,
7601 name-pl-ab = гл. ,
7602 case = g ,
7603 Name-sg = Главы ,
7604 name-sg = главы ,
7605 Name-pl = Глав ,
7606 name-pl = глав ,
7607 Name-sg-ab = Гл. ,
7608 name-sg-ab = гл. ,
7609 Name-pl-ab = Гл. ,
7610 name-pl-ab = гл. ,
7611 case = d ,
7612 Name-sg = Главе ,
7613 name-sg = главе ,
7614 Name-pl = Главам ,
7615 name-pl = главам ,
7616 Name-sg-ab = Гл. ,
7617 name-sg-ab = гл. ,
7618 Name-pl-ab = Гл. ,
7619 name-pl-ab = гл. ,

7620 case = i ,
7621 Name-sg = Главой ,
7622 name-sg = главой ,
7623 Name-pl = Главами ,
7624 name-pl = главами ,
7625 Name-sg-ab = Гл. ,
7626 name-sg-ab = гл. ,
7627 Name-pl-ab = Гл. ,
7628 name-pl-ab = гл. ,
7629 case = p ,
7630 Name-sg = Главе ,
7631 name-sg = главе ,
7632 Name-pl = Главах ,
7633 name-pl = главах ,
7634 Name-sg-ab = Гл. ,
7635 name-sg-ab = гл. ,
7636 Name-pl-ab = Гл. ,
7637 name-pl-ab = гл. ,
7638
7639 type = section ,
7640 gender = m ,
7641 case = n ,
7642 Name-sg = Раздел ,
7643 name-sg = раздел ,
7644 Name-pl = Разделы ,
7645 name-pl = разделы ,
7646 case = a ,
7647 Name-sg = Раздел ,
7648 name-sg = раздел ,
7649 Name-pl = Разделы ,
7650 name-pl = разделы ,
7651 case = g ,
7652 Name-sg = Раздела ,
7653 name-sg = раздела ,
7654 Name-pl = Разделов ,
7655 name-pl = разделов ,
7656 case = d ,
7657 Name-sg = Разделу ,
7658 name-sg = разделу ,
7659 Name-pl = Разделам ,
7660 name-pl = разделам ,
7661 case = i ,
7662 Name-sg = Разделом ,
7663 name-sg = разделом ,
7664 Name-pl = Разделами ,
7665 name-pl = разделами ,
7666 case = p ,
7667 Name-sg = Разделе ,
7668 name-sg = разделе ,
7669 Name-pl = Разделах ,
7670 name-pl = разделах ,
7671
7672 type = paragraph ,
7673 gender = m ,

7674 case = n ,
7675 Name-sg = Абзац ,
7676 name-sg = абзац ,
7677 Name-pl = Абзацы ,
7678 name-pl = абзацы ,
7679 case = a ,
7680 Name-sg = Абзац ,
7681 name-sg = абзац ,
7682 Name-pl = Абзацы ,
7683 name-pl = абзацы ,
7684 case = g ,
7685 Name-sg = Абзаца ,
7686 name-sg = абзаца ,
7687 Name-pl = Абзацев ,
7688 name-pl = абзацев ,
7689 case = d ,
7690 Name-sg = Абзацу ,
7691 name-sg = абзацу ,
7692 Name-pl = Абзацам ,
7693 name-pl = абзацам ,
7694 case = i ,
7695 Name-sg = Абзацем ,
7696 name-sg = абзацем ,
7697 Name-pl = Абзацами ,
7698 name-pl = абзацами ,
7699 case = p ,
7700 Name-sg = Абзаце ,
7701 name-sg = абзаце ,
7702 Name-pl = Абзацах ,
7703 name-pl = абзацах ,
7704
7705 type = appendix ,
7706 gender = n ,
7707 case = n ,
7708 Name-sg = Приложение ,
7709 name-sg = приложение ,
7710 Name-pl = Приложения ,
7711 name-pl = приложения ,
7712 case = a ,
7713 Name-sg = Приложение ,
7714 name-sg = приложение ,
7715 Name-pl = Приложения ,
7716 name-pl = приложения ,
7717 case = g ,
7718 Name-sg = Приложения ,
7719 name-sg = приложения ,
7720 Name-pl = Приложений ,
7721 name-pl = приложений ,
7722 case = d ,
7723 Name-sg = Приложению ,
7724 name-sg = приложению ,
7725 Name-pl = Приложениям ,
7726 name-pl = приложениям ,
7727 case = i ,

7728 Name-sg = Приложением ,
7729 name-sg = приложением ,
7730 Name-pl = Приложениями ,
7731 name-pl = приложениями ,
7732 case = p ,
7733 Name-sg = Приложении ,
7734 name-sg = приложении ,
7735 Name-pl = Приложениях ,
7736 name-pl = приложениях ,
7737
7738 type = page ,
7739 gender = f ,
7740 case = n ,
7741 Name-sg = Страница ,
7742 name-sg = страница ,
7743 Name-pl = Страницы ,
7744 name-pl = страницы ,
7745 Name-sg-ab = С. ,
7746 name-sg-ab = с. ,
7747 Name-pl-ab = Сс. ,
7748 name-pl-ab = сс. ,
7749 case = a ,
7750 Name-sg = Страницу ,
7751 name-sg = страницу ,
7752 Name-pl = Страницы ,
7753 name-pl = страницы ,
7754 Name-sg-ab = С. ,
7755 name-sg-ab = с. ,
7756 Name-pl-ab = Сс. ,
7757 name-pl-ab = сс. ,
7758 case = g ,
7759 Name-sg = Страницы ,
7760 name-sg = страницы ,
7761 Name-pl = Страниц ,
7762 name-pl = страниц ,
7763 Name-sg-ab = С. ,
7764 name-sg-ab = с. ,
7765 Name-pl-ab = Сс. ,
7766 name-pl-ab = сс. ,
7767 case = d ,
7768 Name-sg = Странице ,
7769 name-sg = странице ,
7770 Name-pl = Страницам ,
7771 name-pl = страницам ,
7772 Name-sg-ab = С. ,
7773 name-sg-ab = с. ,
7774 Name-pl-ab = Сс. ,
7775 name-pl-ab = сс. ,
7776 case = i ,
7777 Name-sg = Страницей ,
7778 name-sg = страницей ,
7779 Name-pl = Страницами ,
7780 name-pl = страницами ,
7781 Name-sg-ab = С. ,

```

7782     name-sg-ab = с. ,
7783     Name-pl-ab = Сс. ,
7784     name-pl-ab = сс. ,
7785     case = p ,
7786     Name-sg = Странице ,
7787     name-sg = странице ,
7788     Name-pl = Страницах ,
7789     name-pl = страницах ,
7790     Name-sg-ab = С. ,
7791     name-sg-ab = с. ,
7792     Name-pl-ab = Сс. ,
7793     name-pl-ab = сс. ,
7794     rangesep = {\textendash} ,
7795     rangetopair = false ,
7796     +refbounds-rb = {,,} ,
7797
7798 type = line ,
7799     gender = f ,
7800     case = n ,
7801     Name-sg = Строка ,
7802     name-sg = строка ,
7803     Name-pl = Строки ,
7804     name-pl = строки ,
7805     case = a ,
7806     Name-sg = Строку ,
7807     name-sg = строку ,
7808     Name-pl = Строки ,
7809     name-pl = строки ,
7810     case = g ,
7811     Name-sg = Строки ,
7812     name-sg = строки ,
7813     Name-pl = Строк ,
7814     name-pl = строк ,
7815     case = d ,
7816     Name-sg = Строке ,
7817     name-sg = строке ,
7818     Name-pl = Строкам ,
7819     name-pl = строкам ,
7820     case = i ,
7821     Name-sg = Строкой ,
7822     name-sg = строкой ,
7823     Name-pl = Строками ,
7824     name-pl = строками ,
7825     case = p ,
7826     Name-sg = Строке ,
7827     name-sg = строке ,
7828     Name-pl = Строках ,
7829     name-pl = строках ,
7830
7831 type = figure ,
7832     gender = m ,
7833     case = n ,
7834     Name-sg = Рисунок ,
7835     name-sg = рисунок ,

```

7836 Name-pl = Рисунки ,
7837 name-pl = рисунки ,
7838 Name-sg-ab = Рис. ,
7839 name-sg-ab = рис. ,
7840 Name-pl-ab = Рис. ,
7841 name-pl-ab = рис. ,
7842 case = a ,
7843 Name-sg = Рисунок ,
7844 name-sg = рисунок ,
7845 Name-pl = Рисунки ,
7846 name-pl = рисунки ,
7847 Name-sg-ab = Рис. ,
7848 name-sg-ab = рис. ,
7849 Name-pl-ab = Рис. ,
7850 name-pl-ab = рис. ,
7851 case = g ,
7852 Name-sg = Рисунка ,
7853 name-sg = рисунка ,
7854 Name-pl = Рисунков ,
7855 name-pl = рисунков ,
7856 Name-sg-ab = Рис. ,
7857 name-sg-ab = рис. ,
7858 Name-pl-ab = Рис. ,
7859 name-pl-ab = рис. ,
7860 case = d ,
7861 Name-sg = Рисунку ,
7862 name-sg = рисунку ,
7863 Name-pl = Рисункам ,
7864 name-pl = рисункам ,
7865 Name-sg-ab = Рис. ,
7866 name-sg-ab = рис. ,
7867 Name-pl-ab = Рис. ,
7868 name-pl-ab = рис. ,
7869 case = i ,
7870 Name-sg = Рисунком ,
7871 name-sg = рисунком ,
7872 Name-pl = Рисунками ,
7873 name-pl = рисунками ,
7874 Name-sg-ab = Рис. ,
7875 name-sg-ab = рис. ,
7876 Name-pl-ab = Рис. ,
7877 name-pl-ab = рис. ,
7878 case = p ,
7879 Name-sg = Рисунке ,
7880 name-sg = рисунке ,
7881 Name-pl = Рисунках ,
7882 name-pl = рисунках ,
7883 Name-sg-ab = Рис. ,
7884 name-sg-ab = рис. ,
7885 Name-pl-ab = Рис. ,
7886 name-pl-ab = рис. ,
7887
7888 type = table ,
7889 gender = f ,

