

The zref-clever package

Code documentation

gusbrs

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

Version v0.4.6 – 2024-08-23

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	129
9.3	memoir	131
9.4	amsmath	132
9.5	mathtools	134
9.6	breqn	134
9.7	listings	135
9.8	enumitem	136
9.9	subcaption	137
9.10	subfig	137
10	Language files	138
10.1	Localization guidelines	138
10.2	English	140
10.3	German	144
10.4	French	153
10.5	Portuguese	157
10.6	Spanish	161
10.7	Dutch	166
10.8	Italian	170
10.9	Russian	174
	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} {2024-08-23} {0.4.6}
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_resettors_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@enclnt` 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@enclnt }
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
\l__zrefclever_tmpa_seq
\g__zrefclever_tmpa_seq
\l__zrefclever_tmpa_bool
\l__zrefclever_tmpa_int
134 \tl_new:N \l__zrefclever_tmpa_tl
135 \tl_new:N \l__zrefclever_tmpb_tl
136 \seq_new:N \l__zrefclever_tmpa_seq
137 \seq_new:N \g__zrefclever_tmpa_seq
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:NNTF      \__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:NNTF.)

```

\__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:nm {#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             {
1158                 \__zrefclever_opt_tl_gclear_if_new:c
1159                 {
1160                     \__zrefclever_opt_varname_lang_type:eenn
1161                     { \l__zrefclever_setup_language_tl }
1162                     { \l__zrefclever_setup_type_tl }
1163                     { endrangefunc } { tl }
1164                 }
1165                 \__zrefclever_opt_tl_gclear_if_new:c
1166                 {
1167                     \__zrefclever_opt_varname_lang_type:eenn
1168                     { \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:eenn
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:eenn
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:nm`, 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:nm` 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_tl_clear:c
1684     {
1685         \__zrefclever_opt_varname_general:nn
1686         { endrangeprop } { t1 }
1687     }
1688 }
1689
1690 { pagecomp }
1691 {
1692     \__zrefclever_opt_tl_set:cn
1693     {
1694         \__zrefclever_opt_varname_general:nn
1695         { endrangefunc } { t1 }
1696     }
1697     { __zrefclever_get_endrange_pagecomp }
1698     \__zrefclever_opt_tl_clear:c
1699     {
1700         \__zrefclever_opt_varname_general:nn
1701         { endrangeprop } { t1 }
1702     }
1703 }
1704
1705 { pagecomp2 }
1706 {
1707     \__zrefclever_opt_tl_set:cn
1708     {
1709         \__zrefclever_opt_varname_general:nn
1710         { endrangefunc } { t1 }
1711     }
1712     { __zrefclever_get_endrange_pagecomptwo }
1713     \__zrefclever_opt_tl_clear:c
1714     {
1715         \__zrefclever_opt_varname_general:nn
1716         { endrangeprop } { t1 }
1717     }
1718 }
1719
1720 { unset }
1721 {
1722     \__zrefclever_opt_tl_unset:c
1723     {
1724         \__zrefclever_opt_varname_general:nn
1725         { endrangefunc } { t1 }
1726     }
1727     \__zrefclever_opt_tl_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/endorange-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   }

```



```

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 / iffinaal .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   nonnudge .meta:n = { nudge = false } ,
2235   nonnudge .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 `tcolorbox` 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_striprefix }

```

```

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:nnn
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:nnn { 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:enn
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:enn
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:enn
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_rb_seq 3799 \seq_new:N \l__zrefclever_refbounds_mid_seq
  \l__zrefclever_refbounds_mid_re_seq 3800 \seq_new:N \l__zrefclever_refbounds_mid_rb_seq
  \l__zrefclever_refbounds_last_seq 3801 \seq_new:N \l__zrefclever_refbounds_mid_re_seq
  \l__zrefclever_refbounds_last_pe_seq 3802 \seq_new:N \l__zrefclever_refbounds_last_seq
  \l__zrefclever_refbounds_last_re_seq 3803 \seq_new:N \l__zrefclever_refbounds_last_pe_seq
\l__zrefclever_type_first_refbounds_seq 3804 \seq_new:N \l__zrefclever_refbounds_last_re_seq
\l__zrefclever_type_first_refbounds_set_bool 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:NnTF \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 re-
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` and `\l__zrefclever_name_in_link_bool`. If a type name can't be found, `\l__zrefclever_type_name_tl` is cleared. The function takes no arguments, but is expected to be called in `__zrefclever_typeset_refs_last_of_type:` right before `__zrefclever_get_ref_first:`, which is the main consumer of the variables it sets, though not the only one (and hence this cannot be moved into `__zrefclever_get_ref_first:` itself). It also expects a number of relevant variables to have been appropriately set, and which it uses, prominently `\l__zrefclever_type_first_label_type_tl`, but also the queue itself in `\l__zrefclever_typeset_queue_curr_tl`, which should be "ready except for the first label", and the type counter `\l__zrefclever_type_count_int`.

```
4930 \cs_new_protected:Npn \__zrefclever_type_name_setup:
4931   {
4932     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
4933     {
4934       \tl_clear:N \l__zrefclever_type_name_tl
4935       \bool_set_true:N \l__zrefclever_type_name_missing_bool
4936     }
4937     {
4938       \tl_if_eq:NnTF
4939         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
4940         {
4941           \tl_clear:N \l__zrefclever_type_name_tl
4942           \bool_set_true:N \l__zrefclever_type_name_missing_bool
4943         }
4944         {
4945           % Determine whether we should use capitalization, abbreviation,
4946           % and plural.
4947           \bool_lazy_or:nnTF
4948             { \l__zrefclever_cap_bool }
4949             {
4950               \l__zrefclever_capfirst_bool &&
4951               \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
4952             }
4953             { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
4954             { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
4955           % If the queue is empty, we have a singular, otherwise, plural.
4956           \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
4957             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
4958             { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
4959           \bool_lazy_and:nnTF
4960             { \l__zrefclever_abbrev_bool }
4961             {
4962               ! \int_compare_p:nNn
4963                 { \l__zrefclever_type_count_int } = { 0 } ||
4964               ! \l__zrefclever_noabbrev_first_bool
4965             }
4966             {
4967               \tl_set:NV \l__zrefclever_name_format_fallback_tl
4968                 \l__zrefclever_name_format_tl
4969               \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
```

```

4970     }
4971     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
4972
4973 % Handle number and gender nudges.
4974 \bool_if:NT \l__zrefclever_nudge_enabled_bool
4975 {
4976   \bool_if:NTF \l__zrefclever_nudge_singular_bool
4977   {
4978     \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
4979     {
4980       \msg_warning:nne { zref-clever }
4981       { nudge-plural-when-sg }
4982       { \l__zrefclever_type_first_label_type_tl }
4983     }
4984   }
4985   {
4986     \bool_lazy_all:nT
4987     {
4988       { \l__zrefclever_nudge_comptosing_bool }
4989       { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
4990       {
4991         \int_compare_p:nNn
4992         { \l__zrefclever_label_count_int } > { 0 }
4993       }
4994     }
4995     {
4996       \msg_warning:nne { zref-clever }
4997       { nudge-comptosing }
4998       { \l__zrefclever_type_first_label_type_tl }
4999     }
5000   }
5001   \bool_lazy_and:nnT
5002   { \l__zrefclever_nudge_gender_bool }
5003   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
5004   {
5005     \__zrefclever_get_rf_opt_seq:neeN { gender }
5006     { \l__zrefclever_type_first_label_type_tl }
5007     { \l__zrefclever_ref_language_tl }
5008     \l__zrefclever_type_name_gender_seq
5009     \seq_if_in:NVF
5010     \l__zrefclever_type_name_gender_seq
5011     \l__zrefclever_ref_gender_tl
5012     {
5013       \seq_if_empty:NTF \l__zrefclever_type_name_gender_seq
5014       {
5015         \msg_warning:nneee { zref-clever }
5016         { nudge-gender-not-declared-for-type }
5017         { \l__zrefclever_ref_gender_tl }
5018         { \l__zrefclever_type_first_label_type_tl }
5019         { \l__zrefclever_ref_language_tl }
5020       }
5021       {
5022         \msg_warning:nneeee { zref-clever }
5023         { nudge-gender-mismatch }

```



```

5024         { \l__zrefclever_type_first_label_type_tl }
5025         { \l__zrefclever_ref_gender_tl }
5026         {
5027             \seq_use:Nn
5028                 \l__zrefclever_type_name_gender_seq { ,~ }
5029         }
5030         { \l__zrefclever_ref_language_tl }
5031     }
5032 }
5033 }
5034 }
5035
5036 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
5037 {
5038     \__zrefclever_opt_tl_get:cNF
5039     {
5040         \__zrefclever_opt_varname_type:een
5041         { \l__zrefclever_type_first_label_type_tl }
5042         { \l__zrefclever_name_format_tl }
5043         { tl }
5044     }
5045     \l__zrefclever_type_name_tl
5046     {
5047         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5048         {
5049             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
5050             \tl_put_left:NV \l__zrefclever_name_format_tl
5051                 \l__zrefclever_ref_decl_case_tl
5052         }
5053         \__zrefclever_opt_tl_get:cNF
5054         {
5055             \__zrefclever_opt_varname_lang_type:een
5056             { \l__zrefclever_ref_language_tl }
5057             { \l__zrefclever_type_first_label_type_tl }
5058             { \l__zrefclever_name_format_tl }
5059             { tl }
5060         }
5061         \l__zrefclever_type_name_tl
5062         {
5063             \tl_clear:N \l__zrefclever_type_name_tl
5064             \bool_set_true:N \l__zrefclever_type_name_missing_bool
5065             \msg_warning:nnee { zref-clever } { missing-name }
5066             { \l__zrefclever_name_format_tl }
5067             { \l__zrefclever_type_first_label_type_tl }
5068         }
5069     }
5070 }
5071 {
5072     \__zrefclever_opt_tl_get:cNF
5073     {
5074         \__zrefclever_opt_varname_type:een
5075         { \l__zrefclever_type_first_label_type_tl }
5076         { \l__zrefclever_name_format_tl }
5077         { tl }

```