7890 case = n ,
7891 Name-sg = Таблица ,
7892 name-sg = таблица ,
7893 Name-pl = Таблицы ,
7894 name-pl = таблицы ,
7895 Name-sg-ab = Табл. ,
7896 name-sg-ab = табл. ,
7897 Name-pl-ab = Табл. ,
7898 name-pl-ab = табл. ,
7899 case = a ,
7900 Name-sg = Таблицу ,
7901 name-sg = таблицу ,
7902 Name-pl = Таблицы ,
7903 name-pl = таблицы ,
7904 Name-sg-ab = Табл. ,
7905 name-sg-ab = табл. ,
7906 Name-pl-ab = Табл. ,
7907 name-pl-ab = табл. ,
7908 case = g ,
7909 Name-sg = Таблицы ,
7910 name-sg = таблицы ,
7911 Name-pl = Таблиц ,
7912 name-pl = таблиц ,
7913 Name-sg-ab = Табл. ,
7914 name-sg-ab = табл. ,
7915 Name-pl-ab = Табл. ,
7916 name-pl-ab = табл. ,
7917 case = d ,
7918 Name-sg = Таблице ,
7919 name-sg = таблице ,
7920 Name-pl = Таблицам ,
7921 name-pl = таблицам ,
7922 Name-sg-ab = Табл. ,
7923 name-sg-ab = табл. ,
7924 Name-pl-ab = Табл. ,
7925 name-pl-ab = табл. ,
7926 case = i ,
7927 Name-sg = Таблицей ,
7928 name-sg = таблицей ,
7929 Name-pl = Таблицами ,
7930 name-pl = таблицами ,
7931 Name-sg-ab = Табл. ,
7932 name-sg-ab = табл. ,
7933 Name-pl-ab = Табл. ,
7934 name-pl-ab = табл. ,
7935 case = p ,
7936 Name-sg = Таблице ,
7937 name-sg = таблице ,
7938 Name-pl = Таблицах ,
7939 name-pl = таблицах ,
7940 Name-sg-ab = Табл. ,
7941 name-sg-ab = табл. ,
7942 Name-pl-ab = Табл. ,
7943 name-pl-ab = табл. ,

```

7944
7945 type = item ,
7946   gender = m ,
7947   case = n ,
7948     Name-sg = Пункт ,
7949     name-sg = пункт ,
7950     Name-pl = Пункты ,
7951     name-pl = пункты ,
7952     Name-sg-ab = П. ,
7953     name-sg-ab = п. ,
7954     Name-pl-ab = Пп. ,
7955     name-pl-ab = пп. ,
7956   case = a ,
7957     Name-sg = Пункт ,
7958     name-sg = пункт ,
7959     Name-pl = Пункты ,
7960     name-pl = пункты ,
7961     Name-sg-ab = П. ,
7962     name-sg-ab = п. ,
7963     Name-pl-ab = Пп. ,
7964     name-pl-ab = пп. ,
7965   case = g ,
7966     Name-sg = Пункта ,
7967     name-sg = пункта ,
7968     Name-pl = Пунктов ,
7969     name-pl = пунктов ,
7970     Name-sg-ab = П. ,
7971     name-sg-ab = п. ,
7972     Name-pl-ab = Пп. ,
7973     name-pl-ab = пп. ,
7974   case = d ,
7975     Name-sg = Пункту ,
7976     name-sg = пункту ,
7977     Name-pl = Пунктам ,
7978     name-pl = пунктам ,
7979     Name-sg-ab = П. ,
7980     name-sg-ab = п. ,
7981     Name-pl-ab = Пп. ,
7982     name-pl-ab = пп. ,
7983   case = i ,
7984     Name-sg = Пунктом ,
7985     name-sg = пунктом ,
7986     Name-pl = Пунктами ,
7987     name-pl = пунктами ,
7988     Name-sg-ab = П. ,
7989     name-sg-ab = п. ,
7990     Name-pl-ab = Пп. ,
7991     name-pl-ab = пп. ,
7992   case = p ,
7993     Name-sg = Пункте ,
7994     name-sg = пункте ,
7995     Name-pl = Пунктах ,
7996     name-pl = пунктах ,
7997     Name-sg-ab = П. ,

```



```

7998     name-sg-ab = п. ,
7999     Name-pl-ab = Пп. ,
8000     name-pl-ab = пп. ,
8001
8002 type = footnote ,
8003     gender = f ,
8004     case = n ,
8005     Name-sg = Сноска ,
8006     name-sg = сноска ,
8007     Name-pl = Сноски ,
8008     name-pl = сноски ,
8009     case = a ,
8010     Name-sg = Сноску ,
8011     name-sg = сноску ,
8012     Name-pl = Сноски ,
8013     name-pl = сноски ,
8014     case = g ,
8015     Name-sg = Сноски ,
8016     name-sg = сноски ,
8017     Name-pl = Сносок ,
8018     name-pl = сносок ,
8019     case = d ,
8020     Name-sg = Сноске ,
8021     name-sg = сноске ,
8022     Name-pl = Сноскам ,
8023     name-pl = сноскам ,
8024     case = i ,
8025     Name-sg = Сноской ,
8026     name-sg = сноской ,
8027     Name-pl = Сносками ,
8028     name-pl = сносками ,
8029     case = p ,
8030     Name-sg = Сноске ,
8031     name-sg = сноске ,
8032     Name-pl = Сносках ,
8033     name-pl = сносках ,
8034
8035 type = endnote ,
8036     gender = f ,
8037     case = n ,
8038     Name-sg = Сноска ,
8039     name-sg = сноска ,
8040     Name-pl = Сноски ,
8041     name-pl = сноски ,
8042     case = a ,
8043     Name-sg = Сноску ,
8044     name-sg = сноску ,
8045     Name-pl = Сноски ,
8046     name-pl = сноски ,
8047     case = g ,
8048     Name-sg = Сноски ,
8049     name-sg = сноски ,
8050     Name-pl = Сносок ,
8051     name-pl = сносок ,

```

```

8052 case = d ,
8053     Name-sg = Сноске ,
8054     name-sg = сноске ,
8055     Name-pl = Сноскам ,
8056     name-pl = сноскам ,
8057 case = i ,
8058     Name-sg = Сноской ,
8059     name-sg = сноской ,
8060     Name-pl = Сносками ,
8061     name-pl = сносками ,
8062 case = p ,
8063     Name-sg = Сноске ,
8064     name-sg = сноске ,
8065     Name-pl = Сносках ,
8066     name-pl = сносках ,
8067
8068 type = note ,
8069     gender = f ,
8070     case = n ,
8071     Name-sg = Заметка ,
8072     name-sg = заметка ,
8073     Name-pl = Заметки ,
8074     name-pl = заметки ,
8075     case = a ,
8076     Name-sg = Заметку ,
8077     name-sg = заметку ,
8078     Name-pl = Заметки ,
8079     name-pl = заметки ,
8080     case = g ,
8081     Name-sg = Заметки ,
8082     name-sg = заметки ,
8083     Name-pl = Заметок ,
8084     name-pl = заметок ,
8085     case = d ,
8086     Name-sg = Заметке ,
8087     name-sg = заметке ,
8088     Name-pl = Заметкам ,
8089     name-pl = заметкам ,
8090     case = i ,
8091     Name-sg = Заметкой ,
8092     name-sg = заметкой ,
8093     Name-pl = Заметками ,
8094     name-pl = заметками ,
8095     case = p ,
8096     Name-sg = Заметке ,
8097     name-sg = заметке ,
8098     Name-pl = Заметках ,
8099     name-pl = заметках ,
8100
8101 type = equation ,
8102     gender = n ,
8103     case = n ,
8104     Name-sg = Уравнение ,
8105     name-sg = уравнение ,

```

```

8106     Name-pl = Уравнения ,
8107     name-pl = уравнения ,
8108     Name-sg-ab = Ур. ,
8109     name-sg-ab = ур. ,
8110     Name-pl-ab = Ур. ,
8111     name-pl-ab = ур. ,
8112 case = a ,
8113     Name-sg = Уравнение ,
8114     name-sg = уравнение ,
8115     Name-pl = Уравнения ,
8116     name-pl = уравнения ,
8117     Name-sg-ab = Ур. ,
8118     name-sg-ab = ур. ,
8119     Name-pl-ab = Ур. ,
8120     name-pl-ab = ур. ,
8121 case = g ,
8122     Name-sg = Уравнения ,
8123     name-sg = уравнения ,
8124     Name-pl = Уравнений ,
8125     name-pl = уравнений ,
8126     Name-sg-ab = Ур. ,
8127     name-sg-ab = ур. ,
8128     Name-pl-ab = Ур. ,
8129     name-pl-ab = ур. ,
8130 case = d ,
8131     Name-sg = Уравнению ,
8132     name-sg = уравнению ,
8133     Name-pl = Уравнениям ,
8134     name-pl = уравнениям ,
8135     Name-sg-ab = Ур. ,
8136     name-sg-ab = ур. ,
8137     Name-pl-ab = Ур. ,
8138     name-pl-ab = ур. ,
8139 case = i ,
8140     Name-sg = Уравнением ,
8141     name-sg = уравнением ,
8142     Name-pl = Уравнениями ,
8143     name-pl = уравнениями ,
8144     Name-sg-ab = Ур. ,
8145     name-sg-ab = ур. ,
8146     Name-pl-ab = Ур. ,
8147     name-pl-ab = ур. ,
8148 case = p ,
8149     Name-sg = Уравнении ,
8150     name-sg = уравнении ,
8151     Name-pl = Уравнениях ,
8152     name-pl = уравнениях ,
8153     Name-sg-ab = Ур. ,
8154     name-sg-ab = ур. ,
8155     Name-pl-ab = Ур. ,
8156     name-pl-ab = ур. ,
8157 +refbounds-rb = {c\nobreakspace(,,)} ,
8158 refbounds-first-sg = {(,)} ,
8159 refbounds = {(,,)} ,

```

```

8160
8161 type = theorem ,
8162 gender = f ,
8163 case = n ,
8164   Name-sg = Теорема ,
8165   name-sg = теорема ,
8166   Name-pl = Теоремы ,
8167   name-pl = теоремы ,
8168   Name-sg-ab = Теор. ,
8169   name-sg-ab = теор. ,
8170   Name-pl-ab = Теор. ,
8171   name-pl-ab = теор. ,
8172 case = a ,
8173   Name-sg = Теорему ,
8174   name-sg = теорему ,
8175   Name-pl = Теоремы ,
8176   name-pl = теоремы ,
8177   Name-sg-ab = Теор. ,
8178   name-sg-ab = теор. ,
8179   Name-pl-ab = Теор. ,
8180   name-pl-ab = теор. ,
8181 case = g ,
8182   Name-sg = Теоремы ,
8183   name-sg = теоремы ,
8184   Name-pl = Теорем ,
8185   name-pl = теорем ,
8186   Name-sg-ab = Теор. ,
8187   name-sg-ab = теор. ,
8188   Name-pl-ab = Теор. ,
8189   name-pl-ab = теор. ,
8190 case = d ,
8191   Name-sg = Теореме ,
8192   name-sg = теореме ,
8193   Name-pl = Теоремам ,
8194   name-pl = теоремам ,
8195   Name-sg-ab = Теор. ,
8196   name-sg-ab = теор. ,
8197   Name-pl-ab = Теор. ,
8198   name-pl-ab = теор. ,
8199 case = i ,
8200   Name-sg = Теоремой ,
8201   name-sg = теоремой ,
8202   Name-pl = Теоремами ,
8203   name-pl = теоремами ,
8204   Name-sg-ab = Теор. ,
8205   name-sg-ab = теор. ,
8206   Name-pl-ab = Теор. ,
8207   name-pl-ab = теор. ,
8208 case = p ,
8209   Name-sg = Теореме ,
8210   name-sg = теореме ,
8211   Name-pl = Теоремах ,
8212   name-pl = теоремах ,
8213   Name-sg-ab = Теор. ,

```

8214 name-sg-ab = теор. ,
8215 Name-pl-ab = Теор. ,
8216 name-pl-ab = теор. ,
8217
8218 type = lemma ,
8219 gender = f ,
8220 case = n ,
8221 Name-sg = Лемма ,
8222 name-sg = лемма ,
8223 Name-pl = Леммы ,
8224 name-pl = леммы ,
8225 case = a ,
8226 Name-sg = Лемму ,
8227 name-sg = лемму ,
8228 Name-pl = Леммы ,
8229 name-pl = леммы ,
8230 case = g ,
8231 Name-sg = Леммы ,
8232 name-sg = леммы ,
8233 Name-pl = Лемм ,
8234 name-pl = лемм ,
8235 case = d ,
8236 Name-sg = Лемме ,
8237 name-sg = лемме ,
8238 Name-pl = Леммам ,
8239 name-pl = леммам ,
8240 case = i ,
8241 Name-sg = Леммой ,
8242 name-sg = леммой ,
8243 Name-pl = Леммами ,
8244 name-pl = леммами ,
8245 case = p ,
8246 Name-sg = Лемме ,
8247 name-sg = лемме ,
8248 Name-pl = Леммах ,
8249 name-pl = леммах ,
8250
8251 type = corollary ,
8252 gender = m ,
8253 case = n ,
8254 Name-sg = Вывод ,
8255 name-sg = вывод ,
8256 Name-pl = Выводы ,
8257 name-pl = выводы ,
8258 case = a ,
8259 Name-sg = Вывод ,
8260 name-sg = вывод ,
8261 Name-pl = Выводы ,
8262 name-pl = выводы ,
8263 case = g ,
8264 Name-sg = Вывода ,
8265 name-sg = вывода ,
8266 Name-pl = Выводов ,
8267 name-pl = выводов ,

```

8268 case = d ,
8269     Name-sg = Выводу ,
8270     name-sg = выводу ,
8271     Name-pl = Выводам ,
8272     name-pl = выводам ,
8273 case = i ,
8274     Name-sg = Выводом ,
8275     name-sg = выводом ,
8276     Name-pl = Выводами ,
8277     name-pl = выводами ,
8278 case = p ,
8279     Name-sg = Выводе ,
8280     name-sg = выводе ,
8281     Name-pl = Выводах ,
8282     name-pl = выводах ,
8283
8284 type = proposition ,
8285     gender = n ,
8286     case = n ,
8287         Name-sg = Предложение ,
8288         name-sg = предложение ,
8289         Name-pl = Предложения ,
8290         name-pl = предложения ,
8291         Name-sg-ab = Предл. ,
8292         name-sg-ab = предл. ,
8293         Name-pl-ab = Предл. ,
8294         name-pl-ab = предл. ,
8295     case = a ,
8296         Name-sg = Предложение ,
8297         name-sg = предложение ,
8298         Name-pl = Предложения ,
8299         name-pl = предложения ,
8300         Name-sg-ab = Предл. ,
8301         name-sg-ab = предл. ,
8302         Name-pl-ab = Предл. ,
8303         name-pl-ab = предл. ,
8304     case = g ,
8305         Name-sg = Предложения ,
8306         name-sg = предложения ,
8307         Name-pl = Предложений ,
8308         name-pl = предложений ,
8309         Name-sg-ab = Предл. ,
8310         name-sg-ab = предл. ,
8311         Name-pl-ab = Предл. ,
8312         name-pl-ab = предл. ,
8313     case = d ,
8314         Name-sg = Предложению ,
8315         name-sg = предложению ,
8316         Name-pl = Предложениям ,
8317         name-pl = предложениям ,
8318         Name-sg-ab = Предл. ,
8319         name-sg-ab = предл. ,
8320         Name-pl-ab = Предл. ,
8321         name-pl-ab = предл. ,

```

8322 case = i ,
8323 Name-sg = Предложением ,
8324 name-sg = предложением ,
8325 Name-pl = Предложениями ,
8326 name-pl = предложениями ,
8327 Name-sg-ab = Предл. ,
8328 name-sg-ab = предл. ,
8329 Name-pl-ab = Предл. ,
8330 name-pl-ab = предл. ,
8331 case = p ,
8332 Name-sg = Предложении ,
8333 name-sg = предложении ,
8334 Name-pl = Предложениях ,
8335 name-pl = предложениях ,
8336 Name-sg-ab = Предл. ,
8337 name-sg-ab = предл. ,
8338 Name-pl-ab = Предл. ,
8339 name-pl-ab = предл. ,
8340
8341 type = definition ,
8342 gender = n ,
8343 case = n ,
8344 Name-sg = Определение ,
8345 name-sg = определение ,
8346 Name-pl = Определения ,
8347 name-pl = определения ,
8348 Name-sg-ab = Опр. ,
8349 name-sg-ab = опр. ,
8350 Name-pl-ab = Опр. ,
8351 name-pl-ab = опр. ,
8352 case = a ,
8353 Name-sg = Определение ,
8354 name-sg = определение ,
8355 Name-pl = Определения ,
8356 name-pl = определения ,
8357 Name-sg-ab = Опр. ,
8358 name-sg-ab = опр. ,
8359 Name-pl-ab = Опр. ,
8360 name-pl-ab = опр. ,
8361 case = g ,
8362 Name-sg = Определения ,
8363 name-sg = определения ,
8364 Name-pl = Определений ,
8365 name-pl = определений ,
8366 Name-sg-ab = Опр. ,
8367 name-sg-ab = опр. ,
8368 Name-pl-ab = Опр. ,
8369 name-pl-ab = опр. ,
8370 case = d ,
8371 Name-sg = Определению ,
8372 name-sg = определению ,
8373 Name-pl = Определениям ,
8374 name-pl = определениям ,
8375 Name-sg-ab = Опр. ,

8376 name-sg-ab = опр. ,
8377 Name-pl-ab = Опр. ,
8378 name-pl-ab = опр. ,
8379 case = i ,
8380 Name-sg = Определением ,
8381 name-sg = определением ,
8382 Name-pl = Определениями ,
8383 name-pl = определениями ,
8384 Name-sg-ab = Опр. ,
8385 name-sg-ab = опр. ,
8386 Name-pl-ab = Опр. ,
8387 name-pl-ab = опр. ,
8388 case = p ,
8389 Name-sg = Определении ,
8390 name-sg = определении ,
8391 Name-pl = Определениях ,
8392 name-pl = определениях ,
8393 Name-sg-ab = Опр. ,
8394 name-sg-ab = опр. ,
8395 Name-pl-ab = Опр. ,
8396 name-pl-ab = опр. ,
8397
8398 type = proof ,
8399 gender = n ,
8400 case = n ,
8401 Name-sg = Доказательство ,
8402 name-sg = доказательство ,
8403 Name-pl = Доказательства ,
8404 name-pl = доказательства ,
8405 case = a ,
8406 Name-sg = Доказательство ,
8407 name-sg = доказательство ,
8408 Name-pl = Доказательства ,
8409 name-pl = доказательства ,
8410 case = g ,
8411 Name-sg = Доказательства ,
8412 name-sg = доказательства ,
8413 Name-pl = Доказательств ,
8414 name-pl = доказательств ,
8415 case = d ,
8416 Name-sg = Доказательству ,
8417 name-sg = доказательству ,
8418 Name-pl = Доказательствам ,
8419 name-pl = доказательствам ,
8420 case = i ,
8421 Name-sg = Доказательством ,
8422 name-sg = доказательством ,
8423 Name-pl = Доказательствами ,
8424 name-pl = доказательствами ,
8425 case = p ,
8426 Name-sg = Доказательстве ,
8427 name-sg = доказательстве ,
8428 Name-pl = Доказательствах ,
8429 name-pl = доказательствах ,

8430
8431 type = result ,
8432 gender = m ,
8433 case = n ,
8434 Name-sg = Результат ,
8435 name-sg = результат ,
8436 Name-pl = Результаты ,
8437 name-pl = результаты ,
8438 case = a ,
8439 Name-sg = Результат ,
8440 name-sg = результат ,
8441 Name-pl = Результаты ,
8442 name-pl = результаты ,
8443 case = g ,
8444 Name-sg = Результата ,
8445 name-sg = результата ,
8446 Name-pl = Результатов ,
8447 name-pl = результатов ,
8448 case = d ,
8449 Name-sg = Результату ,
8450 name-sg = результату ,
8451 Name-pl = Результатам ,
8452 name-pl = результатам ,
8453 case = i ,
8454 Name-sg = Результатом ,
8455 name-sg = результатом ,
8456 Name-pl = Результатами ,
8457 name-pl = результатами ,
8458 case = p ,
8459 Name-sg = Результате ,
8460 name-sg = результате ,
8461 Name-pl = Результатах ,
8462 name-pl = результатах ,
8463
8464 type = remark ,
8465 gender = n ,
8466 case = n ,
8467 Name-sg = Примечание ,
8468 name-sg = примечание ,
8469 Name-pl = Примечания ,
8470 name-pl = примечания ,
8471 Name-sg-ab = Прим. ,
8472 name-sg-ab = прим. ,
8473 Name-pl-ab = Прим. ,
8474 name-pl-ab = прим. ,
8475 case = a ,
8476 Name-sg = Примечание ,
8477 name-sg = примечание ,
8478 Name-pl = Примечания ,
8479 name-pl = примечания ,
8480 Name-sg-ab = Прим. ,
8481 name-sg-ab = прим. ,
8482 Name-pl-ab = Прим. ,
8483 name-pl-ab = прим. ,

```

8484 case = g ,
8485     Name-sg = Примечания ,
8486     name-sg = примечания ,
8487     Name-pl = Примечаний ,
8488     name-pl = примечаний ,
8489     Name-sg-ab = Прим. ,
8490     name-sg-ab = прим. ,
8491     Name-pl-ab = Прим. ,
8492     name-pl-ab = прим. ,
8493 case = d ,
8494     Name-sg = Примечанию ,
8495     name-sg = примечанию ,
8496     Name-pl = Примечаниям ,
8497     name-pl = примечаниям ,
8498     Name-sg-ab = Прим. ,
8499     name-sg-ab = прим. ,
8500     Name-pl-ab = Прим. ,
8501     name-pl-ab = прим. ,
8502 case = i ,
8503     Name-sg = Примечанием ,
8504     name-sg = примечанием ,
8505     Name-pl = Примечаниями ,
8506     name-pl = примечаниями ,
8507     Name-sg-ab = Прим. ,
8508     name-sg-ab = прим. ,
8509     Name-pl-ab = Прим. ,
8510     name-pl-ab = прим. ,
8511 case = p ,
8512     Name-sg = Примечании ,
8513     name-sg = примечании ,
8514     Name-pl = Примечаниях ,
8515     name-pl = примечаниях ,
8516     Name-sg-ab = Прим. ,
8517     name-sg-ab = прим. ,
8518     Name-pl-ab = Прим. ,
8519     name-pl-ab = прим. ,
8520
8521 type = example ,
8522     gender = m ,
8523     case = n ,
8524     Name-sg = Пример ,
8525     name-sg = пример ,
8526     Name-pl = Примеры ,
8527     name-pl = примеры ,
8528     case = a ,
8529     Name-sg = Пример ,
8530     name-sg = пример ,
8531     Name-pl = Примеры ,
8532     name-pl = примеры ,
8533     case = g ,
8534     Name-sg = Примера ,
8535     name-sg = примера ,
8536     Name-pl = Примеров ,
8537     name-pl = примеров ,

```

```

8538 case = d ,
8539     Name-sg = Примеру ,
8540     name-sg = примеру ,
8541     Name-pl = Примерам ,
8542     name-pl = примерам ,
8543 case = i ,
8544     Name-sg = Примером ,
8545     name-sg = примером ,
8546     Name-pl = Примерами ,
8547     name-pl = примерами ,
8548 case = p ,
8549     Name-sg = Примере ,
8550     name-sg = примере ,
8551     Name-pl = Примерах ,
8552     name-pl = примерах ,
8553
8554 type = algorithm ,
8555     gender = m ,
8556     case = n ,
8557     Name-sg = Алгоритм ,
8558     name-sg = алгоритм ,
8559     Name-pl = Алгоритмы ,
8560     name-pl = алгоритмы ,
8561     case = a ,
8562     Name-sg = Алгоритм ,
8563     name-sg = алгоритм ,
8564     Name-pl = Алгоритмы ,
8565     name-pl = алгоритмы ,
8566     case = g ,
8567     Name-sg = Алгоритма ,
8568     name-sg = алгоритма ,
8569     Name-pl = Алгоритмов ,
8570     name-pl = алгоритмов ,
8571     case = d ,
8572     Name-sg = Алгоритму ,
8573     name-sg = алгоритму ,
8574     Name-pl = Алгоритмам ,
8575     name-pl = алгоритмам ,
8576     case = i ,
8577     Name-sg = Алгоритмом ,
8578     name-sg = алгоритмом ,
8579     Name-pl = Алгоритмами ,
8580     name-pl = алгоритмами ,
8581     case = p ,
8582     Name-sg = Алгоритме ,
8583     name-sg = алгоритме ,
8584     Name-pl = Алгоритмах ,
8585     name-pl = алгоритмах ,
8586
8587 type = listing ,
8588     gender = m ,
8589     case = n ,
8590     Name-sg = Листинг ,
8591     name-sg = листинг ,