```

5078 }
5079 \l__zrefclever_type_name_tl
5080 {
5081   \__zrefclever_opt_tl_get:cNF
5082   {
5083     \__zrefclever_opt_varname_type:een
5084     { \l__zrefclever_type_first_label_type_tl }
5085     { \l__zrefclever_name_format_fallback_tl }
5086     { tl }
5087   }
5088   \l__zrefclever_type_name_tl
5089   {
5090     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
5091     {
5092       \tl_put_left:Nn
5093         \l__zrefclever_name_format_tl { - }
5094       \tl_put_left:NV \l__zrefclever_name_format_tl
5095         \l__zrefclever_ref_decl_case_tl
5096       \tl_put_left:Nn
5097         \l__zrefclever_name_format_fallback_tl { - }
5098       \tl_put_left:NV
5099         \l__zrefclever_name_format_fallback_tl
5100         \l__zrefclever_ref_decl_case_tl
5101     }
5102     \__zrefclever_opt_tl_get:cNF
5103     {
5104       \__zrefclever_opt_varname_lang_type:eeen
5105       { \l__zrefclever_ref_language_tl }
5106       { \l__zrefclever_type_first_label_type_tl }
5107       { \l__zrefclever_name_format_tl }
5108       { tl }
5109     }
5110     \l__zrefclever_type_name_tl
5111     {
5112       \__zrefclever_opt_tl_get:cNF
5113       {
5114         \__zrefclever_opt_varname_lang_type:eeen
5115         { \l__zrefclever_ref_language_tl }
5116         { \l__zrefclever_type_first_label_type_tl }
5117         { \l__zrefclever_name_format_fallback_tl }
5118         { tl }
5119       }
5120       \l__zrefclever_type_name_tl
5121       {
5122         \tl_clear:N \l__zrefclever_type_name_tl
5123         \bool_set_true:N
5124           \l__zrefclever_type_name_missing_bool
5125         \msg_warning:nnee { zref-clever }
5126           { missing-name }
5127           { \l__zrefclever_name_format_tl }
5128           { \l__zrefclever_type_first_label_type_tl }
5129       }
5130     }
5131   }

```

```

5132     }
5133   }
5134 }
5135 }
5136
5137 % Signal whether the type name is to be included in the hyperlink or not.
5138 \bool_lazy_any:nTF
5139 {
5140   { ! \l__zrefclever_hyperlink_bool }
5141   { \l__zrefclever_link_star_bool }
5142   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
5143   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
5144 }
5145 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5146 {
5147   \bool_lazy_any:nTF
5148   {
5149     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
5150     {
5151       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
5152       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
5153     }
5154     {
5155       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
5156       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
5157       \l__zrefclever_typeset_last_bool &&
5158       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
5159     }
5160   }
5161   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
5162   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
5163 }
5164 }

```

(End of definition for `__zrefclever_type_name_setup:`)

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

```

\__zrefclever_hyperlink:nnn {\url/file} {\anchor} {\text}
5165 \cs_new_protected:Npn \__zrefclever_hyperlink:nnn #1#2#3
5166 {
5167   \tl_if_empty:nTF {#1}
5168     { \hyperlink {#2} {#3} }
5169     { \hyper@linkfile {#3} {#1} {#2} }
5170 }

```

(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`.

```

5171 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
5172 {
5173   \zref@ifpropundefined { urluse }
5174   { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5175   {
5176     \zref@ifrefcontainsprop {#1} { urluse }
5177     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { } }
5178     { \__zrefclever_extract_unexp:nnn {#1} { url } { } }
5179   }
5180 }
5181 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End of definition for __zrefclever_extract_url_unexp:n.)

__zrefclever_labels_in_sequence:nn

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

```

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

5182 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
5183 {
5184   \exp_args:Nee \tl_if_eq:nnT
5185   { \__zrefclever_extract_unexp:nnn {#1} { externaldocument } { } }
5186   { \__zrefclever_extract_unexp:nnn {#2} { externaldocument } { } }
5187   {
5188     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
5189     {
5190       \exp_args:Nee \tl_if_eq:nnT
5191       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
5192       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
5193       {
5194         \int_compare:nNnTF
5195         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
5196         =
5197         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5198         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
5199         {
5200           \int_compare:nNnTF
5201           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
5202           =
5203           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
5204           {
5205             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
5206             \bool_set_true:N \l__zrefclever_next_is_same_bool
5207           }
5208         }
5209       }
5210     }
5211   {
5212     \exp_args:Nee \tl_if_eq:nnT

```

```

5213 { \_zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
5214 { \_zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
5215 {
5216   \exp_args:Nee \tl_if_eq:nnT
5217   { \_zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
5218   { \_zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
5219   {
5220     \int_compare:nNnTF
5221     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
5222     =
5223     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5224     { \bool_set_true:N \l_zrefclever_next_maybe_range_bool }
5225     {
5226       \int_compare:nNnT
5227       { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
5228       =
5229       { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
5230       {

```

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.

```

5231   \exp_args:Nee \tl_if_eq:nnT
5232   {
5233     \_zrefclever_extract_unexp:nvn {#1}
5234     { l_zrefclever_ref_property_tl } { }
5235   }
5236   {
5237     \_zrefclever_extract_unexp:nvn {#2}
5238     { l_zrefclever_ref_property_tl } { }
5239   }
5240   {
5241     \bool_set_true:N
5242     \l_zrefclever_next_maybe_range_bool
5243     \bool_set_true:N
5244     \l_zrefclever_next_is_same_bool
5245   }
5246 }
5247 }
5248 }
5249 }
5250 }
5251 }
5252 }

```

(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>}
5253 \cs_new_protected:Npn \__zrefclever_get_rf_opt_tl:nnnN #1#2#3#4
5254 {
5255   % First attempt: general options.
5256   \__zrefclever_opt_tl_get:cNF
5257   { \__zrefclever_opt_varname_general:nn {#1} { tl } }
5258   #4
5259   {
5260     % If not found, try type specific options.
5261     \__zrefclever_opt_tl_get:cNF
5262     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { tl } }
5263     #4
5264     {
5265       % If not found, try type- and language-specific.
5266       \__zrefclever_opt_tl_get:cNF
5267       { \__zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { tl } }
5268       #4
5269       {
5270         % If not found, try language-specific default.
5271         \__zrefclever_opt_tl_get:cNF
5272         { \__zrefclever_opt_varname_lang_default:nnn {#3} {#1} { tl } }
5273         #4
5274         {
5275           % If not found, try fallback.
5276           \__zrefclever_opt_tl_get:cNF
5277           { \__zrefclever_opt_varname_fallback:nn {#1} { tl } }
5278           #4
5279           { \tl_clear:N #4 }
5280         }
5281       }
5282     }
5283   }
5284 }
5285 \cs_generate_variant:Nn \__zrefclever_get_rf_opt_tl:nnnN { neeN }

```

(End of definition for __zrefclever_get_rf_opt_tl:nnnN.)

```

\__zrefclever_get_rf_opt_seq:nnnN
\__zrefclever_get_rf_opt_seq:nnnN {<option>}
  {<ref type>} {<language>} {<seq variable>}
5286 \cs_new_protected:Npn \__zrefclever_get_rf_opt_seq:nnnN #1#2#3#4
5287 {
5288   % First attempt: general options.
5289   \__zrefclever_opt_seq_get:cNF
5290   { \__zrefclever_opt_varname_general:nn {#1} { seq } }
5291   #4
5292   {
5293     % If not found, try type specific options.
5294     \__zrefclever_opt_seq_get:cNF
5295     { \__zrefclever_opt_varname_type:nnn {#2} {#1} { seq } }
5296     #4
5297     {
5298       % If not found, try type- and language-specific.
5299       \__zrefclever_opt_seq_get:cNF

```

```

5300     { \_zrefclever_opt_varname_lang_type:nnnn {#3} {#2} {#1} { seq } }
5301     #4
5302     {
5303     % If not found, try language-specific default.
5304     \_zrefclever_opt_seq_get:cNF
5305     { \_zrefclever_opt_varname_lang_default:nnn {#3} {#1} { seq } }
5306     #4
5307     {
5308     % If not found, try fallback.
5309     \_zrefclever_opt_seq_get:cNF
5310     { \_zrefclever_opt_varname_fallback:nn {#1} { seq } }
5311     #4
5312     { \seq_clear:N #4 }
5313     }
5314     }
5315     }
5316     }
5317     }
5318 \cs_generate_variant:Nn \_zrefclever_get_rf_opt_seq:nnnN { neeN }

```

(End of definition for _zrefclever_get_rf_opt_seq:nnnN.)

```

\_zrefclever_get_rf_opt_bool:nnnnN    \_zrefclever_get_rf_opt_bool:nN {<option>} {<default>}
                                       {<ref type>} {<language>} {<bool variable>}
5319 \cs_new_protected:Npn \_zrefclever_get_rf_opt_bool:nnnnN #1#2#3#4#5
5320 {
5321 % First attempt: general options.
5322 \_zrefclever_opt_bool_get:cNF
5323 { \_zrefclever_opt_varname_general:nn {#1} { bool } }
5324 #5
5325 {
5326 % If not found, try type specific options.
5327 \_zrefclever_opt_bool_get:cNF
5328 { \_zrefclever_opt_varname_type:nnn {#3} {#1} { bool } }
5329 #5
5330 {
5331 % If not found, try type- and language-specific.
5332 \_zrefclever_opt_bool_get:cNF
5333 { \_zrefclever_opt_varname_lang_type:nnnn {#4} {#3} {#1} { bool } }
5334 #5
5335 {
5336 % If not found, try language-specific default.
5337 \_zrefclever_opt_bool_get:cNF
5338 { \_zrefclever_opt_varname_lang_default:nnn {#4} {#1} { bool } }
5339 #5
5340 {
5341 % If not found, try fallback.
5342 \_zrefclever_opt_bool_get:cNF
5343 { \_zrefclever_opt_varname_fallback:nn {#1} { bool } }
5344 #5
5345 { \use:c { bool_set_ #2 :N } #5 }
5346 }
5347 }
5348 }

```

```

5349     }
5350   }
5351 \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.

```

5352 \__zrefclever_compat_module:nm { appendix }
5353 {
5354   \newcounter { zc@appendix }
5355   \cs_if_exist:cTF { chapter }
5356   {
5357     \__zrefclever_zcsetup:e
5358     {
5359       counterresetby =
5360       {

```

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

```

5361         zc@appendix = \__zrefclever_counter_reset_by:n { chapter } ,
5362         chapter = zc@appendix ,
5363       } ,
5364     }
5365   }
5366   {
5367     \cs_if_exist:cT { section }
5368     {
5369       \__zrefclever_zcsetup:e

```



```

5370         {
5371             counterresetby =
5372             {
5373                 zc@appendix = \__zrefclever_counter_reset_by:n { section } ,
5374                 section = zc@appendix ,
5375             } ,
5376         }
5377     }
5378 }
5379 \AddToHook { cmd / appendix / before }
5380 {
5381     \setcounter { zc@appendix } { 1 }
5382     \__zrefclever_zcsetup:n
5383     {
5384         countertype =
5385         {
5386             chapter      = appendix ,
5387             section      = appendix ,
5388             subsection   = appendix ,
5389             subsubsection = appendix ,
5390             paragraph    = appendix ,
5391             subparagraph = appendix ,
5392         }
5393     }
5394 }
5395 }

```

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.

```

5396 \__zrefclever_compat_module:nm { appendices }
5397 {
5398     \__zrefclever_if_package_loaded:nT { appendix }
5399     {
5400         \AddToHook { env / appendices / begin }
5401         {