```

8592 Name-pl = Листинги ,
8593 name-pl = листинги ,
8594 case = a ,
8595 Name-sg = Листинг ,
8596 name-sg = листинг ,
8597 Name-pl = Листинги ,
8598 name-pl = листинги ,
8599 case = g ,
8600 Name-sg = Листинга ,
8601 name-sg = листинга ,
8602 Name-pl = Листингов ,
8603 name-pl = листингов ,
8604 case = d ,
8605 Name-sg = Листингу ,
8606 name-sg = листингу ,
8607 Name-pl = Листингам ,
8608 name-pl = листингам ,
8609 case = i ,
8610 Name-sg = Листингом ,
8611 name-sg = листингм ,
8612 Name-pl = Листингами ,
8613 name-pl = листингами ,
8614 case = p ,
8615 Name-sg = Листинге ,
8616 name-sg = листинге ,
8617 Name-pl = Листингах ,
8618 name-pl = листингах ,
8619
8620 type = exercise ,
8621 gender = n ,
8622 case = n ,
8623 Name-sg = Упражнение ,
8624 name-sg = упражнение ,
8625 Name-pl = Упражнения ,
8626 name-pl = упражнения ,
8627 Name-sg-ab = Упр. ,
8628 name-sg-ab = упр. ,
8629 Name-pl-ab = Упр. ,
8630 name-pl-ab = упр. ,
8631 case = a ,
8632 Name-sg = Упражнение ,
8633 name-sg = упражнение ,
8634 Name-pl = Упражнения ,
8635 name-pl = упражнения ,
8636 Name-sg-ab = Упр. ,
8637 name-sg-ab = упр. ,
8638 Name-pl-ab = Упр. ,
8639 name-pl-ab = упр. ,
8640 case = g ,
8641 Name-sg = Упражнения ,
8642 name-sg = упражнения ,
8643 Name-pl = Упражнений ,
8644 name-pl = упражнений ,
8645 Name-sg-ab = Упр. ,

8646 name-sg-ab = упр. ,
8647 Name-pl-ab = Упр. ,
8648 name-pl-ab = упр. ,
8649 case = d ,
8650 Name-sg = Упражнению ,
8651 name-sg = упражнению ,
8652 Name-pl = Упражнениям ,
8653 name-pl = упражнениям ,
8654 Name-sg-ab = Упр. ,
8655 name-sg-ab = упр. ,
8656 Name-pl-ab = Упр. ,
8657 name-pl-ab = упр. ,
8658 case = i ,
8659 Name-sg = Упражнением ,
8660 name-sg = упражнением ,
8661 Name-pl = Упражнениями ,
8662 name-pl = упражнениями ,
8663 Name-sg-ab = Упр. ,
8664 name-sg-ab = упр. ,
8665 Name-pl-ab = Упр. ,
8666 name-pl-ab = упр. ,
8667 case = p ,
8668 Name-sg = Упражнении ,
8669 name-sg = упражнении ,
8670 Name-pl = Упражнениях ,
8671 name-pl = упражнениях ,
8672 Name-sg-ab = Упр. ,
8673 name-sg-ab = упр. ,
8674 Name-pl-ab = Упр. ,
8675 name-pl-ab = упр. ,
8676
8677 type = solution ,
8678 gender = n ,
8679 case = n ,
8680 Name-sg = Решение ,
8681 name-sg = решение ,
8682 Name-pl = Решения ,
8683 name-pl = решения ,
8684 case = a ,
8685 Name-sg = Решение ,
8686 name-sg = решение ,
8687 Name-pl = Решения ,
8688 name-pl = решения ,
8689 case = g ,
8690 Name-sg = Решения ,
8691 name-sg = решения ,
8692 Name-pl = Решений ,
8693 name-pl = решений ,
8694 case = d ,
8695 Name-sg = Решению ,
8696 name-sg = решению ,
8697 Name-pl = Решениям ,
8698 name-pl = решениям ,
8699 case = i ,

8700 Name-sg = Решением ,
8701 name-sg = решением ,
8702 Name-pl = Решениями ,
8703 name-pl = решениями ,
8704 case = p ,
8705 Name-sg = Решении ,
8706 name-sg = решении ,
8707 Name-pl = Решениях ,
8708 name-pl = решениях ,
8709 `</lang-russian>`

Index

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

A	
<code>\A</code>	1906
<code>\AddToHook</code>	<u>120</u> , 2061, 2105, 2128, 2159, 2161, 2199, 2272, 2314, 2468, 2481, 2489, 5396, 5417, 5433, 5457, 5472, 5497, 5515, 5549, 5585, 5610
<code>\AddToHookWithArguments</code>	2443
<code>\alph</code>	5530, 5601
<code>\appendix</code>	<u>2</u> , <u>128</u> , <u>129</u> , <u>131</u> , <u>139</u>
<code>\appendixname</code>	<u>128</u> , <u>140</u>
<code>\AtEndOfPackage</code>	<u>2479</u>
B	
<code>\babelname</code>	2115
<code>\babelprovide</code>	<u>30</u> , <u>56</u>
bool commands:	
<code>\bool_gset_false:N</code>	559
<code>\bool_gset_true:N</code>	552, 2455
<code>\bool_if:NTF</code>	403, 480, 518, 576, 1779, 1830, 2065, 2069, 2445, 2491, 3471, 3874, 4015, 4142, 4178, 4212, 4279, 4292, 4304, 4355, 4372, 4382, 4387, 4433, 4437, 4493, 4518, 4525, 4531, 4542, 4548, 4576, 4621, 4643, 4672, 4821, 4989, 4991, 5552, 5613
<code>\bool_if:nTF</code>	86, 3585, 3595, 3619, 3636, 3651, 3716, 3724, 4365, 4742, 4783, 4864, 4881, 4932
<code>\bool_if_exist:NTF</code>	357, 367, 391, 401, 451, 468, 478, 503, 506, 514, 516, 530, 533, 540, 543, 550, 557
<code>\bool_lazy_all:nTF</code>	4461, 5001
<code>\bool_lazy_and:nnTF</code> ..	49, 54, 2615, 3442, 3463, 4246, 4317, 4695, 4971, 5016
<code>\bool_lazy_any:nTF</code>	5154, 5163
<code>\bool_lazy_or:nnTF</code>	1396, 1429, 1986, 2561, 2828, 3330, 3355, 3446, 4958, 5571
<code>\bool_new:N</code> .	138, 358, 368, 393, 452, 470, 508, 531, 534, 541, 544, 551, 558, 818, 1593, 1594, 1621, 1645, 2009, 2016, 2023, 2036, 2037, 2207, 2208, 2209, 2210, 2211, 2307, 2308, 2436, 2448, 3479, 3494, 3756, 3757, 3768, 3769, 3776, 3778, 3779, 3792, 3793, 3794, 3806, 3807, 5514, 5559, 5584
<code>\bool_set:Nn</code>	3439
<code>\bool_set_eq:NN</code>	566
<code>\bool_set_false:N</code> ..	392, 469, 505, 507, 542, 1606, 1610, 1782, 1880, 1928, 1973, 2044, 2053, 2054, 2071, 2078, 2219, 2223, 2230, 2238, 2239, 2240, 2332, 2344, 3484, 3577, 3823, 3824, 3841, 3880, 3891, 4291, 4483, 4484, 4491, 4492, 4725, 4939, 5161, 5178
<code>\bool_set_true:N</code>	359, 369, 453, 532, 535, 545, 1600, 1601, 1605, 1611, 1804, 1826, 1892, 1894, 1932, 1934, 1946, 1948, 1977, 1979, 1998, 2000, 2043, 2048, 2049, 2217, 2224, 2229, 2246, 2248, 2250, 2253, 2254, 2255, 2320, 2325, 3591, 3601, 3605, 3627, 3642, 3657, 3681, 3849, 3875, 3881, 3885, 3892, 4038, 4049, 4060, 4110, 4146, 4182, 4216, 4233, 4314, 4348, 4520, 4580, 4626, 4648, 4677, 4940, 4946, 4953, 5079, 5138, 5177, 5215, 5222, 5223, 5241, 5258, 5260, 5528, 5564, 5598
<code>\bool_until_do:Nn</code>	1881, 1935, 1980, 3617, 3842
<code>\bool_while_do:nn</code>	5643

C	
<code>\chaptername</code>	140
clist commands:	
<code>\clist_map_inline:nn</code> ...	651, 699, 716, 1035, 2241, 2457, 2967, 5531, 5603
<code>\counterwithin</code>	5, 62, 63
<code>\counterwithout</code>	63
<code>\crefstripprefix</code>	46
cs commands:	
<code>\cs_generate_variant:Nn</code>	82, 83, 300, 306, 313, 324, 335, 346, 361, 371, 378, 385, 396, 420, 430, 455, 462, 473, 511, 537, 547, 554, 561, 989, 1863, 1902, 1956, 2008, 2623, 4777, 4815, 5198, 5302, 5335, 5368
<code>\cs_if_exist:NTF</code> 24, 27, 66, 75, 96, 5372, 5384, 5436
<code>\cs_if_exist_p:N</code> 51, 56, 4248, 4319, 4697, 5645
<code>\cs_if_exist_use:N</code>	5496, 5683
<code>\cs_new:Npn</code>	64, 73, 84, 94, 105, 301, 307, 309, 311, 314, 325, 336, 348, 350, 738, 4738, 4778, 4816, 5188
<code>\cs_new_protected:Npn</code>	295, 352, 362, 372, 379, 386, 411, 421, 442, 444, 446, 456, 463, 501, 528, 538, 548, 555, 831, 933, 1542, 1561, 1765, 1864, 1912, 1957, 2487, 2621, 3434, 3498, 3540, 3551, 3564, 3694, 3746, 3808, 4022, 4487, 4734, 4736, 4930, 5182, 5199, 5270, 5303, 5336, 5565
<code>\cs_set_eq:NN</code>	740
<code>\cs_to_str:N</code>	351
D	
<code>\d</code>	1906
<code>\def</code>	3
E	
<code>\eqref</code>	134
exp commands:	
<code>\exp_args:Ne</code>	943
<code>\exp_args:Nee</code> 1783, 1795, 1807, 1817, 1883, 1937, 1982, 5201, 5207, 5229, 5233, 5248
<code>\exp_args:NNe</code>	36, 39
<code>\exp_args:NNNo</code>	297
<code>\exp_args:NNNV</code> .	1844, 1896, 1950, 2002
<code>\exp_args:NNo</code>	297, 303
<code>\exp_args:No</code>	303
<code>\exp_not:N</code> 114, 4389, 4402, 4405, 4408, 4754, 4758, 4766, 4770, 4795, 4798, 4806, 4809, 4837, 4840, 4844, 4849, 4855, 4858, 4870, 4873, 4901, 4906, 4916, 4921
<code>\exp_not:n</code>	304, 4054, 4077, 4085, 4103, 4116, 4120, 4155, 4164, 4186, 4194, 4201, 4225, 4239, 4256, 4266, 4308, 4331, 4341, 4390, 4401, 4406, 4407, 4564, 4587, 4599, 4633, 4655, 4664, 4684, 4705, 4715, 4755, 4767, 4796, 4797, 4807, 4808, 4838, 4839, 4841, 4845, 4856, 4857, 4859, 4867, 4871, 4872, 4875, 4902, 4917
<code>\ExplSyntaxOn</code>	31, 949
F	
<code>\figurename</code>	140
file commands:	
<code>\file_get:nnNTF</code>	943
<code>\fmtversion</code>	5
<code>\footnote</code>	2
G	
group commands:	
<code>\group_begin:</code>	754, 935, 1781, 1868, 1916, 1961, 2891, 3436, 3450, 3482, 4389, 4405, 4754, 4766, 4795, 4806, 4837, 4844, 4855, 4870, 4901, 4916
<code>\group_end:</code>	766, 987, 1845, 1897, 1951, 2003, 2919, 3453, 3476, 3486, 4402, 4408, 4758, 4770, 4798, 4809, 4840, 4849, 4858, 4873, 4906, 4921
H	
<code>\hyperlink</code>	5185
I	
<code>\IfBooleanT</code>	3483
<code>\IfClassLoadedTF</code>	133
<code>\ifdraft</code>	2222
<code>\IfFormatAtLeastTF</code>	5, 6
<code>\ifoptionfinal</code>	2228
<code>\IfPackageAtLeastTF</code>	2318
<code>\IfPackageLoadedTF</code>	131
<code>\input</code>	30
int commands:	
<code>\int_case:nnTF</code> 4025, 4067, 4129, 4440, 4555, 4612
<code>\int_compare:nNnTF</code> 3628, 3643, 3658, 3670, 3682, 3702, 3704, 3748, 3922, 4045, 4073, 4095, 4150, 4220, 4425, 4427, 4504, 4534, 4593, 5211, 5217, 5237, 5243, 5661
<code>\int_compare_p:nNn</code> ...	1399, 1432, 1987, 1989, 2564, 2831, 3333, 3358, 3718, 3726, 4465, 4962, 4974, 5006, 5174
<code>\int_gincr:N</code>	124

<code>\int_incr:N</code> . 4480, 4512, 4524, 4526, 4541, 4543, 4547, 4549, 4561, 4584, 4596, 4630, 4652, 4661, 4681, 4732, 5659	186, 191, 196, 201, 207, 212, 215, 218, 223, 227, 234, 239, 244, 251, 260, 265, 270, 274, 276, 278, 280, 287
<code>\int_new:N</code> 118, 139, 3495, 3496, 3758, 3759, 3760, 3773, 3774	<code>\msg_new:nnn</code> 140, 146, 151, 153, 155, 161, 167, 173, 179, 184, 189, 194, 199, 204, 209, 214, 216, 221, 226, 228, 230, 232, 237, 242, 248, 250, 252, 257, 263, 268, 273, 275, 277, 279, 281, 283, 285, 290
<code>\int_rand:n</code> 322, 333, 344	<code>\msg_warning:nn</code> 2070, 2076, 2348, 2618, 4467
<code>\int_set:Nn</code> 3703, 3705, 3709, 3712, 5642	<code>\msg_warning:nnn</code> 758, 779, 1574, 1581, 1737, 1743, 2148, 2185, 2197, 2259, 2270, 2312, 2337, 2412, 2474, 2484, 2637, 2751, 2757, 2918, 2962, 3008, 3200, 3206, 3287, 3906, 4286, 4995, 5011
<code>\int_to_roman:n</code> 5646, 5653, 5654, 5657	<code>\msg_warning:nnnn</code> 847, 864, 898, 916, 2101, 2288, 2297, 2366, 2522, 2528, 2534, 2540, 2573, 2786, 2792, 2798, 2804, 2843, 2935, 2942, 2972, 3255, 3261, 3267, 3273, 3346, 3371, 3914, 5080, 5140
<code>\int_use:N</code> 52, 57, 62, 77, 128	<code>\msg_warning:nnnnn</code> 884, 923, 2956, 5030 <code>\msg_warning:nnnnnn</code> 5037
<code>\int_zero:N</code> 3696, 3697, 3818, 3819, 3820, 3821, 3822, 4478, 4479, 4481, 4482, 4727, 4728	N
iow commands:	<code>\NeedsTeXFormat</code> 4
<code>\iow_char:N</code> 143, 158, 159, 164, 165, 170, 171, 176, 177, 229, 246, 293, 2289, 2298	<code>\newcounter</code> 5, 5371, 5435
<code>\iow_newline:</code> 287	<code>\NewDocumentCommand</code> 752, 769, 2619, 2624, 2889, 3432, 3480
K	<code>\newfloat</code> 132
keys commands:	<code>\NewHook</code> 1654
<code>\l_keys_choice_tl</code> 823	<code>\newsubfloat</code> 132
<code>\keys_define:nn</code> 21, 659, 705, 722, 783, 990, 1079, 1104, 1132, 1341, 1380, 1458, 1568, 1595, 1622, 1631, 1646, 1655, 2010, 2017, 2024, 2030, 2038, 2073, 2082, 2096, 2124, 2163, 2194, 2201, 2213, 2274, 2281, 2283, 2292, 2302, 2309, 2321, 2333, 2345, 2353, 2360, 2389, 2406, 2430, 2437, 2450, 2470, 2499, 2517, 2547, 2584, 2607, 2633, 2645, 2669, 2781, 2811, 2854, 2922, 2993, 3017, 3044, 3250, 3280, 3320, 3382	<code>\newtheorem</code> 139
<code>\keys_set:nn</code> 28, 31, 58, 59, 85, 763, 911, 974, 2326, 2622, 2627, 2916, 3437	<code>\nobreakspace</code> . . 1555, 5719, 5720, 5722, 5723, 5725, 5727, 5924, 5925, 5927, 5928, 5930, 5932, 6371, 6372, 6374, 6375, 6377, 6379, 6588, 6589, 6591, 6592, 6594, 6596, 6814, 6815, 6817, 6818, 6820, 6822, 7028, 7029, 7031, 7032, 7034, 7036, 7246, 7247, 7249, 7250, 7252, 7254, 7255, 7380, 7481, 7482, 7484, 7485, 7487, 7489, 7490, 8157
keyval commands:	<code>\noeqref</code> 5574
<code>\keyval_parse:nnn</code> . . . 1547, 2364, 2410	<code>\NumCheckSetup</code> 46
L	<code>\NumsCheckSetup</code> 46
<code>\label</code> 2, 61, 64, 135	P
<code>\labelformat</code> 3	<code>\PackageError</code> 9
<code>\languagename</code> 25, 141, 2109	<code>\pagename</code> 140
M	<code>\pagenote</code> 132
<code>\mainbabelname</code> 25, 2116	<code>\pagenumbering</code> 7
msg commands:	<code>\pageref</code> 86
<code>\msg_info:nnn</code> 977, 1030, 1095, 1288, 1294, 1348, 5474, 5507, 5556, 5577, 5617, 5635, 5662	
<code>\msg_info:nnnn</code> 1003, 1010, 1040, 1412, 1446	
<code>\msg_info:nnnnn</code> 1024	
<code>\msg_line_context:</code> 142, 148, 152, 154, 157, 163, 169, 175, 181,	

<code>\str_case:nnTF</code>	1137, 1659, 2130, 2167, 2243, 2673, 3049
<code>\str_compare:nNnTF</code>	3608
<code>\str_if_eq:nnTF</code>	107, 4780
<code>\str_if_eq_p:nn</code>	5159, 5165, 5167, 5171, 5572, 5573
<code>\str_new:N</code>	2081
<code>\str_set:Nn</code>	2086, 2088, 2090, 2092
<code>\subparagraph</code>	139
<code>\subref</code>	138
<code>\subsection</code>	139
<code>\subsubsection</code>	62
<code>\subsubsections</code>	139
<code>\subsubsubsection</code>	62
T	
<code>\tablename</code>	140
<code>\tag</code>	125, 133, 135
T _E X and L ^A T _E X _{2ϵ} commands:	
<code>\@Alph</code>	128
<code>\@addtoreset</code>	5
<code>\@bsphack</code>	936
<code>\@capttype</code>	5496, 5683
<code>\@chapapp</code>	128
<code>\@currentcounter</code>	2–5, 64, 133, 136, 27, 28, 55, 56, 57, 2433, 2434
<code>\@currentlabel</code>	3, 125, 136
<code>\@elt</code>	5
<code>\@esphack</code>	986
<code>\@ifl@t@r</code>	5
<code>\@onlypreamble</code>	768, 782, 2921
<code>\bbl@loaded</code>	56
<code>\bbl@main@language</code>	25, 2110
<code>\c@lstnumber</code>	136
<code>\c@page</code>	7
<code>\caption@subtyphook</code>	5684
<code>\hyper@@link</code>	114, 123
<code>\hyper@linkfile</code>	5186
<code>\lst@AddToHook</code>	5633
<code>\ltx@gobble</code>	64
<code>\MT@newlabel</code>	134
<code>\zref@addprop</code>	63, 21, 31, 46, 61, 63, 115, 129
<code>\zref@default</code>	114, 4735, 4737
<code>\zref@extractdefault</code>	12, 124, 298, 304, 308
<code>\zref@ifpropundefined</code>	43, 1292, 1579, 1741, 2755, 3204, 5190, 5600
<code>\zref@ifrefcontainsprop</code>	43, 46, 1769, 1777, 1836, 1854, 1866, 1914, 1959, 3909, 4740, 4823, 4877, 5193
<code>\zref@ifrefundefined</code>	3508, 3510, 3522, 3877, 3879, 3884, 3901, 4283, 4294, 4495, 4818, 4943
<code>\zref@label</code>	64, 2446
<code>\zref@localaddprop</code>	5498, 5553, 5614, 5685
<code>\ZREF@mainlist</code>	21, 31, 46, 61, 63, 115, 129, 5498, 5553, 5614, 5685
<code>\zref@newprop</code> ..	6, 8, 20, 22, 32, 47, 62, 110, 116, 128, 5495, 5530, 5601, 5682
<code>\zref@refused</code>	3899
<code>\zref@wrapper@babel</code> ..	64, 85, 2446, 3433
<code>\zrefclever@required@kernel</code> ..	3, 4, 6, 11
<code>\textendash</code>	1559, 5774, 6038, 6650, 6872, 7086, 7309, 7794
<code>\thechapter</code>	128, 130
<code>\thelstnumber</code>	136
<code>\thepage</code>	7, 8, 122, 125
<code>\thesection</code>	128
tl commands:	
<code>\c_novaluel_tl</code> ..	707, 708, 709, 710, 711, 712, 713, 2501, 2549, 2647, 2813
<code>\tl_clear:N</code>	366, 390, 851, 889, 902, 919, 928, 953, 962, 995, 2628, 2894, 2904, 2927, 3812, 3813, 3814, 3815, 3816, 3817, 3848, 4473, 4474, 4475, 4476, 4477, 4523, 4540, 4938, 4945, 4952, 4983, 5078, 5137, 5296
<code>\tl_const:Nn</code>	1544
<code>\tl_gclear:N</code>	383, 427
<code>\tl_gset:Nn</code>	376, 417, 761, 776
<code>\tl_gset_eq:NN</code>	125
<code>\tl_head:N</code>	3646, 3659, 3671, 3673, 3683, 3685
<code>\tl_head:n</code>	1884, 1885, 1938, 1939, 1983, 1984, 1990
<code>\tl_if_empty:NTF</code>	34, 98, 845, 855, 882, 893, 914, 921, 1028, 1084, 1109, 1141, 1176, 1213, 1250, 1298, 1346, 1352, 1385, 1463, 1501, 1767, 1891, 1945, 2960, 2998, 3022, 3053, 3088, 3125, 3162, 3210, 3285, 3291, 3325, 3387, 3408, 3454, 4281, 4874, 4968, 4993, 5051, 5062, 5105
<code>\tl_if_empty:nTF</code>	755, 771, 994, 1286, 1563, 1572, 1735, 2454, 2749, 2926, 3198, 5184
<code>\tl_if_empty_p:N</code> ..	50, 55, 2617, 4247, 4318, 4696, 5004, 5018, 5158, 5168, 5172
<code>\tl_if_empty_p:n</code>	1397, 1430, 2562, 2829, 3331, 3356, 3587, 3588, 3597, 3598, 3623, 3624, 3639, 3654
<code>\tl_if_eq:NNTF</code> ..	122, 3558, 3581, 3888
<code>\tl_if_eq:NnTF</code>	1793, 3501, 3533, 3708, 3711, 3736, 3739, 3856, 3904, 4949, 5205