```

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.

```

5402         \setcounter { zc@appendix } { 1 }
5403         \__zrefclever_zcsetup:n
5404         {
5405             countertype =
5406             {
5407                 chapter      = appendix ,
5408                 section      = appendix ,
5409                 subsection   = appendix ,
5410                 subsubsection = appendix ,
5411                 paragraph    = appendix ,
5412                 subparagraph = appendix ,
5413             }
5414         }
5415     }
5416     \AddToHook { env / appendices / end }
5417     { \setcounter { zc@appendix } { 0 } }
5418     \newcounter { zc@subappendix }
5419     \cs_if_exist:cTF { chapter }
5420     {
5421         \__zrefclever_zcsetup:e
5422         {
5423             counterresetby =
5424             {
5425                 zc@subappendix = \__zrefclever_counter_reset_by:n { section } ,
5426                 section = zc@subappendix ,
5427             } ,
5428         }
5429     }
5430     {
5431         \__zrefclever_zcsetup:e
5432         {
5433             counterresetby =
5434             {
5435                 zc@subappendix = \__zrefclever_counter_reset_by:n { subsection } ,
5436                 subsection = zc@subappendix ,
5437             } ,
5438         }
5439     }
5440     \AddToHook { env / subappendices / begin }
5441     {

```

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.

```

5442         \setcounter { zc@subappendix } { 1 }
5443         \__zrefclever_zcsetup:n
5444         {

```

```

5445         countertype =
5446         {
5447             section      = appendix ,
5448             subsection   = appendix ,
5449             subsubsection = appendix ,
5450             paragraph     = appendix ,
5451             subparagraph  = appendix ,
5452         } ,
5453     } ,
5454 }
5455 \AddToHook { env / subappendices / end }
5456 { \setcounter { zc@subappendix } { 0 } }
5457 \msg_info:nnn { zref-clever } { compat-package } { appendix }
5458 }
5459 }

```

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, verfootnotes, 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.

```

5460 \__zrefclever_compat_module:nn { memoir }
5461 {
5462     \__zrefclever_if_class_loaded:nT { memoir }
5463     {

```

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`.

```

5464     \__zrefclever_zcsetup:n
5465     {
5466         countertype =
5467         {
5468             subfigure = figure ,
5469             subtable  = table ,
5470             poemline  = line ,
5471         } ,
5472         counterresetby =
5473         {
5474             subfigure = figure ,

```

```

5475         subtable = table ,
5476     } ,
5477 }

```

Support for subcaption references.

```

5478 \zref@newprop { subcaption }
5479 { \cs_if_exist_use:c { @@thesub \@captype } }
5480 \AddToHook{ memoir/subcaption/aftercounter }
5481 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

Support for \sidefootnote and \pagenote.

```

5482 \__zrefclever_zcsetup:n
5483 {
5484     countertype =
5485     {
5486         sidefootnote = footnote ,
5487         pagenote = endnote ,
5488     } ,
5489 }
5490 \msg_info:nnn { zref-clever } { compat-class } { memoir }
5491 }
5492 }

```

9.4 amsmath

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

```

5493 \__zrefclever_compat_module:nn { amsmath }
5494 {
5495     \__zrefclever_if_package_loaded:nT { amsmath }
5496     {

```

The `subequations` environment uses `parentequation` and `equation` as counters, but only the later is subject to `\refstepcounter`. What happens is: at the start, `equation` is `refstepped`, it is then stored in `parentequation` and set to ‘0’ and, at the end of the environment it is restored to the value of `parentequation`. We cannot even set `\@currentcounter` at `env/.../begin`, since the call to `\refstepcounter{equation}` done by `subequations` will override that in sequence. Unfortunately, the suggestion to set `\@currentcounter` to `parentequation` here was not accepted, see <https://github.com/latex3/latex2e/issues/687#issuecomment-951451024> and subsequent discussion. So, for `subequations`, we really must specify manually `currentcounter` and the resetting. Note that, for `subequations`, `\zlabel` works just fine (that is, if given immediately after `\begin{subequations}`, to refer to the parent equation).

```

5497     \bool_new:N \l__zrefclever_amsmath_subequations_bool
5498     \AddToHook { env / subequations / begin }
5499     {
5500         \__zrefclever_zcsetup:e
5501         {
5502             counterresetby =
5503             {
5504                 parentequation =
5505                 \__zrefclever_counter_reset_by:n { equation } ,
5506                 equation = parentequation ,

```

```

5507     } ,
5508     currentcounter = parentequation ,
5509     countertype = { parentequation = equation } ,
5510   }
5511   \bool_set_true:N \l__zrefclever_amsmath_subequations_bool
5512 }

```

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

```

5513   \zref@newprop { subeq } { \alph { equation } }
5514   \clist_map_inline:nn
5515     {
5516     equation ,
5517     equation* ,
5518     align ,
5519     align* ,
5520     alignat ,
5521     alignat* ,
5522     flalign ,
5523     flalign* ,
5524     xalignat ,
5525     xalignat* ,
5526     gather ,
5527     gather* ,
5528     multiline ,
5529     multiline* ,
5530   }
5531   {
5532     \AddToHook { env / #1 / begin }
5533     {
5534       \__zrefclever_zcsetup:n { currentcounter = equation }
5535       \bool_if:NT \l__zrefclever_amsmath_subequations_bool
5536         { \zref@localaddprop \ZREF@mainlist { subeq } }
5537     }
5538   }
5539   \msg_info:nnn { zref-clever } { compat-package } { amsmath }
5540 }
5541 }

```

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.

```
5542 \bool_new:N \l__zrefclever_mathtools_loaded_bool
5543 \__zrefclever_compat_module:nm { mathtools }
5544 {
5545   \__zrefclever_if_package_loaded:nT { mathtools }
5546   {
5547     \bool_set_true:N \l__zrefclever_mathtools_loaded_bool
5548     \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
5549     {
5550       \seq_map_inline:Nn #1
5551       {
5552         \tl_set:Nc \l__zrefclever_tmpa_tl
5553         { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
5554         \bool_lazy_or:nnT
5555         { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { equation } }
5556         { \str_if_eq_p:Vn \l__zrefclever_tmpa_tl { parentequation } }
5557         { \noeqref {##1} }
5558       }
5559     }
5560     \msg_info:nnn { zref-clever } { compat-package } { mathtools }
5561   }
5562 }
```

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.

```
5563 \__zrefclever_compat_module:nm { breqn }
5564 {
5565   \__zrefclever_if_package_loaded:nT { breqn }
5566   {
```

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>).

```

5567     \bool_new:N \l__zrefclever_breqn_dgroup_bool
5568     \AddToHook { env / dgroup / begin }
5569     {
5570         \__zrefclever_zcsetup:e
5571         {
5572             counterresetby =
5573             {
5574                 parentequation =
5575                 \__zrefclever_counter_reset_by:n { equation } ,
5576                 equation = parentequation ,
5577             } ,
5578             currentcounter = parentequation ,
5579             countertype = { parentequation = equation } ,
5580         }
5581     \bool_set_true:N \l__zrefclever_breqn_dgroup_bool
5582     }
5583     \zref@ifpropundefined { subeq }
5584     { \zref@newprop { subeq } { \alph { equation } } }
5585     { }
5586     \clist_map_inline:nn
5587     {
5588         dmath ,
5589         dseries ,
5590         darray ,
5591     }
5592     {
5593         \AddToHook { env / #1 / begin }
5594         {
5595             \__zrefclever_zcsetup:n { currentcounter = equation }
5596             \bool_if:NT \l__zrefclever_breqn_dgroup_bool
5597             { \zref@localaddprop \ZREF@mainlist { subeq } }
5598         }
5599     }
5600     \msg_info:nnn { zref-clever } { compat-package } { breqn }
5601 }
5602 }

```

9.7 listings

```

5603 \__zrefclever_compat_module:nn { listings }
5604 {
5605     \__zrefclever_if_package_loaded:nT { listings }
5606     {
5607         \__zrefclever_zcsetup:n
5608         {
5609             countertype =
5610             {

```

```

5611         lstlisting = listing ,
5612         lstnumber = line ,
5613     } ,
5614     counterresetby = { lstnumber = lstlisting } ,
5615 }

```

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.

```

5616     \lst@AddToHook { Init }
5617         { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
5618     \msg_info:nnn { zref-clever } { compat-package } { listings }
5619 }
5620 }

```

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

```

5621 \__zrefclever_compat_module:nn { enumitem }
5622 {
5623     \__zrefclever_if_package_loaded:nT { enumitem }
5624     {
5625         \int_set:Nn \l__zrefclever_tmpa_int { 5 }
5626         \bool_while_do:nn
5627             {
5628                 \cs_if_exist_p:c
5629                     { c@ enum \int_to_roman:n { \l__zrefclever_tmpa_int } }
5630             }
5631         {
5632             \__zrefclever_zcsetup:e
5633             {
5634                 counterresetby =
5635                 {
5636                     enum \int_to_roman:n { \l__zrefclever_tmpa_int } =
5637                     enum \int_to_roman:n { \l__zrefclever_tmpa_int - 1 }
5638                 } ,

```



```

5639         countertype =
5640             { enum \int_to_roman:n { \l__zrefclever_tmpa_int } = item } ,
5641         }
5642         \int_incr:N \l__zrefclever_tmpa_int
5643     }
5644     \int_compare:nNnT { \l__zrefclever_tmpa_int } > { 5 }
5645     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
5646 }
5647 }

```

9.9 subcaption

```

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

```

Support for subref reference.

```

5665     \zref@newprop { subref }
5666     { \cs_if_exist_use:c { thesub \@capttype } }
5667     \tl_put_right:Nn \caption@subtypehook
5668     { \zref@localaddprop \ZREF@mainlist { subref } }
5669 }
5670 }

```

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.