<code>\tl_if_eq:nnTF</code>	<code>\zcDeclareLanguageAlias</code>
1783, 1795, 1807, 1817, 1883, 1937, 1982, 3700, 5201, 5207, 5229, 5233, 5248 26, 769, 5710, 5711, 5712, 5713, 5714, 5715, 5716, 5917, 5918, 5919, 5920, 5921, 6368, 6583, 6584, 6585
<code>\tl_if_exist:NTF</code>	<code>\zcLanguageSetup</code>
354, 364, 374, 381, 388, 399, 415, 425, 744 21, 30, 31, 69, 74, 75, 140, 2889
<code>\tl_if_exist_p:N</code>	<code>\zcpageref</code>
2616	86, 3480
<code>\tl_if_novalue:nTF</code>	<code>\zcref</code>
..... 2504, 2552, 2650, 2816	21, 66, 68, 85–88, 94, 96, 130, 134, 3432, 3485
<code>\tl_map_break:n</code>	<code>\zcRefTypeSetup</code>
108	21, 69, 2624
<code>\tl_map_tokens:Nn</code>	<code>\zcsetup</code>
100	21, 55, 66, 68, 69, 2619
<code>\tl_new:N</code>	<code>\zlabel</code>
119, 134, 135, 355, 365, 375, 382, 416, 426, 581, 582, 583, 733, 734, 735, 736, 760, 775, 1567, 2193, 2212, 2280, 2301, 2352, 2429, 3488, 3489, 3490, 3491, 3492, 3493, 3761, 3762, 3763, 3764, 3765, 3766, 3767, 3770, 3771, 3775, 3777, 3780, 3781, 3782, 3783, 3784, 3785, 3786, 3787, 3788, 3789, 3790, 3791	2, 64, 131, 133, 135
<code>\tl_put_left:Nn</code>	zrefcheck commands:
4368, 4375, 4418, 5064, 5065, 5107, 5109, 5111, 5113	<code>\zrefcheck_zcref_beg_label:</code> ... 3445
<code>\tl_put_right:Nn</code>	<code>\zrefcheck_zcref_end_label_-</code> maybe:
4052, 4075, 4083, 4101, 4114, 4153, 4162, 4184, 4192, 4199, 4223, 4237, 4254, 4264, 4562, 4585, 4597, 4631, 4653, 4662, 4682, 4703, 4713, 4969, 4970, 4981, 5684	3467
<code>\tl_reverse:N</code>	<code>\zrefcheck_zcref_run_checks_on_-</code> labels:n
3568, 3571	3468
<code>\tl_set:Nn</code>	zrefclever commands:
297, 356, 737, 762, 952, 996, 1008, 1576, 1583, 1585, 1774, 1842, 1846, 1859, 1870, 1875, 1887, 1889, 1898, 1900, 1918, 1923, 1941, 1943, 1952, 1954, 1963, 1968, 1993, 1995, 2004, 2006, 2109, 2110, 2115, 2116, 2119, 2120, 2134, 2140, 2145, 2171, 2177, 2182, 2626, 2895, 2928, 2940, 3675, 3677, 3858, 3859, 4032, 4034, 4306, 4329, 4339, 4385, 4508, 4510, 4521, 4538, 4964, 4965, 4979, 5569	<code>\zrefclever_language_if_declared:n</code> 750
<code>\tl_set_eq:NN</code>	<code>\zrefclever_language_if_declared:nTF</code> 750
435, 4471	<code>\zrefclever_language_varname:n</code> 740, 740
<code>\tl_show:N</code>	<code>\l_zrefclever_ref_language_tl</code> ..
4434	736
<code>\tl_tail:N</code>	zrefclever internal commands:
3676, 3678	<code>\l__zrefclever_abbrev_bool</code> 3780, 3967, 4972
<code>\tl_tail:n</code>	<code>\l__zrefclever_amsmath_subequations_-</code> bool
... 1888, 1890, 1942, 1944, 1994, 1996	5514, 5528, 5552
<code>\tl_use:N</code>	<code>\l__zrefclever_breqn_dgroup_bool</code> 5584, 5598, 5613
319, 330, 341, 777, 941, 946, 976, 978, 982	<code>\l__zrefclever_cap_bool</code>
	3780, 3963, 4959
	<code>\l__zrefclever_capfirst_bool</code> 2016, 2019, 4961
	<code>__zrefclever_compat_module:nn</code> . . 66, 2487, 2487, 5369, 5413, 5477, 5510, 5560, 5580, 5620, 5638, 5665, 5688
	<code>__zrefclever_counter_reset_by:n</code> 6, 62, 63, 66, 68, 70, 75, 77, 79, 84, 84, 5378, 5390, 5442, 5452, 5522, 5592
	<code>__zrefclever_counter_reset_by_-</code> aux:nn
	91, 94
	<code>__zrefclever_counter_reset_by_-</code> auxi:nnn
	101, 105
	<code>\l__zrefclever_counter_resetby_-</code> prop
	6, 63, 87, 88, 2405, 2417
	<code>\l__zrefclever_counter_resettters_-</code> seq
	5, 6, 62, 63, 90, 2388, 2392
	<code>\l__zrefclever_counter_type_prop</code> 4, 61, 36, 39, 2359, 2371
<code>\Z</code>	
1906	
<code>\zcDeclareLanguage</code>	
13, 26, 752, 5709, 5914, 6367, 6582, 6811, 7025, 7243, 7476	

U

use commands:

<code>\use:N</code>	25, 28, 823, 4250, 4325, 4699, 5362
<code>\UseHook</code>	1869, 1917, 1962

Z

<code>\Z</code>	1906
<code>\zcDeclareLanguage</code>	13, 26, 752, 5709, 5914, 6367, 6582, 6811, 7025, 7243, 7476

\l_zrefclever_current_counter_-
 tl 3, 5, 64, 20, 24, 25, 37,
 40, 42, 50, 51, 52, 113, 117, 2429, 2432
\l_zrefclever_current_language_-
 tl 25,
 55, 734, 2109, 2115, 2119, 2135, 2172
\l_zrefclever_endrangefunc_tl .
 3780, 3955, 4247, 4248,
 4250, 4318, 4319, 4325, 4696, 4697, 4699
\l_zrefclever_endrangeprop_tl .
 48, 1767, 1777, 3780, 3959
__zrefclever_extract:nnn
 12, 307, 307, 1872,
 1877, 1920, 1925, 1965, 1970, 3629,
 3631, 3644, 3661, 3749, 3751, 5212,
 5214, 5218, 5220, 5238, 5240, 5244, 5246
__zrefclever_extract_default:Nnnn
 12, 295, 295, 300, 1771,
 1832, 1839, 1849, 1856, 3542, 3553,
 3555, 3566, 3569, 3572, 3574, 3862, 3865
__zrefclever_extract_unexp:nnn
 12, 124, 301, 301, 306, 1785,
 1789, 1797, 1801, 1809, 1813, 1819,
 1823, 4397, 4751, 4756, 4768, 4792,
 4833, 4846, 4895, 4903, 4918, 5191,
 5194, 5195, 5202, 5203, 5208, 5209,
 5230, 5231, 5234, 5235, 5250, 5254, 5570
__zrefclever_extract_url_-
 unexp:n 4393,
 4750, 4791, 4829, 4891, 5188, 5188, 5198
__zrefclever_get_enclosing_-
 counters:n 6, 64, 64, 69, 82, 117
__zrefclever_get_enclosing_-
 counters_value:n 6, 64, 73, 78, 83, 112
__zrefclever_get_endrange_-
 pagecomp:nnN 1912, 1956
__zrefclever_get_endrange_-
 pagecomptwo:nnN 1957, 2008
__zrefclever_get_endrange_-
 property:nnN 45, 1765, 1863
__zrefclever_get_endrange_-
 stripprefix:nnN 1864, 1902
__zrefclever_get_ref:nN
 114, 115, 4055, 4078, 4086,
 4104, 4117, 4121, 4156, 4165, 4187,
 4195, 4202, 4226, 4240, 4267, 4309,
 4342, 4377, 4565, 4588, 4600, 4634,
 4656, 4665, 4685, 4716, 4738, 4738, 4777
__zrefclever_get_ref_endrange:nnN
 46,
 115, 4257, 4332, 4706, 4778, 4778, 4815
__zrefclever_get_ref_first:
 114, 115, 119, 4369, 4419, 4816, 4816
__zrefclever_get_rf_opt_bool:nN 127
__zrefclever_get_rf_opt_-
 bool:nnnnN 21,
 3960, 3964, 3968, 5336, 5336, 5368
__zrefclever_get_rf_opt_-
 seq:nnnN 21, 127, 3972,
 3976, 3980, 3984, 3988, 3992, 3996,
 4000, 4004, 4008, 5020, 5303, 5303, 5335
__zrefclever_get_rf_opt_tl:nnnN
 21, 23, 46, 126, 3456, 3827, 3831,
 3835, 3924, 3928, 3932, 3936, 3940,
 3944, 3948, 3952, 3956, 5270, 5270, 5302
__zrefclever_hyperlink:nnn . 124,
 4391, 4749, 4790, 4827, 4889, 5182, 5182
\l__zrefclever_hyperlink_bool
 2036, 2043, 2048, 2053, 2065,
 2071, 2078, 3484, 4744, 4785, 4883, 5156
\l_zrefclever_hyperref_warn_-
 bool 2037, 2044, 2049, 2054, 2069
__zrefclever_if_class_loaded:n
 130, 132
__zrefclever_if_class_loaded:nTF
 5479
__zrefclever_if_package_-
 loaded:n 130, 130
__zrefclever_if_package_-
 loaded:nTF
 2063, 2107, 2113, 2316, 5415,
 5512, 5562, 5582, 5622, 5640, 5667, 5690
__zrefclever_is_integer_rgx:n
 1903, 1904, 1911
__zrefclever_is_integer_rgx:nTF
 1929, 1931, 1974, 1976
\l__zrefclever_label_a_tl
 93, 3761, 3845, 3864, 3877,
 3899, 3901, 3907, 3910, 3916, 4033,
 4055, 4078, 4086, 4121, 4187, 4202,
 4252, 4258, 4267, 4299, 4309, 4327,
 4333, 4342, 4495, 4499, 4509, 4522,
 4539, 4565, 4601, 4665, 4701, 4707, 4716
\l_zrefclever_label_b_tl 93,
 3761, 3848, 3853, 3867, 3879, 3884, 4499
\l_zrefclever_label_count_int
 94, 3758,
 3818, 3922, 4025, 4478, 4504, 4732, 5007
\l__zrefclever_label_enclval_a_-
 tl 3488, 3566, 3568,
 3623, 3639, 3659, 3671, 3675, 3676, 3683
\l__zrefclever_label_enclval_b_-
 tl 3488, 3569, 3571,
 3624, 3646, 3654, 3673, 3677, 3678, 3685
\l__zrefclever_label_extdoc_a_tl
 3488, 3572, 3582, 3587, 3597, 3610
\l__zrefclever_label_extdoc_b_tl
 3488, 3574, 3583, 3588, 3598, 3609

<code>\l_zrefclever_label_type_a_tl</code> .	<code>\l_zrefclever_main_language_tl</code>
..... 3457, 3488, 3543,	25, 55, 735, 2110, 2116, 2120, 2141, 2178
3545, 3548, 3554, 3559, 3708, 3736,	<code>\l_zrefclever_mathtools_loaded_-</code>
3828, 3832, 3836, 3858, 3863, 3889,	bool 3471, 5559, 5564
3904, 3925, 3929, 3933, 3937, 3941,	<code>_zrefclever_mathtools_showonlyrefs:n</code>
3945, 3949, 3953, 3957, 3961, 3965, 3473, 5565
3969, 3973, 3977, 3981, 3985, 3989,	<code>_zrefclever_name_default:</code>
3993, 3997, 4001, 4005, 4009, 4035, 4511 4734, 4736, 4866
<code>\l_zrefclever_label_type_b_tl</code> .	<code>\l_zrefclever_name_format_-</code>
..... 3488,	fallback_tl 3767,
3556, 3560, 3711, 3739, 3859, 3866, 3890	4979, 4983, 5051, 5100, 5112, 5114, 5132
<code>_zrefclever_label_type_put_-</code>	<code>\l_zrefclever_name_format_tl . .</code>
new_right:n 87, 89, 3504, 3540, 3540 3767, 4964, 4965, 4969,
<code>\l_zrefclever_label_types_seq</code> .	4970, 4980, 4981, 5057, 5064, 5065,
..... 88, 3497, 3500, 3544, 3547, 3734	5073, 5081, 5091, 5108, 5109, 5122, 5142
<code>\l_zrefclever_labelhook_bool</code> ..	<code>\l_zrefclever_name_in_link_bool</code>
..... 2436, 2439, 2445 116, 119,
<code>_zrefclever_labels_in_sequence:n</code>	3767, 4387, 4821, 4939, 5161, 5177, 5178
... 48, 95, 124, 4297, 4498, 5199, 5199	<code>\l_zrefclever_namefont_tl</code>
<code>\l_zrefclever_lang_decl_case_tl</code>	3780, 3947, 4390, 4406, 4838, 4856, 4871
.. 581, 962, 965, 1008, 1013, 1352,	<code>\l_zrefclever_nameinlink_str</code> ..
1369, 2904, 2907, 2940, 2945, 3291, 3308 2081, 2086,
<code>\l_zrefclever_lang_declension_-</code>	2088, 2090, 2092, 5159, 5165, 5167, 5171
seq 581,	<code>\l_zrefclever_namesep_tl</code>
841, 842, 843, 857, 861, 868, 959,	... 3780, 3927, 4841, 4859, 4867, 4875
960, 961, 964, 1001, 1007, 1012,	<code>\l_zrefclever_next_is_same_bool</code>
2901, 2902, 2903, 2906, 2933, 2939, 2944 94, 124,
<code>\l_zrefclever_lang_gender_seq</code> .	3773, 4492, 4525, 4542, 4548, 5223, 5261
..... 581, 878, 879, 880, 895, 972,	<code>\l_zrefclever_next_maybe_range_-</code>
973, 1022, 1037, 2914, 2915, 2954, 2969	bool .. 94, 124, 3773, 4291, 4304,
<code>_zrefclever_language_if_-</code>	4491, 4518, 4531, 5215, 5222, 5241, 5259
declared:n 26, 742, 749, 751	<code>\l_zrefclever_noabbrev_first_-</code>
<code>_zrefclever_language_if_-</code>	bool 2023, 2026, 4976
declared:n(TF) 25	<code>\g_zrefclever_nocompat_bool</code>
<code>_zrefclever_language_if_-</code> 2448, 2455, 2491
declared:nTF ... 316, 327, 338,	<code>\l_zrefclever_nocompat_bool</code> 65
742, 757, 773, 833, 937, 2146, 2183, 2892	<code>\g_zrefclever_nocompat_modules_-</code>
<code>_zrefclever_language_varname:n</code>	seq 2449, 2459, 2462, 2483, 2492, 2493
..... 25, 319, 330,	<code>\l_zrefclever_nocompat_modules_-</code>
341, 738, 738, 741, 744, 760, 761,	seq 65
775, 776, 777, 941, 946, 976, 978, 982	<code>\l_zrefclever_nudge_comptosing_-</code>
<code>\l_zrefclever_last_of_type_bool</code>	bool .. 2209, 2239, 2248, 2254, 5003
..... 94, 3755,	<code>\l_zrefclever_nudge_enabled_-</code>
3875, 3880, 3881, 3885, 3891, 3892, 4015	bool 2207, 2217,
<code>\l_zrefclever_lastsep_tl</code>	2219, 2223, 2224, 2229, 2230, 4463, 4989
3780, 3939, 4085, 4120, 4164, 4201, 4239	<code>\l_zrefclever_nudge_gender_bool</code>
<code>\l_zrefclever_link_star_bool</code> 2211, 2240, 2250, 2255, 5017
... 3439, 3478, 4745, 4786, 4884, 5157	<code>\l_zrefclever_nudge_multitype_-</code>
<code>\l_zrefclever_listsep_tl</code>	bool .. 2208, 2238, 2246, 2253, 4464
..... 3780, 3935, 4116, 4194,	<code>\l_zrefclever_nudge_singular_-</code>
4564, 4587, 4599, 4633, 4655, 4664, 4684	bool 2210, 2266, 4991
<code>\g_zrefclever_loaded_langfiles_-</code>	<code>_zrefclever_opt_bool_get:NN</code> ..
seq 932, 940, 975, 981 562, 572

```

\__zrefclever_opt_bool_get:NN(TF)
..... 20
\__zrefclever_opt_bool_get:NNTF
..... 562, 5339, 5344, 5349, 5354, 5359
\__zrefclever_opt_bool_gset_-
false:N ..... 19,
528, 555, 561, 1510, 1527, 3410, 3418
\__zrefclever_opt_bool_gset_-
true:N ..... 19,
528, 548, 554, 1472, 1489, 3389, 3397
\__zrefclever_opt_bool_if:N . 573, 580
\__zrefclever_opt_bool_if:N(TF) . 20
\__zrefclever_opt_bool_if:NNTF ..
..... 573, 906
\__zrefclever_opt_bool_if_set:N
..... 512, 527
\__zrefclever_opt_bool_if_-
set:N(TF) ..... 19
\__zrefclever_opt_bool_if_-
set:NNTF .....
. 512, 564, 575, 1465, 1481, 1503, 1519
\__zrefclever_opt_bool_set_-
false:N 19, 528, 538, 547, 2594, 2868
\__zrefclever_opt_bool_set_-
true:N . 19, 528, 528, 537, 2589, 2859
\__zrefclever_opt_bool_unset:N .
..... 18, 501, 501, 511, 2599, 2877
\__zrefclever_opt_seq_get:NN 490, 500
\__zrefclever_opt_seq_get:NN(TF) 18
\__zrefclever_opt_seq_get:NNTF .
..... 490, 836, 873, 954, 967,
2896, 2909, 5306, 5311, 5316, 5321, 5326
\__zrefclever_opt_seq_gset_-
clist_split:Nn .....
.. 17, 442, 444, 1394, 1427, 3328, 3353
\__zrefclever_opt_seq_gset_eq:NN
..... 17, 442,
456, 462, 1403, 1436, 2977, 3337, 3362
\__zrefclever_opt_seq_if_set:N .
..... 474, 489
\__zrefclever_opt_seq_if_-
set:N(TF) ..... 18
\__zrefclever_opt_seq_if_set:NNTF
..... 474, 492, 1045, 1387, 1419
\__zrefclever_opt_seq_set_clist_-
split:Nn ... 17, 442, 442, 2559, 2826
\__zrefclever_opt_seq_set_eq:NN
..... 17, 442, 446, 455, 2568, 2835
\__zrefclever_opt_seq_unset:N ..
..... 17, 463, 463, 473, 2554, 2818
\__zrefclever_opt_tl_clear:N ...
15, 352, 362, 371, 1663, 1668, 1683,
1698, 1713, 2677, 2682, 2697, 2712, 2727
\__zrefclever_opt_tl_cset_-
fallback:nm ..... 1542, 1549
\__zrefclever_opt_tl_gclear:N ..
15, 352, 379, 385, 3055, 3061, 3069,
3076, 3097, 3113, 3134, 3150, 3171, 3187
\__zrefclever_opt_tl_gclear_if_-
new:N ..... 16,
411, 421, 430, 1143, 1149, 1157,
1164, 1185, 1201, 1222, 1238, 1259, 1275
\__zrefclever_opt_tl_get:NN . 431, 441
\__zrefclever_opt_tl_get:NN(TF) . 17
\__zrefclever_opt_tl_get:NNTF ..
..... 431, 5053, 5068, 5087, 5096,
5117, 5127, 5273, 5278, 5283, 5288, 5293
\__zrefclever_opt_tl_gset:N ..... 15
\__zrefclever_opt_tl_gset:Nn ...
..... 352, 372, 378, 3000, 3024,
3032, 3090, 3105, 3127, 3142, 3164,
3179, 3212, 3219, 3228, 3236, 3293, 3303
\__zrefclever_opt_tl_gset_if_-
new:Nn ..... 16,
411, 411, 420, 1086, 1111, 1120,
1178, 1193, 1215, 1230, 1252, 1267,
1300, 1307, 1316, 1324, 1354, 1364
\__zrefclever_opt_tl_if_set:N .. 397
\__zrefclever_opt_tl_if_set:N(TF)
..... 16
\__zrefclever_opt_tl_if_set:NNTF
..... 397, 413, 423, 433
\__zrefclever_opt_tl_set:N ..... 15
\__zrefclever_opt_tl_set:Nn ...
..... 352, 352,
361, 1677, 1692, 1707, 1747, 1753,
2510, 2659, 2691, 2706, 2721, 2761, 2768
\__zrefclever_opt_tl_unset:N ...
..... 15, 386, 386,
396, 1722, 1727, 2506, 2652, 2736, 2741
\__zrefclever_opt_var_set_bool:n
..... 14, 15, 350, 350, 357,
358, 359, 367, 368, 369, 391, 392,
393, 401, 403, 451, 452, 453, 468,
469, 470, 478, 480, 506, 507, 508,
516, 518, 533, 534, 535, 543, 544, 545
\__zrefclever_opt_varname_-
fallback:nm .....
.. 14, 348, 348, 1545, 5294, 5327, 5360
\__zrefclever_opt_varname_-
general:nm ..... 13,
309, 309, 1665, 1670, 1679, 1685,
1694, 1700, 1709, 1715, 1724, 1729,
1749, 1755, 2507, 2511, 2555, 2569,
2590, 2595, 2600, 5274, 5307, 5340
\__zrefclever_opt_varname_lang_-
default:nm ..... 14, 325, 325,

```

335, 1088, 1113, 1145, 1151, 1180,
1187, 1217, 1224, 1254, 1261, 1302,
1309, 1389, 1405, 1467, 1474, 1505,
1512, 3002, 3026, 3057, 3063, 3092,
3099, 3129, 3136, 3166, 3173, 3214,
3221, 3339, 3391, 3412, 5289, 5322, 5355
__zrefclever_opt_varname_lang_-
 type:nnnn 14,
336, 336, 347, 1047, 1056, 1064,
1122, 1159, 1166, 1195, 1203, 1232,
1240, 1269, 1277, 1318, 1326, 1356,
1366, 1421, 1438, 1483, 1491, 1521,
1529, 2979, 3034, 3071, 3078, 3107,
3115, 3144, 3152, 3181, 3189, 3230,
3238, 3295, 3305, 3364, 3399, 3420,
5070, 5119, 5129, 5284, 5317, 5350
__zrefclever_opt_varname_-
 language:nnn 13, 314,
314, 324, 789, 794, 804, 809, 820,
825, 838, 875, 908, 956, 969, 2898, 2911
__zrefclever_opt_varname_-
 type:nnn 13, 311, 311, 313,
2654, 2661, 2679, 2684, 2693, 2699,
2708, 2714, 2723, 2729, 2738, 2743,
2763, 2770, 2820, 2837, 2861, 2870,
2879, 5055, 5089, 5098, 5279, 5312, 5345
\g__zrefclever_page_format_int .
 118, 124, 128
\l__zrefclever_pairsep_tl
 3780, 3931,
4054, 4077, 4103, 4155, 4186, 4225, 4308
\g__zrefclever_prev_page_format_-
 tl 8, 119, 122, 125
__zrefclever_process_language_-
 settings: ... 58, 59, 831, 831, 3441
__zrefclever_prop_put_non_-
 empty:Nnn 43, 1561, 1561, 2370, 2416
__zrefclever_provide_langfile:n
 21, 31, 32, 85, 933, 933, 989, 2152, 3440
\l__zrefclever_range_beg_is_-
 first_bool
 3773, 3823, 4142, 4178, 4212,
4483, 4520, 4576, 4621, 4643, 4672, 4725
\l__zrefclever_range_beg_label_-
 tl 94, 3773, 3816, 4105, 4118, 4157,
4166, 4196, 4227, 4241, 4251, 4476,
4521, 4538, 4589, 4635, 4657, 4686, 4700
\l__zrefclever_range_count_int .
 . 94, 3773, 3821, 4067, 4131, 4481,
4524, 4535, 4541, 4547, 4555, 4614, 4727
\l__zrefclever_range_end_ref_tl
 3773, 3817, 4253, 4259,
4328, 4334, 4477, 4523, 4540, 4702, 4708
\l__zrefclever_range_same_count_-
 int 94,
3773, 3822, 4045, 4096, 4132, 4482,
4526, 4543, 4549, 4594, 4615, 4728
\l__zrefclever_rangeseq_tl 3780,
3943, 4256, 4266, 4331, 4341, 4705, 4715
\l__zrefclever_rangetopair_bool
 3780, 3971, 4292
\l__zrefclever_ref_count_int 3758,
3820, 4073, 4151, 4221, 4479, 4512,
4561, 4584, 4596, 4630, 4652, 4661, 4681
\l__zrefclever_ref_decl_case_tl
 28, 845, 850, 851, 855,
858, 862, 866, 869, 914, 917, 919,
2193, 2203, 5062, 5066, 5105, 5110, 5115
__zrefclever_ref_default:
 4734, 4734, 4775, 4781, 4819, 4860, 4926
\l__zrefclever_ref_gender_tl ...
 29, 882, 888,
889, 893, 896, 901, 902, 921, 927,
928, 2212, 2276, 5018, 5026, 5032, 5040
\l__zrefclever_ref_language_tl .
 25, 28, 55, 733,
737, 834, 839, 849, 867, 876, 886,
900, 909, 918, 925, 2134, 2140, 2145,
2153, 2171, 2177, 2182, 3440, 3458,
3829, 3833, 3837, 3926, 3930, 3934,
3938, 3942, 3946, 3950, 3954, 3958,
3962, 3966, 3970, 3974, 3978, 3982,
3986, 3990, 3994, 3998, 4002, 4006,
4010, 5022, 5034, 5045, 5071, 5120, 5130
\l__zrefclever_ref_property_tl .
 43, 48,
1567, 1576, 1583, 1585, 1769, 1793,
1837, 1854, 1866, 1873, 1878, 1914,
1921, 1926, 1959, 1966, 1971, 3533,
3856, 3911, 3915, 4740, 4825, 4879, 5205
\l__zrefclever_ref_propserity_tl 3501
\l__zrefclever_ref_typeset_font_-
 tl 2280, 2282, 3451
\l__zrefclever_refbounds_first_-
 pb_seq 3795,
3983, 4059, 4109, 4181, 4232, 4313
\l__zrefclever_refbounds_first_-
 rb_seq . 3795, 3987, 4215, 4347, 4676
\l__zrefclever_refbounds_first_-
 seq 3795, 3975, 4358, 4579, 4625, 4647
\l__zrefclever_refbounds_first_-
 sg_seq . 3795, 3979, 4037, 4048, 4145
\l__zrefclever_refbounds_last_-
 pe_seq 3795,
4007, 4056, 4079, 4106, 4158, 4188, 4310
\l__zrefclever_refbounds_last_-
 re_seq