```

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

```

```

5684         subfigure = figure ,
5685         subtable = table ,
5686     } ,
5687 }
5688 }
5689 }
5690 </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 `\languagename`. 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.

```

5691 (*package)
5692 \zcDeclareLanguage { english }
5693 \zcDeclareLanguageAlias { american } { english }

```

```

5694 \zcDeclareLanguageAlias { australian } { english }
5695 \zcDeclareLanguageAlias { british } { english }
5696 \zcDeclareLanguageAlias { canadian } { english }
5697 \zcDeclareLanguageAlias { newzealand } { english }
5698 \zcDeclareLanguageAlias { UKenglish } { english }
5699 \zcDeclareLanguageAlias { USenglish } { english }
5700 \end{package}

5701 \begin{lang-english}

5702 namesep = {\nobreakspace} ,
5703 pairsep = {\and\nobreakspace} ,
5704 listsep = {,~} ,
5705 lastsep = {\and\nobreakspace} ,
5706 tpairsep = {\and\nobreakspace} ,
5707 tlistsep = {,~} ,
5708 tlastsep = {,~\and\nobreakspace} ,
5709 notesep = {} ,
5710 rangesep = {\to\nobreakspace} ,
5711
5712 type = book ,
5713   Name-sg = Book ,
5714   name-sg = book ,
5715   Name-pl = Books ,
5716   name-pl = books ,
5717
5718 type = part ,
5719   Name-sg = Part ,
5720   name-sg = part ,
5721   Name-pl = Parts ,
5722   name-pl = parts ,
5723
5724 type = chapter ,
5725   Name-sg = Chapter ,
5726   name-sg = chapter ,
5727   Name-pl = Chapters ,
5728   name-pl = chapters ,
5729
5730 type = section ,
5731   Name-sg = Section ,
5732   name-sg = section ,
5733   Name-pl = Sections ,
5734   name-pl = sections ,
5735
5736 type = paragraph ,
5737   Name-sg = Paragraph ,
5738   name-sg = paragraph ,
5739   Name-pl = Paragraphs ,
5740   name-pl = paragraphs ,
5741   Name-sg-ab = Par. ,
5742   name-sg-ab = par. ,
5743   Name-pl-ab = Par. ,
5744   name-pl-ab = par. ,
5745
5746 type = appendix ,

```

```

5747 Name-sg = Appendix ,
5748 name-sg = appendix ,
5749 Name-pl = Appendices ,
5750 name-pl = appendices ,
5751
5752 type = page ,
5753 Name-sg = Page ,
5754 name-sg = page ,
5755 Name-pl = Pages ,
5756 name-pl = pages ,
5757 rangesep = {\textendash} ,
5758 rangetopair = false ,
5759
5760 type = line ,
5761 Name-sg = Line ,
5762 name-sg = line ,
5763 Name-pl = Lines ,
5764 name-pl = lines ,
5765
5766 type = figure ,
5767 Name-sg = Figure ,
5768 name-sg = figure ,
5769 Name-pl = Figures ,
5770 name-pl = figures ,
5771 Name-sg-ab = Fig. ,
5772 name-sg-ab = fig. ,
5773 Name-pl-ab = Figs. ,
5774 name-pl-ab = figs. ,
5775
5776 type = table ,
5777 Name-sg = Table ,
5778 name-sg = table ,
5779 Name-pl = Tables ,
5780 name-pl = tables ,
5781
5782 type = item ,
5783 Name-sg = Item ,
5784 name-sg = item ,
5785 Name-pl = Items ,
5786 name-pl = items ,
5787
5788 type = footnote ,
5789 Name-sg = Footnote ,
5790 name-sg = footnote ,
5791 Name-pl = Footnotes ,
5792 name-pl = footnotes ,
5793
5794 type = endnote ,
5795 Name-sg = Note ,
5796 name-sg = note ,
5797 Name-pl = Notes ,
5798 name-pl = notes ,
5799
5800 type = note ,

```

```

5801 Name-sg = Note ,
5802 name-sg = note ,
5803 Name-pl = Notes ,
5804 name-pl = notes ,
5805
5806 type = equation ,
5807 Name-sg = Equation ,
5808 name-sg = equation ,
5809 Name-pl = Equations ,
5810 name-pl = equations ,
5811 Name-sg-ab = Eq. ,
5812 name-sg-ab = eq. ,
5813 Name-pl-ab = Eqs. ,
5814 name-pl-ab = eqs. ,
5815 refbounds-first-sg = {,(,)}, ,
5816 refbounds = {(,,)} ,
5817
5818 type = theorem ,
5819 Name-sg = Theorem ,
5820 name-sg = theorem ,
5821 Name-pl = Theorems ,
5822 name-pl = theorems ,
5823
5824 type = lemma ,
5825 Name-sg = Lemma ,
5826 name-sg = lemma ,
5827 Name-pl = Lemmas ,
5828 name-pl = lemmas ,
5829
5830 type = corollary ,
5831 Name-sg = Corollary ,
5832 name-sg = corollary ,
5833 Name-pl = Corollaries ,
5834 name-pl = corollaries ,
5835
5836 type = proposition ,
5837 Name-sg = Proposition ,
5838 name-sg = proposition ,
5839 Name-pl = Propositions ,
5840 name-pl = propositions ,
5841
5842 type = definition ,
5843 Name-sg = Definition ,
5844 name-sg = definition ,
5845 Name-pl = Definitions ,
5846 name-pl = definitions ,
5847
5848 type = proof ,
5849 Name-sg = Proof ,
5850 name-sg = proof ,
5851 Name-pl = Proofs ,
5852 name-pl = proofs ,
5853
5854 type = result ,

```

```

5855 Name-sg = Result ,
5856 name-sg = result ,
5857 Name-pl = Results ,
5858 name-pl = results ,
5859
5860 type = remark ,
5861 Name-sg = Remark ,
5862 name-sg = remark ,
5863 Name-pl = Remarks ,
5864 name-pl = remarks ,
5865
5866 type = example ,
5867 Name-sg = Example ,
5868 name-sg = example ,
5869 Name-pl = Examples ,
5870 name-pl = examples ,
5871
5872 type = algorithm ,
5873 Name-sg = Algorithm ,
5874 name-sg = algorithm ,
5875 Name-pl = Algorithms ,
5876 name-pl = algorithms ,
5877
5878 type = listing ,
5879 Name-sg = Listing ,
5880 name-sg = listing ,
5881 Name-pl = Listings ,
5882 name-pl = listings ,
5883
5884 type = exercise ,
5885 Name-sg = Exercise ,
5886 name-sg = exercise ,
5887 Name-pl = Exercises ,
5888 name-pl = exercises ,
5889
5890 type = solution ,
5891 Name-sg = Solution ,
5892 name-sg = solution ,
5893 Name-pl = Solutions ,
5894 name-pl = solutions ,
5895 </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`.

```

5896 *package
5897 \zcDeclareLanguage
5898 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
5899 { german }
5900 \zcDeclareLanguageAlias { ngerman } { german }
5901 \zcDeclareLanguageAlias { austrian } { german }

```



```

5902 \zcDeclareLanguageAlias { naustrian } { german }
5903 \zcDeclareLanguageAlias { swissgerman } { german }
5904 \zcDeclareLanguageAlias { nswissgerman } { german }
5905 \end{package}
5906 \begin{lang-german}

5907 namesep = {\nobreakspace} ,
5908 pairsep = {\simund\nobreakspace} ,
5909 listsep = {,~} ,
5910 lastsep = {\simund\nobreakspace} ,
5911 tpairsep = {\simund\nobreakspace} ,
5912 tlistsep = {,~} ,
5913 tlastsep = {\simund\nobreakspace} ,
5914 notesep = {\sim} ,
5915 rangesep = {\simbis\nobreakspace} ,
5916
5917 type = book ,
5918   gender = n ,
5919   case = N ,
5920     Name-sg = Buch ,
5921     Name-pl = Bücher ,
5922   case = A ,
5923     Name-sg = Buch ,
5924     Name-pl = Bücher ,
5925   case = D ,
5926     Name-sg = Buch ,
5927     Name-pl = Büchern ,
5928   case = G ,
5929     Name-sg = Buches ,
5930     Name-pl = Bücher ,
5931
5932 type = part ,
5933   gender = m ,
5934   case = N ,
5935     Name-sg = Teil ,
5936     Name-pl = Teile ,
5937   case = A ,
5938     Name-sg = Teil ,
5939     Name-pl = Teile ,
5940   case = D ,
5941     Name-sg = Teil ,
5942     Name-pl = Teilen ,
5943   case = G ,
5944     Name-sg = Teiles ,
5945     Name-pl = Teile ,
5946
5947 type = chapter ,
5948   gender = n ,
5949   case = N ,
5950     Name-sg = Kapitel ,
5951     Name-pl = Kapitel ,
5952   case = A ,
5953     Name-sg = Kapitel ,
5954     Name-pl = Kapitel ,

```

```

5955 case = D ,
5956     Name-sg = Kapitel ,
5957     Name-pl = Kapiteln ,
5958 case = G ,
5959     Name-sg = Kapitels ,
5960     Name-pl = Kapitel ,
5961
5962 type = section ,
5963     gender = m ,
5964     case = N ,
5965         Name-sg = Abschnitt ,
5966         Name-pl = Abschnitte ,
5967     case = A ,
5968         Name-sg = Abschnitt ,
5969         Name-pl = Abschnitte ,
5970     case = D ,
5971         Name-sg = Abschnitt ,
5972         Name-pl = Abschnitten ,
5973     case = G ,
5974         Name-sg = Abschnitts ,
5975         Name-pl = Abschnitte ,
5976
5977 type = paragraph ,
5978     gender = m ,
5979     case = N ,
5980         Name-sg = Absatz ,
5981         Name-pl = Absätze ,
5982     case = A ,
5983         Name-sg = Absatz ,
5984         Name-pl = Absätze ,
5985     case = D ,
5986         Name-sg = Absatz ,
5987         Name-pl = Absätzen ,
5988     case = G ,
5989         Name-sg = Absatzes ,
5990         Name-pl = Absätze ,
5991
5992 type = appendix ,
5993     gender = m ,
5994     case = N ,
5995         Name-sg = Anhang ,
5996         Name-pl = Anhänge ,
5997     case = A ,
5998         Name-sg = Anhang ,
5999         Name-pl = Anhänge ,
6000     case = D ,
6001         Name-sg = Anhang ,
6002         Name-pl = Anhängen ,
6003     case = G ,
6004         Name-sg = Anhangs ,
6005         Name-pl = Anhänge ,
6006
6007 type = page ,
6008     gender = f ,

```

```

6009 case = N ,
6010     Name-sg = Seite ,
6011     Name-pl = Seiten ,
6012 case = A ,
6013     Name-sg = Seite ,
6014     Name-pl = Seiten ,
6015 case = D ,
6016     Name-sg = Seite ,
6017     Name-pl = Seiten ,
6018 case = G ,
6019     Name-sg = Seite ,
6020     Name-pl = Seiten ,
6021 rangesep = {\textendash} ,
6022 rangetopair = false ,
6023
6024 type = line ,
6025     gender = f ,
6026     case = N ,
6027         Name-sg = Zeile ,
6028         Name-pl = Zeilen ,
6029     case = A ,
6030         Name-sg = Zeile ,
6031         Name-pl = Zeilen ,
6032     case = D ,
6033         Name-sg = Zeile ,
6034         Name-pl = Zeilen ,
6035     case = G ,
6036         Name-sg = Zeile ,
6037         Name-pl = Zeilen ,
6038
6039 type = figure ,
6040     gender = f ,
6041     case = N ,
6042         Name-sg = Abbildung ,
6043         Name-pl = Abbildungen ,
6044         Name-sg-ab = Abb. ,
6045         Name-pl-ab = Abb. ,
6046     case = A ,
6047         Name-sg = Abbildung ,
6048         Name-pl = Abbildungen ,
6049         Name-sg-ab = Abb. ,
6050         Name-pl-ab = Abb. ,
6051     case = D ,
6052         Name-sg = Abbildung ,
6053         Name-pl = Abbildungen ,
6054         Name-sg-ab = Abb. ,
6055         Name-pl-ab = Abb. ,
6056     case = G ,
6057         Name-sg = Abbildung ,
6058         Name-pl = Abbildungen ,
6059         Name-sg-ab = Abb. ,
6060         Name-pl-ab = Abb. ,
6061
6062 type = table ,

```

```

6063 gender = f ,
6064 case = N ,
6065     Name-sg = Tabelle ,
6066     Name-pl = Tabellen ,
6067 case = A ,
6068     Name-sg = Tabelle ,
6069     Name-pl = Tabellen ,
6070 case = D ,
6071     Name-sg = Tabelle ,
6072     Name-pl = Tabellen ,
6073 case = G ,
6074     Name-sg = Tabelle ,
6075     Name-pl = Tabellen ,
6076
6077 type = item ,
6078 gender = m ,
6079 case = N ,
6080     Name-sg = Punkt ,
6081     Name-pl = Punkte ,
6082 case = A ,
6083     Name-sg = Punkt ,
6084     Name-pl = Punkte ,
6085 case = D ,
6086     Name-sg = Punkt ,
6087     Name-pl = Punkten ,
6088 case = G ,
6089     Name-sg = Punktes ,
6090     Name-pl = Punkte ,
6091
6092 type = footnote ,
6093 gender = f ,
6094 case = N ,
6095     Name-sg = Fußnote ,
6096     Name-pl = Fußnoten ,
6097 case = A ,
6098     Name-sg = Fußnote ,
6099     Name-pl = Fußnoten ,
6100 case = D ,
6101     Name-sg = Fußnote ,
6102     Name-pl = Fußnoten ,
6103 case = G ,
6104     Name-sg = Fußnote ,
6105     Name-pl = Fußnoten ,
6106
6107 type = endnote ,
6108 gender = f ,
6109 case = N ,
6110     Name-sg = Endnote ,
6111     Name-pl = Endnoten ,
6112 case = A ,
6113     Name-sg = Endnote ,
6114     Name-pl = Endnoten ,
6115 case = D ,
6116     Name-sg = Endnote ,

```

```

6117     Name-pl = Endnoten ,
6118     case = G ,
6119     Name-sg = Endnote ,
6120     Name-pl = Endnoten ,
6121
6122 type = note ,
6123     gender = f ,
6124     case = N ,
6125     Name-sg = Anmerkung ,
6126     Name-pl = Anmerkungen ,
6127     case = A ,
6128     Name-sg = Anmerkung ,
6129     Name-pl = Anmerkungen ,
6130     case = D ,
6131     Name-sg = Anmerkung ,
6132     Name-pl = Anmerkungen ,
6133     case = G ,
6134     Name-sg = Anmerkung ,
6135     Name-pl = Anmerkungen ,
6136
6137 type = equation ,
6138     gender = f ,
6139     case = N ,
6140     Name-sg = Gleichung ,
6141     Name-pl = Gleichungen ,
6142     case = A ,
6143     Name-sg = Gleichung ,
6144     Name-pl = Gleichungen ,
6145     case = D ,
6146     Name-sg = Gleichung ,
6147     Name-pl = Gleichungen ,
6148     case = G ,
6149     Name-sg = Gleichung ,
6150     Name-pl = Gleichungen ,
6151     refbounds-first-sg = {,(,)}, ,
6152     refbounds = {(,,)} ,
6153
6154 type = theorem ,
6155     gender = n ,
6156     case = N ,
6157     Name-sg = Theorem ,
6158     Name-pl = Theoreme ,
6159     case = A ,
6160     Name-sg = Theorem ,
6161     Name-pl = Theoreme ,
6162     case = D ,
6163     Name-sg = Theorem ,
6164     Name-pl = Theoremen ,
6165     case = G ,
6166     Name-sg = Theorems ,
6167     Name-pl = Theoreme ,
6168
6169 type = lemma ,
6170     gender = n ,

```

```

6171 case = N ,
6172     Name-sg = Lemma ,
6173     Name-pl = Lemmata ,
6174 case = A ,
6175     Name-sg = Lemma ,
6176     Name-pl = Lemmata ,
6177 case = D ,
6178     Name-sg = Lemma ,
6179     Name-pl = Lemmata ,
6180 case = G ,
6181     Name-sg = Lemmas ,
6182     Name-pl = Lemmata ,
6183
6184 type = corollary ,
6185 gender = n ,
6186 case = N ,
6187     Name-sg = Korollar ,
6188     Name-pl = Korollare ,
6189 case = A ,
6190     Name-sg = Korollar ,
6191     Name-pl = Korollare ,
6192 case = D ,
6193     Name-sg = Korollar ,
6194     Name-pl = Korollaren ,
6195 case = G ,
6196     Name-sg = Korollars ,
6197     Name-pl = Korollare ,
6198
6199 type = proposition ,
6200 gender = m ,
6201 case = N ,
6202     Name-sg = Satz ,
6203     Name-pl = Sätze ,
6204 case = A ,
6205     Name-sg = Satz ,
6206     Name-pl = Sätze ,
6207 case = D ,
6208     Name-sg = Satz ,
6209     Name-pl = Sätzen ,
6210 case = G ,
6211     Name-sg = Satzes ,
6212     Name-pl = Sätze ,
6213
6214 type = definition ,
6215 gender = f ,
6216 case = N ,
6217     Name-sg = Definition ,
6218     Name-pl = Definitionen ,
6219 case = A ,
6220     Name-sg = Definition ,
6221     Name-pl = Definitionen ,
6222 case = D ,
6223     Name-sg = Definition ,
6224     Name-pl = Definitionen ,

```

```

6225 case = G ,
6226     Name-sg = Definition ,
6227     Name-pl = Definitionen ,
6228
6229 type = proof ,
6230     gender = m ,
6231     case = N ,
6232         Name-sg = Beweis ,
6233         Name-pl = Beweise ,
6234     case = A ,
6235         Name-sg = Beweis ,
6236         Name-pl = Beweise ,
6237     case = D ,
6238         Name-sg = Beweis ,
6239         Name-pl = Beweisen ,
6240     case = G ,
6241         Name-sg = Beweises ,
6242         Name-pl = Beweise ,
6243
6244 type = result ,
6245     gender = n ,
6246     case = N ,
6247         Name-sg = Ergebnis ,
6248         Name-pl = Ergebnisse ,
6249     case = A ,
6250         Name-sg = Ergebnis ,
6251         Name-pl = Ergebnisse ,
6252     case = D ,
6253         Name-sg = Ergebnis ,
6254         Name-pl = Ergebnissen ,
6255     case = G ,
6256         Name-sg = Ergebnisses ,
6257         Name-pl = Ergebnisse ,
6258
6259 type = remark ,
6260     gender = f ,
6261     case = N ,
6262         Name-sg = Bemerkung ,
6263         Name-pl = Bemerkungen ,
6264     case = A ,
6265         Name-sg = Bemerkung ,
6266         Name-pl = Bemerkungen ,
6267     case = D ,
6268         Name-sg = Bemerkung ,
6269         Name-pl = Bemerkungen ,
6270     case = G ,
6271         Name-sg = Bemerkung ,
6272         Name-pl = Bemerkungen ,
6273
6274 type = example ,
6275     gender = n ,
6276     case = N ,
6277         Name-sg = Beispiel ,
6278         Name-pl = Beispiele ,

```

```

6279 case = A ,
6280     Name-sg = Beispiel ,
6281     Name-pl = Beispiele ,
6282 case = D ,
6283     Name-sg = Beispiel ,
6284     Name-pl = Beispielen ,
6285 case = G ,
6286     Name-sg = Beispiels ,
6287     Name-pl = Beispiele ,
6288
6289 type = algorithm ,
6290     gender = m ,
6291     case = N ,
6292         Name-sg = Algorithmus ,
6293         Name-pl = Algorithmen ,
6294     case = A ,
6295         Name-sg = Algorithmus ,
6296         Name-pl = Algorithmen ,
6297     case = D ,
6298         Name-sg = Algorithmus ,
6299         Name-pl = Algorithmen ,
6300     case = G ,
6301         Name-sg = Algorithmus ,
6302         Name-pl = Algorithmen ,
6303
6304 type = listing ,
6305     gender = n ,
6306     case = N ,
6307         Name-sg = Listing ,
6308         Name-pl = Listings ,
6309     case = A ,
6310         Name-sg = Listing ,
6311         Name-pl = Listings ,
6312     case = D ,
6313         Name-sg = Listing ,
6314         Name-pl = Listings ,
6315     case = G ,
6316         Name-sg = Listings ,
6317         Name-pl = Listings ,
6318
6319 type = exercise ,
6320     gender = f ,
6321     case = N ,
6322         Name-sg = Übungsaufgabe ,
6323         Name-pl = Übungsaufgaben ,
6324     case = A ,
6325         Name-sg = Übungsaufgabe ,
6326         Name-pl = Übungsaufgaben ,
6327     case = D ,
6328         Name-sg = Übungsaufgabe ,
6329         Name-pl = Übungsaufgaben ,
6330     case = G ,
6331         Name-sg = Übungsaufgabe ,
6332         Name-pl = Übungsaufgaben ,

```



```

6333
6334 type = solution ,
6335   gender = f ,
6336   case = N ,
6337     Name-sg = Lösung ,
6338     Name-pl = Lösungen ,
6339   case = A ,
6340     Name-sg = Lösung ,
6341     Name-pl = Lösungen ,
6342   case = D ,
6343     Name-sg = Lösung ,
6344     Name-pl = Lösungen ,
6345   case = G ,
6346     Name-sg = Lösung ,
6347     Name-pl = Lösungen ,
6348 </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`.

```

6349 <*package>
6350 \zcDeclareLanguage [ gender = { f , m } ] { french }
6351 \zcDeclareLanguageAlias { acadian } { french }
6352 </package>
6353 <*lang-french>
6354 namesep = {\nobreakspace} ,
6355 pairsep = {\~et\nobreakspace} ,
6356 listsep = {,~} ,
6357 lastsep = {\~et\nobreakspace} ,
6358 tpairsep = {\~et\nobreakspace} ,
6359 tlistsep = {,~} ,
6360 tlastsep = {\~et\nobreakspace} ,
6361 notesep = {\~} ,
6362 rangesep = {\~à\nobreakspace} ,
6363
6364 type = book ,
6365   gender = m ,
6366   Name-sg = Livre ,
6367   name-sg = livre ,
6368   Name-pl = Livres ,
6369   name-pl = livres ,
6370
6371 type = part ,
6372   gender = f ,
6373   Name-sg = Partie ,
6374   name-sg = partie ,