...	3795, 4011, 4260, 4268, 4335, 4343	1423, 1440, 1463, 1485, 1493, 1501,
\l__zrefclever_refbounds_last_-		1523, 1531, 2626, 2628, 2655, 2662,
seq	3795, 4003, 4087, 4122, 4167, 4203	2680, 2685, 2694, 2700, 2709, 2715,
\l__zrefclever_refbounds_mid_rb_-		2724, 2730, 2739, 2744, 2764, 2771,
seq	... 3795, 3995, 4228, 4242, 4687	2821, 2838, 2862, 2871, 2880, 2894,
\l__zrefclever_refbounds_mid_re_-		2927, 2928, 2960, 2981, 2998, 3022,
seq 3795, 3999, 4709, 4717	3036, 3053, 3073, 3080, 3088, 3109,
\l__zrefclever_refbounds_mid_seq		3117, 3125, 3146, 3154, 3162, 3183,
.....	3795, 3991, 4119,	3191, 3210, 3232, 3240, 3285, 3297,
	4197, 4566, 4590, 4602, 4636, 4658, 4666	3307, 3325, 3366, 3387, 3401, 3408, 3422
\l__zrefclever_reffont_tl	\l__zrefclever_sort_decided_bool
.....	3780, 3951, 3494, 3577, 3591,
	4755, 4767, 4796, 4807, 4845, 4902, 4917	3601, 3605, 3617, 3627, 3642, 3657, 3681
\l__zrefclever_reftype_override_-		__zrefclever_sort_default:nn
tl 34, 44, 2352, 2355 89, 3535, 3551, 3551
\g__zrefclever_rf_opts_bool_-		__zrefclever_sort_default_-
maybe_type_specific_seq	different_types:nn
.....	53, 586, 1456, 2582, 2852, 3380 45, 87, 92, 3562, 3694, 3694
\g__zrefclever_rf_opts_seq_-		__zrefclever_sort_default_same_-
refbounds_seq	type:nn
.....	586, 1378, 2545, 2809, 3318 87, 89, 3561, 3564, 3564
\g__zrefclever_rf_opts_tl_maybe_-		__zrefclever_sort_labels:
type_specific_seq	.. 586, 1102, 3015 87-89, 93, 3449, 3498, 3498
\g__zrefclever_rf_opts_tl_not_-		__zrefclever_sort_page:nn
type_specific_seq 93, 3534, 3746, 3746
.....	586, 1077, 2631, 2991	\l__zrefclever_sort_prior_a_int
\g__zrefclever_rf_opts_tl_-		3495, 3696, 3702, 3703, 3709, 3719, 3727
reference_seq 586, 2497	\l__zrefclever_sort_prior_b_int
\g__zrefclever_rf_opts_tl_type_-		3495, 3697, 3704, 3705, 3712, 3720, 3728
names_seq 586, 1339, 3278	\l__zrefclever_tlastsep_tl
\g__zrefclever_rf_opts_tl_-	 3780, 3838, 4457
typesetup_seq 586, 2643	\l__zrefclever_tlistsep_tl
\l__zrefclever_setup_language_tl	 3780, 3834, 4428
.....	581, 762, 790, 795,	\l__zrefclever_tmpa_bool
	805, 810, 821, 826, 952, 1004, 1011,	134, 1782,
	1025, 1042, 1048, 1057, 1065, 1089,	1804, 1826, 1830, 1880, 1881, 1892,
	1114, 1123, 1146, 1152, 1160, 1167,	1894, 1928, 1932, 1934, 1935, 1946,
	1181, 1188, 1196, 1204, 1218, 1225,	1948, 1973, 1977, 1979, 1980, 1998, 2000
	1233, 1241, 1255, 1262, 1270, 1278,	\l__zrefclever_tmpa_int
	1303, 1310, 1319, 1327, 1357, 1367, 134,
	1390, 1406, 1422, 1439, 1468, 1475,	5642, 5646, 5653, 5654, 5657, 5659, 5661
	1484, 1492, 1506, 1513, 1522, 1530,	\g__zrefclever_tmpa_seq
	2895, 2936, 2943, 2957, 2974, 2980,	134, 1393,
	3003, 3027, 3035, 3058, 3064, 3072,	1395, 1400, 1409, 1414, 1426, 1428,
	3079, 3093, 3100, 3108, 3116, 3130,	1433, 1443, 1448, 3327, 3329, 3334,
	3137, 3145, 3153, 3167, 3174, 3182,	3343, 3348, 3352, 3354, 3359, 3368, 3373
	3190, 3215, 3222, 3231, 3239, 3296,	\l__zrefclever_tmpa_seq
	3306, 3340, 3365, 3392, 3400, 3413, 3421
\l__zrefclever_setup_type_tl 134, 1034, 1038, 1070,
...	581, 953, 995, 996, 1028, 1049,	2558, 2560, 2565, 2570, 2575, 2825,
	1058, 1066, 1084, 1109, 1124, 1141,	2827, 2832, 2840, 2845, 2966, 2970, 2985
	1161, 1168, 1176, 1197, 1205, 1213,	\l__zrefclever_tmpa_tl
	1234, 1242, 1250, 1271, 1279, 1298, 134, 950, 974,
	1320, 1328, 1346, 1358, 1368, 1385,	1870, 1884, 1887, 1888, 1918, 1929,
		1938, 1941, 1942, 1963, 1974, 1983,
		1993, 1994, 3459, 3460, 5569, 5572, 5573
		\l__zrefclever_tmpb_tl
	 134,
		1832, 1839, 1842, 1846, 1875, 1885,
		1889, 1890, 1891, 1898, 1923, 1931,

1939, 1943, 1944, 1945, 1952, 1968,
 1976, 1984, 1987, 1990, 1995, 1996, 2004
 \l_zrefclever_tpairsep_tl
 3780, 3830, 4450
 \l_zrefclever_type_count_int
 94, 119, 3758, 3819, 4425,
 4427, 4440, 4465, 4480, 4962, 4975, 5174
 \l_zrefclever_type_first_label_-
 tl 94, 116, 3761, 3814, 4032, 4283,
 4294, 4298, 4326, 4377, 4394, 4398,
 4474, 4508, 4818, 4824, 4830, 4834,
 4847, 4878, 4892, 4896, 4904, 4919, 4943
 \l_zrefclever_type_first_label_-
 type_tl 94, 119, 3761,
 3815, 4034, 4287, 4475, 4510, 4950,
 4997, 5013, 5021, 5033, 5039, 5056,
 5072, 5082, 5090, 5099, 5121, 5131, 5143
 \l_zrefclever_type_first_-
 refbounds_seq
 3795, 4036, 4047, 4058,
 4108, 4144, 4180, 4214, 4231, 4312,
 4346, 4357, 4378, 4578, 4624, 4646,
 4675, 4842, 4843, 4850, 4852, 4888,
 4900, 4908, 4911, 4914, 4915, 4922, 4923
 \l_zrefclever_type_first_-
 refbounds_set_bool
 3795, 3824, 4038, 4049, 4060,
 4111, 4147, 4183, 4217, 4234, 4314,
 4348, 4355, 4484, 4581, 4627, 4649, 4678
 \l_zrefclever_type_name_gender_-
 seq 3767, 5023, 5025, 5028, 5043
 \l_zrefclever_type_name_-
 missing_bool 119,
 3767, 4864, 4940, 4946, 4953, 5079, 5139
 \l_zrefclever_type_name_setup:
 21, 23, 116, 4364, 4930, 4930
 \l_zrefclever_type_name_tl
 116, 119,
 3767, 4401, 4407, 4839, 4857, 4872,
 4874, 4938, 4945, 4952, 5060, 5076,
 5078, 5094, 5103, 5125, 5135, 5137, 5158
 \l_zrefclever_typeset_compress_-
 bool 1645, 1648, 4493
 \l_zrefclever_typeset_labels_-
 seq 93, 3755, 3810, 3844, 3846, 3852
 \l_zrefclever_typeset_last_bool
 94,
 3755, 3841, 3842, 3849, 3874, 4437, 5173
 \l_zrefclever_typeset_name_bool
 1594, 1601, 1606, 1611, 4366, 4382, 4933
 \l_zrefclever_typeset_queue_-
 curr_tl 94, 96, 114, 119, 3761,
 3813, 4052, 4075, 4083, 4101, 4114,
 4153, 4162, 4184, 4192, 4199, 4223,
 4237, 4254, 4264, 4281, 4306, 4329,
 4339, 4368, 4375, 4385, 4418, 4434,
 4445, 4451, 4458, 4472, 4473, 4562,
 4585, 4597, 4631, 4653, 4662, 4682,
 4703, 4713, 4968, 4993, 5004, 5168, 5172
 \l_zrefclever_typeset_queue_-
 prev_tl 94, 3761, 3812, 4429, 4471
 \l_zrefclever_typeset_range_-
 bool 1779, 2009, 2012, 3448, 4279
 \l_zrefclever_typeset_ref_bool
 1593, 1600, 1605, 1610, 4366, 4372, 4933
 \l_zrefclever_typeset_refs:
 93, 95, 96, 3452, 3808, 3808
 \l_zrefclever_typeset_refs_last_-
 of_type:
 100, 114, 116, 119, 4017, 4022, 4022
 \l_zrefclever_typeset_refs_not_-
 last_of_type:
 95, 101, 114, 124, 4019, 4487, 4487
 \l_zrefclever_typeset_sort_bool
 1621, 1624, 3447
 \l_zrefclever_typesort_seq
 45, 92, 1630, 1635, 1636, 1642, 3698
 \l_zrefclever_verbose_testing_-
 bool 3807, 4433
 \l_zrefclever_zcref:nnn
 28, 57, 3433, 3434
 \l_zrefclever_zcref:nnnn 85, 87, 3434
 \l_zrefclever_zcref_labels_seq
 87, 88,
 3438, 3469, 3474, 3478, 3503, 3506, 3811
 \l_zrefclever_zcref_note_tl
 2301, 2304, 3454, 3461
 \l_zrefclever_zcref_with_check_-
 bool 2308, 2325, 3444, 3465
 \l_zrefclever_zcsetup:n
 69, 2620, 2621, 2621,
 2623, 5374, 5386, 5399, 5420, 5438,
 5448, 5460, 5481, 5499, 5517, 5551,
 5587, 5612, 5624, 5634, 5649, 5669, 5692
 \l_zrefclever_zrefcheck_-
 available_bool
 2307, 2320, 2332, 2344, 3443, 3464