```

```

6375 Name-pl = Parties ,
6376 name-pl = parties ,
6377
6378 type = chapter ,
6379 gender = m ,
6380 Name-sg = Chapitre ,
6381 name-sg = chapitre ,
6382 Name-pl = Chapitres ,
6383 name-pl = chapitres ,
6384
6385 type = section ,
6386 gender = f ,
6387 Name-sg = Section ,
6388 name-sg = section ,
6389 Name-pl = Sections ,
6390 name-pl = sections ,
6391
6392 type = paragraph ,
6393 gender = m ,
6394 Name-sg = Paragraphe ,
6395 name-sg = paragraphe ,
6396 Name-pl = Paragraphes ,
6397 name-pl = paragraphes ,
6398
6399 type = appendix ,
6400 gender = f ,
6401 Name-sg = Annexe ,
6402 name-sg = annexe ,
6403 Name-pl = Annexes ,
6404 name-pl = annexes ,
6405
6406 type = page ,
6407 gender = f ,
6408 Name-sg = Page ,
6409 name-sg = page ,
6410 Name-pl = Pages ,
6411 name-pl = pages ,
6412 rangsep = {-} ,
6413 rangetopair = false ,
6414
6415 type = line ,
6416 gender = f ,
6417 Name-sg = Ligne ,
6418 name-sg = ligne ,
6419 Name-pl = Lignes ,
6420 name-pl = lignes ,
6421
6422 type = figure ,
6423 gender = f ,
6424 Name-sg = Figure ,
6425 name-sg = figure ,
6426 Name-pl = Figures ,
6427 name-pl = figures ,
6428

```

```

6429 type = table ,
6430   gender = f ,
6431   Name-sg = Table ,
6432   name-sg = table ,
6433   Name-pl = Tables ,
6434   name-pl = tables ,
6435
6436 type = item ,
6437   gender = m ,
6438   Name-sg = Point ,
6439   name-sg = point ,
6440   Name-pl = Points ,
6441   name-pl = points ,
6442
6443 type = footnote ,
6444   gender = f ,
6445   Name-sg = Note ,
6446   name-sg = note ,
6447   Name-pl = Notes ,
6448   name-pl = notes ,
6449
6450 type = endnote ,
6451   gender = f ,
6452   Name-sg = Note ,
6453   name-sg = note ,
6454   Name-pl = Notes ,
6455   name-pl = notes ,
6456
6457 type = note ,
6458   gender = f ,
6459   Name-sg = Note ,
6460   name-sg = note ,
6461   Name-pl = Notes ,
6462   name-pl = notes ,
6463
6464 type = equation ,
6465   gender = f ,
6466   Name-sg = Équation ,
6467   name-sg = équation ,
6468   Name-pl = Équations ,
6469   name-pl = équations ,
6470   refbounds-first-sg = {,(,)}, ,
6471   refbounds = {(,,)} ,
6472
6473 type = theorem ,
6474   gender = m ,
6475   Name-sg = Théorème ,
6476   name-sg = théorème ,
6477   Name-pl = Théorèmes ,
6478   name-pl = théorèmes ,
6479
6480 type = lemma ,
6481   gender = m ,
6482   Name-sg = Lemme ,

```

```

6483 name-sg = lemme ,
6484 Name-pl = Lemmes ,
6485 name-pl = lemmes ,
6486
6487 type = corollary ,
6488 gender = m ,
6489 Name-sg = Corollaire ,
6490 name-sg = corollaire ,
6491 Name-pl = Corollaires ,
6492 name-pl = corollaires ,
6493
6494 type = proposition ,
6495 gender = f ,
6496 Name-sg = Proposition ,
6497 name-sg = proposition ,
6498 Name-pl = Propositions ,
6499 name-pl = propositions ,
6500
6501 type = definition ,
6502 gender = f ,
6503 Name-sg = Définition ,
6504 name-sg = définition ,
6505 Name-pl = Définitions ,
6506 name-pl = définitions ,
6507
6508 type = proof ,
6509 gender = f ,
6510 Name-sg = Démonstration ,
6511 name-sg = démonstration ,
6512 Name-pl = Démonstrations ,
6513 name-pl = démonstrations ,
6514
6515 type = result ,
6516 gender = m ,
6517 Name-sg = Résultat ,
6518 name-sg = résultat ,
6519 Name-pl = Résultats ,
6520 name-pl = résultats ,
6521
6522 type = remark ,
6523 gender = f ,
6524 Name-sg = Remarque ,
6525 name-sg = remarque ,
6526 Name-pl = Remarques ,
6527 name-pl = remarques ,
6528
6529 type = example ,
6530 gender = m ,
6531 Name-sg = Exemple ,
6532 name-sg = exemple ,
6533 Name-pl = Exemples ,
6534 name-pl = exemples ,
6535
6536 type = algorithm ,

```

```

6537   gender = m ,
6538   Name-sg = Algorithme ,
6539   name-sg = algorithme ,
6540   Name-pl = Algorithmes ,
6541   name-pl = algorithmes ,
6542
6543   type = listing ,
6544   gender = m ,
6545   Name-sg = Listing ,
6546   name-sg = listing ,
6547   Name-pl = Listings ,
6548   name-pl = listings ,
6549
6550   type = exercise ,
6551   gender = m ,
6552   Name-sg = Exercice ,
6553   name-sg = exercice ,
6554   Name-pl = Exercices ,
6555   name-pl = exercices ,
6556
6557   type = solution ,
6558   gender = f ,
6559   Name-sg = Solution ,
6560   name-sg = solution ,
6561   Name-pl = Solutions ,
6562   name-pl = solutions ,
6563 </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.

```

6564 <*package>
6565 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
6566 \zcDeclareLanguageAlias { brazilian } { portuguese }
6567 \zcDeclareLanguageAlias { brazil } { portuguese }
6568 \zcDeclareLanguageAlias { portuges } { portuguese }
6569 </package>
6570 <*lang-portuguese>
6571 namesep = {\nobreakspace} ,
6572 pairsep = {\~e\nobreakspace} ,
6573 listsep = { ,\~ } ,
6574 lastsep = {\~e\nobreakspace} ,
6575 tpairsep = {\~e\nobreakspace} ,
6576 tlistsep = { ,\~ } ,
6577 tlastsep = {\~e\nobreakspace} ,
6578 notesep = {\~ } ,
6579 rangesep = {\~a\nobreakspace} ,
6580
6581 type = book ,
6582   gender = m ,

```

```

6583 Name-sg = Livro ,
6584 name-sg = livro ,
6585 Name-pl = Livros ,
6586 name-pl = livros ,
6587
6588 type = part ,
6589 gender = f ,
6590 Name-sg = Parte ,
6591 name-sg = parte ,
6592 Name-pl = Partes ,
6593 name-pl = partes ,
6594
6595 type = chapter ,
6596 gender = m ,
6597 Name-sg = Capítulo ,
6598 name-sg = capítulo ,
6599 Name-pl = Capítulos ,
6600 name-pl = capítulos ,
6601
6602 type = section ,
6603 gender = f ,
6604 Name-sg = Seção ,
6605 name-sg = seção ,
6606 Name-pl = Seções ,
6607 name-pl = seções ,
6608
6609 type = paragraph ,
6610 gender = m ,
6611 Name-sg = Parágrafo ,
6612 name-sg = parágrafo ,
6613 Name-pl = Parágrafos ,
6614 name-pl = parágrafos ,
6615 Name-sg-ab = Par. ,
6616 name-sg-ab = par. ,
6617 Name-pl-ab = Par. ,
6618 name-pl-ab = par. ,
6619
6620 type = appendix ,
6621 gender = m ,
6622 Name-sg = Apêndice ,
6623 name-sg = apêndice ,
6624 Name-pl = Apêndices ,
6625 name-pl = apêndices ,
6626
6627 type = page ,
6628 gender = f ,
6629 Name-sg = Página ,
6630 name-sg = página ,
6631 Name-pl = Páginas ,
6632 name-pl = páginas ,
6633 rangeseq = {\textendash} ,
6634 rangetopair = false ,
6635
6636 type = line ,

```

```

6637 gender = f ,
6638 Name-sg = Linha ,
6639 name-sg = linha ,
6640 Name-pl = Linhas ,
6641 name-pl = linhas ,
6642
6643 type = figure ,
6644 gender = f ,
6645 Name-sg = Figura ,
6646 name-sg = figura ,
6647 Name-pl = Figuras ,
6648 name-pl = figuras ,
6649 Name-sg-ab = Fig. ,
6650 name-sg-ab = fig. ,
6651 Name-pl-ab = Figs. ,
6652 name-pl-ab = figs. ,
6653
6654 type = table ,
6655 gender = f ,
6656 Name-sg = Tabela ,
6657 name-sg = tabela ,
6658 Name-pl = Tabelas ,
6659 name-pl = tabelas ,
6660
6661 type = item ,
6662 gender = m ,
6663 Name-sg = Item ,
6664 name-sg = item ,
6665 Name-pl = Itens ,
6666 name-pl = itens ,
6667
6668 type = footnote ,
6669 gender = f ,
6670 Name-sg = Nota ,
6671 name-sg = nota ,
6672 Name-pl = Notas ,
6673 name-pl = notas ,
6674
6675 type = endnote ,
6676 gender = f ,
6677 Name-sg = Nota ,
6678 name-sg = nota ,
6679 Name-pl = Notas ,
6680 name-pl = notas ,
6681
6682 type = note ,
6683 gender = f ,
6684 Name-sg = Nota ,
6685 name-sg = nota ,
6686 Name-pl = Notas ,
6687 name-pl = notas ,
6688
6689 type = equation ,
6690 gender = f ,

```

```

6691 Name-sg = Equação ,
6692 name-sg = equação ,
6693 Name-pl = Equações ,
6694 name-pl = equações ,
6695 Name-sg-ab = Eq. ,
6696 name-sg-ab = eq. ,
6697 Name-pl-ab = Eqs. ,
6698 name-pl-ab = eqs. ,
6699 refbounds-first-sg = {,(,)}, ,
6700 refbounds = {(,,)} ,
6701
6702 type = theorem ,
6703 gender = m ,
6704 Name-sg = Teorema ,
6705 name-sg = teorema ,
6706 Name-pl = Teoremas ,
6707 name-pl = teoremas ,
6708
6709 type = lemma ,
6710 gender = m ,
6711 Name-sg = Lema ,
6712 name-sg = lema ,
6713 Name-pl = Lemas ,
6714 name-pl = lemas ,
6715
6716 type = corollary ,
6717 gender = m ,
6718 Name-sg = Corolário ,
6719 name-sg = corolário ,
6720 Name-pl = Corolários ,
6721 name-pl = corolários ,
6722
6723 type = proposition ,
6724 gender = f ,
6725 Name-sg = Proposição ,
6726 name-sg = proposição ,
6727 Name-pl = Proposições ,
6728 name-pl = proposições ,
6729
6730 type = definition ,
6731 gender = f ,
6732 Name-sg = Definição ,
6733 name-sg = definição ,
6734 Name-pl = Definições ,
6735 name-pl = definições ,
6736
6737 type = proof ,
6738 gender = f ,
6739 Name-sg = Demonstração ,
6740 name-sg = demonstração ,
6741 Name-pl = Demonstrações ,
6742 name-pl = demonstrações ,
6743
6744 type = result ,

```



```

6745   gender = m ,
6746   Name-sg = Resultado ,
6747   name-sg = resultado ,
6748   Name-pl = Resultados ,
6749   name-pl = resultados ,
6750
6751   type = remark ,
6752   gender = f ,
6753   Name-sg = Observação ,
6754   name-sg = observação ,
6755   Name-pl = Observações ,
6756   name-pl = observações ,
6757
6758   type = example ,
6759   gender = m ,
6760   Name-sg = Exemplo ,
6761   name-sg = exemplo ,
6762   Name-pl = Exemplos ,
6763   name-pl = exemplos ,
6764
6765   type = algorithm ,
6766   gender = m ,
6767   Name-sg = Algoritmo ,
6768   name-sg = algoritmo ,
6769   Name-pl = Algoritmos ,
6770   name-pl = algoritmos ,
6771
6772   type = listing ,
6773   gender = f ,
6774   Name-sg = Listagem ,
6775   name-sg = listagem ,
6776   Name-pl = Listagens ,
6777   name-pl = listagens ,
6778
6779   type = exercise ,
6780   gender = m ,
6781   Name-sg = Exercício ,
6782   name-sg = exercício ,
6783   Name-pl = Exercícios ,
6784   name-pl = exercícios ,
6785
6786   type = solution ,
6787   gender = f ,
6788   Name-sg = Solução ,
6789   name-sg = solução ,
6790   Name-pl = Soluções ,
6791   name-pl = soluções ,
6792 </lang-portuguese>

```

10.6 Spanish

Spanish language file has been initially provided by the author.

```
6793 <{*package}
```

```

6794 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
6795 \end{package}
6796 \langspanish

6797 namesep = {\nobreakspace} ,
6798 pairsep = {\~y\nobreakspace} ,
6799 listsep = { , ~ } ,
6800 lastsep = {\~y\nobreakspace} ,
6801 tpairsep = {\~y\nobreakspace} ,
6802 tlistsep = { , ~ } ,
6803 tlastsep = {\~y\nobreakspace} ,
6804 notesep = { ~ } ,
6805 rangesep = {\~a\nobreakspace} ,
6806
6807 type = book ,
6808     gender = m ,
6809     Name-sg = Libro ,
6810     name-sg = libro ,
6811     Name-pl = Libros ,
6812     name-pl = libros ,
6813
6814 type = part ,
6815     gender = f ,
6816     Name-sg = Parte ,
6817     name-sg = parte ,
6818     Name-pl = Partes ,
6819     name-pl = partes ,
6820
6821 type = chapter ,
6822     gender = m ,
6823     Name-sg = Capítulo ,
6824     name-sg = capítulo ,
6825     Name-pl = Capítulos ,
6826     name-pl = capítulos ,
6827
6828 type = section ,
6829     gender = f ,
6830     Name-sg = Sección ,
6831     name-sg = sección ,
6832     Name-pl = Secciones ,
6833     name-pl = secciones ,
6834
6835 type = paragraph ,
6836     gender = m ,
6837     Name-sg = Párrafo ,
6838     name-sg = párrafo ,
6839     Name-pl = Párrafos ,
6840     name-pl = párrafos ,
6841
6842 type = appendix ,
6843     gender = m ,
6844     Name-sg = Apéndice ,
6845     name-sg = apéndice ,
6846     Name-pl = Apéndices ,

```

```

6847 name-pl = apéndices ,
6848
6849 type = page ,
6850 gender = f ,
6851 Name-sg = Página ,
6852 name-sg = página ,
6853 Name-pl = Páginas ,
6854 name-pl = páginas ,
6855 rangesep = {\textendash} ,
6856 rangetopair = false ,
6857
6858 type = line ,
6859 gender = f ,
6860 Name-sg = Línea ,
6861 name-sg = línea ,
6862 Name-pl = Líneas ,
6863 name-pl = líneas ,
6864
6865 type = figure ,
6866 gender = f ,
6867 Name-sg = Figura ,
6868 name-sg = figura ,
6869 Name-pl = Figuras ,
6870 name-pl = figuras ,
6871
6872 type = table ,
6873 gender = m ,
6874 Name-sg = Cuadro ,
6875 name-sg = cuadro ,
6876 Name-pl = Cuadros ,
6877 name-pl = cuadros ,
6878
6879 type = item ,
6880 gender = m ,
6881 Name-sg = Punto ,
6882 name-sg = punto ,
6883 Name-pl = Puntos ,
6884 name-pl = puntos ,
6885
6886 type = footnote ,
6887 gender = f ,
6888 Name-sg = Nota ,
6889 name-sg = nota ,
6890 Name-pl = Notas ,
6891 name-pl = notas ,
6892
6893 type = endnote ,
6894 gender = f ,
6895 Name-sg = Nota ,
6896 name-sg = nota ,
6897 Name-pl = Notas ,
6898 name-pl = notas ,
6899
6900 type = note ,

```

```

6901   gender = f ,
6902   Name-sg = Nota ,
6903   name-sg = nota ,
6904   Name-pl = Notas ,
6905   name-pl = notas ,
6906
6907   type = equation ,
6908   gender = f ,
6909   Name-sg = Ecuación ,
6910   name-sg = ecuación ,
6911   Name-pl = Ecuaciones ,
6912   name-pl = ecuaciones ,
6913   refbounds-first-sg = {,(,)}, ,
6914   refbounds = {(,,)} ,
6915
6916   type = theorem ,
6917   gender = m ,
6918   Name-sg = Teorema ,
6919   name-sg = teorema ,
6920   Name-pl = Teoremas ,
6921   name-pl = teoremas ,
6922
6923   type = lemma ,
6924   gender = m ,
6925   Name-sg = Lema ,
6926   name-sg = lema ,
6927   Name-pl = Lemas ,
6928   name-pl = lemas ,
6929
6930   type = corollary ,
6931   gender = m ,
6932   Name-sg = Corolario ,
6933   name-sg = corolario ,
6934   Name-pl = Corolarios ,
6935   name-pl = corolarios ,
6936
6937   type = proposition ,
6938   gender = f ,
6939   Name-sg = Proposición ,
6940   name-sg = proposición ,
6941   Name-pl = Proposiciones ,
6942   name-pl = proposiciones ,
6943
6944   type = definition ,
6945   gender = f ,
6946   Name-sg = Definición ,
6947   name-sg = definición ,
6948   Name-pl = Definiciones ,
6949   name-pl = definiciones ,
6950
6951   type = proof ,
6952   gender = f ,
6953   Name-sg = Demostración ,
6954   name-sg = demostración ,

```

```

6955 Name-pl = Demostraciones ,
6956 name-pl = demostraciones ,
6957
6958 type = result ,
6959 gender = m ,
6960 Name-sg = Resultado ,
6961 name-sg = resultado ,
6962 Name-pl = Resultados ,
6963 name-pl = resultados ,
6964
6965 type = remark ,
6966 gender = f ,
6967 Name-sg = Observación ,
6968 name-sg = observación ,
6969 Name-pl = Observaciones ,
6970 name-pl = observaciones ,
6971
6972 type = example ,
6973 gender = m ,
6974 Name-sg = Ejemplo ,
6975 name-sg = ejemplo ,
6976 Name-pl = Ejemplos ,
6977 name-pl = ejemplos ,
6978
6979 type = algorithm ,
6980 gender = m ,
6981 Name-sg = Algoritmo ,
6982 name-sg = algoritmo ,
6983 Name-pl = Algoritmos ,
6984 name-pl = algoritmos ,
6985
6986 type = listing ,
6987 gender = m ,
6988 Name-sg = Listado ,
6989 name-sg = listado ,
6990 Name-pl = Listados ,
6991 name-pl = listados ,
6992
6993 type = exercise ,
6994 gender = m ,
6995 Name-sg = Ejercicio ,
6996 name-sg = ejercicio ,
6997 Name-pl = Ejercicios ,
6998 name-pl = ejercicios ,
6999
7000 type = solution ,
7001 gender = f ,
7002 Name-sg = Solución ,
7003 name-sg = solución ,
7004 Name-pl = Soluciones ,
7005 name-pl = soluciones ,
7006 </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.

```
7007 (*package)
7008 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
7009 </package>
7010 (*lang-dutch)
7011 namesep = {\nobreakspace} ,
7012 pairsep = {\en\nobreakspace} ,
7013 listsep = {,~} ,
7014 lastsep = {\en\nobreakspace} ,
7015 tpairsep = {\en\nobreakspace} ,
7016 tlistsep = {,~} ,
7017 tlastsep = {,~en\nobreakspace} ,
7018 notesep = {~} ,
7019 rangesep = {\t/m\nobreakspace} ,
7020
7021 type = book ,
7022   gender = n ,
7023   Name-sg = Boek ,
7024   name-sg = boek ,
7025   Name-pl = Boeken ,
7026   name-pl = boeken ,
7027
7028 type = part ,
7029   gender = n ,
7030   Name-sg = Deel ,
7031   name-sg = deel ,
7032   Name-pl = Delen ,
7033   name-pl = delen ,
7034
7035 type = chapter ,
7036   gender = n ,
7037   Name-sg = Hoofdstuk ,
7038   name-sg = hoofdstuk ,
7039   Name-pl = Hoofdstukken ,
7040   name-pl = hoofdstukken ,
7041
7042 type = section ,
7043   gender = m ,
7044   Name-sg = Paragraaf ,
7045   name-sg = paragraaf ,
7046   Name-pl = Paragrafen ,
7047   name-pl = paragrafen ,
7048
7049 type = paragraph ,
7050   gender = f ,
7051   Name-sg = Alinea ,
7052   name-sg = alinea ,
7053   Name-pl = Alinea's ,
7054   name-pl = alinea's ,
7055
```

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.

```
7056 type = appendix ,
7057   gender = { f , m } ,
7058   Name-sg = Bijlage ,
7059   name-sg = bijlage ,
7060   Name-pl = Bijlagen ,
7061   name-pl = bijlagen ,
7062
7063 type = page ,
7064   gender = { f , m } ,
7065   Name-sg = Pagina ,
7066   name-sg = pagina ,
7067   Name-pl = Pagina's ,
7068   name-pl = pagina's ,
7069   rangesep = {\textendash} ,
7070   rangetopair = false ,
7071
7072 type = line ,
7073   gender = m ,
7074   Name-sg = Regel ,
7075   name-sg = regel ,
7076   Name-pl = Regels ,
7077   name-pl = regels ,
7078
7079 type = figure ,
7080   gender = { n , f , m } ,
7081   Name-sg = Figuur ,
7082   name-sg = figuur ,
7083   Name-pl = Figuren ,
7084   name-pl = figuren ,
7085
7086 type = table ,
7087   gender = { f , m } ,
7088   Name-sg = Tabel ,
7089   name-sg = tabel ,
7090   Name-pl = Tabellen ,
7091   name-pl = tabellen ,
7092
7093 type = item ,
7094   gender = n ,
7095   Name-sg = Punt ,
7096   name-sg = punt ,
7097   Name-pl = Punten ,
7098   name-pl = punten ,
7099
7100 type = footnote ,
7101   gender = { f , m } ,
7102   Name-sg = Voetnoot ,
7103   name-sg = voetnoot ,
7104   Name-pl = Voetnoten ,
7105   name-pl = voetnoten ,
7106
```

```

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

```


7157 Name-sg = Propositie ,
7158 name-sg = propositie ,
7159 Name-pl = Propositiones ,
7160 name-pl = proposities ,
7161
7162 type = definition ,
7163 gender = f ,
7164 Name-sg = Definitie ,
7165 name-sg = definitie ,
7166 Name-pl = Definities ,
7167 name-pl = definities ,
7168
7169 type = proof ,
7170 gender = n ,
7171 Name-sg = Bewijs ,
7172 name-sg = bewijs ,
7173 Name-pl = Bewijzen ,
7174 name-pl = bewijzen ,
7175
7176 type = result ,
7177 gender = n ,
7178 Name-sg = Resultaat ,
7179 name-sg = resultaat ,
7180 Name-pl = Resultaten ,
7181 name-pl = resultaten ,
7182
7183 type = remark ,
7184 gender = f ,
7185 Name-sg = Opmerking ,
7186 name-sg = opmerking ,
7187 Name-pl = Opmerkingen ,
7188 name-pl = opmerkingen ,
7189
7190 type = example ,
7191 gender = n ,
7192 Name-sg = Voorbeeld ,
7193 name-sg = voorbeeld ,
7194 Name-pl = Voorbeelden ,
7195 name-pl = voorbeelden ,
7196

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".

7197 type = algorithm ,
7198 gender = { n , f , m } ,
7199 Name-sg = Algoritme ,
7200 name-sg = algoritme ,
7201 Name-pl = Algoritmen ,
7202 name-pl = algoritmen ,
7203

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

7204 type = listing ,
7205 gender = m ,

```

7206 Name-sg = Listing ,
7207 name-sg = listing ,
7208 Name-pl = Listings ,
7209 name-pl = listings ,
7210
7211 type = exercise ,
7212   gender = { f , m } ,
7213 Name-sg = Opgave ,
7214 name-sg = opgave ,
7215 Name-pl = Opgaven ,
7216 name-pl = opgaven ,
7217
7218 type = solution ,
7219   gender = f ,
7220 Name-sg = Oplossing ,
7221 name-sg = oplossing ,
7222 Name-pl = Oplossingen ,
7223 name-pl = oplossingen ,
7224 </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->

```

7225 <{*package}
7226 \zcDeclareLanguage [ gender = { f , m } ] { italian }
7227 </package>
7228 <{*lang-italian}
7229 namesep = {\nobreakspace} ,
7230 pairsep = {\~e\nobreakspace} ,
7231 listsep = { , ~ } ,
7232 lastsep = {\~e\nobreakspace} ,
7233 tpairsep = {\~e\nobreakspace} ,
7234 tlistsep = { , ~ } ,
7235 tlastsep = { , ~e\nobreakspace} ,
7236 notesep = { ~ } ,
7237 rangesep = {\~a\nobreakspace} ,
7238 +refbounds-rb = {da\nobreakspace,,,} ,
7239
7240 type = book ,
7241   gender = m ,
7242 Name-sg = Libro ,
7243 name-sg = libro ,
7244 Name-pl = Libri ,
7245 name-pl = libri ,
7246
7247 type = part ,
7248   gender = f ,
7249 Name-sg = Parte ,
7250 name-sg = parte ,
7251 Name-pl = Parti ,

```

```

7252     name-pl = parti ,
7253
7254 type = chapter ,
7255     gender = m ,
7256     Name-sg = Capitolo ,
7257     name-sg = capitolo ,
7258     Name-pl = Capitoli ,
7259     name-pl = capitoli ,
7260
7261 type = section ,
7262     gender = m ,
7263     Name-sg = Paragrafo ,
7264     name-sg = paragrafo ,
7265     Name-pl = Paragrafi ,
7266     name-pl = paragrafi ,
7267
7268 type = paragraph ,
7269     gender = m ,
7270     Name-sg = Capoverso ,
7271     name-sg = capoverso ,
7272     Name-pl = Capoversi ,
7273     name-pl = capoversi ,
7274
7275 type = appendix ,
7276     gender = f ,
7277     Name-sg = Appendice ,
7278     name-sg = appendice ,
7279     Name-pl = Appendici ,
7280     name-pl = appendici ,
7281
7282 type = page ,
7283     gender = f ,
7284     Name-sg = Pagina ,
7285     name-sg = pagina ,
7286     Name-pl = Pagine ,
7287     name-pl = pagine ,
7288     Name-sg-ab = Pag. ,
7289     name-sg-ab = pag. ,
7290     Name-pl-ab = Pag. ,
7291     name-pl-ab = pag. ,
7292     rangesep = {\textendash} ,
7293     rangetopair = false ,
7294     +refbounds-rb = {,,} ,
7295
7296 type = line ,
7297     gender = f ,
7298     Name-sg = Riga ,
7299     name-sg = riga ,
7300     Name-pl = Righe ,
7301     name-pl = righe ,
7302
7303 type = figure ,
7304     gender = f ,
7305     Name-sg = Figura ,

```

```

7306 name-sg = figura ,
7307 Name-pl = Figure ,
7308 name-pl = figure ,
7309 Name-sg-ab = Fig. ,
7310 name-sg-ab = fig. ,
7311 Name-pl-ab = Fig. ,
7312 name-pl-ab = fig. ,
7313
7314 type = table ,
7315 gender = f ,
7316 Name-sg = Tabella ,
7317 name-sg = tabella ,
7318 Name-pl = Tabelle ,
7319 name-pl = tabelle ,
7320 Name-sg-ab = Tab. ,
7321 name-sg-ab = tab. ,
7322 Name-pl-ab = Tab. ,
7323 name-pl-ab = tab. ,
7324
7325 type = item ,
7326 gender = m ,
7327 Name-sg = Punto ,
7328 name-sg = punto ,
7329 Name-pl = Punti ,
7330 name-pl = punti ,
7331
7332 type = footnote ,
7333 gender = f ,
7334 Name-sg = Nota ,
7335 name-sg = nota ,
7336 Name-pl = Note ,
7337 name-pl = note ,
7338
7339 type = endnote ,
7340 gender = f ,
7341 Name-sg = Nota ,
7342 name-sg = nota ,
7343 Name-pl = Note ,
7344 name-pl = note ,
7345
7346 type = note ,
7347 gender = f ,
7348 Name-sg = Nota ,
7349 name-sg = nota ,
7350 Name-pl = Note ,
7351 name-pl = note ,
7352
7353 type = equation ,
7354 gender = f ,
7355 Name-sg = Equazione ,
7356 name-sg = equazione ,
7357 Name-pl = Equazioni ,
7358 name-pl = equazioni ,
7359 Name-sg-ab = Eq. ,

```

```

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

```

```

7414   name-pl = risultati ,
7415
7416 type = remark ,
7417   gender = f ,
7418   Name-sg = Osservazione ,
7419   name-sg = osservazione ,
7420   Name-pl = Osservazioni ,
7421   name-pl = osservazioni ,
7422
7423 type = example ,
7424   gender = m ,
7425   Name-sg = Esempio ,
7426   name-sg = esempio ,
7427   Name-pl = Esempi ,
7428   name-pl = esempi ,
7429
7430 type = algorithm ,
7431   gender = m ,
7432   Name-sg = Algoritmo ,
7433   name-sg = algoritmo ,
7434   Name-pl = Algoritmi ,
7435   name-pl = algoritmi ,
7436
7437 type = listing ,
7438   gender = m ,
7439   Name-sg = Listato ,
7440   name-sg = listato ,
7441   Name-pl = Listati ,
7442   name-pl = listati ,
7443
7444 type = exercise ,
7445   gender = m ,
7446   Name-sg = Esercizio ,
7447   name-sg = esercizio ,
7448   Name-pl = Esercizi ,
7449   name-pl = esercizi ,
7450
7451 type = solution ,
7452   gender = f ,
7453   Name-sg = Soluzione ,
7454   name-sg = soluzione ,
7455   Name-pl = Soluzioni ,
7456   name-pl = soluzioni ,
7457 </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.”.

```

7458 <{*package}

```

```

7459 \zcDeclareLanguage
7460 [ declension = { n , a , g , d , i , p } , gender = { f , m , n } ]
7461 { russian }
7462 \</package>
7463 \<lang-russian>

7464 namesep = {\nobreakspace} ,
7465 pairsep = {\~и\nobreakspace} ,
7466 listsep = { ,\~ } ,
7467 lastsep = {\~и\nobreakspace} ,
7468 tpairsep = {\~и\nobreakspace} ,
7469 tlistsep = { ,\~ } ,
7470 tlastsep = { ,\~и\nobreakspace} ,
7471 notesep = {\~ } ,
7472 rangesep = {\~по\nobreakspace} ,
7473 +refbounds-rb = {c\nobreakspace,,} ,
7474
7475 type = book ,
7476 gender = f ,
7477 case = n ,
7478 Name-sg = Книга ,
7479 name-sg = книга ,
7480 Name-pl = Книги ,
7481 name-pl = книги ,
7482 case = a ,
7483 Name-sg = Книгу ,
7484 name-sg = книгу ,
7485 Name-pl = Книги ,
7486 name-pl = книги ,
7487 case = g ,
7488 Name-sg = Книги ,
7489 name-sg = книги ,
7490 Name-pl = Книг ,
7491 name-pl = книг ,
7492 case = d ,
7493 Name-sg = Книге ,
7494 name-sg = книге ,
7495 Name-pl = Книгам ,
7496 name-pl = книгам ,
7497 case = i ,
7498 Name-sg = Книгой ,
7499 name-sg = книгой ,
7500 Name-pl = Книгами ,
7501 name-pl = книгами ,
7502 case = p ,
7503 Name-sg = Книге ,
7504 name-sg = книге ,
7505 Name-pl = Книгах ,
7506 name-pl = книгах ,
7507
7508 type = part ,
7509 gender = f ,
7510 case = n ,
7511 Name-sg = Часть ,

```

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

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

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

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

```

7728     Name-sg-ab = C. ,
7729     name-sg-ab = c. ,
7730     Name-pl-ab = Сс. ,
7731     name-pl-ab = cc. ,
7732     case = a ,
7733     Name-sg = Страницу ,
7734     name-sg = страницу ,
7735     Name-pl = Страницы ,
7736     name-pl = страницы ,
7737     Name-sg-ab = C. ,
7738     name-sg-ab = c. ,
7739     Name-pl-ab = Сс. ,
7740     name-pl-ab = cc. ,
7741     case = g ,
7742     Name-sg = Страницы ,
7743     name-sg = страницы ,
7744     Name-pl = Страниц ,
7745     name-pl = страниц ,
7746     Name-sg-ab = C. ,
7747     name-sg-ab = c. ,
7748     Name-pl-ab = Сс. ,
7749     name-pl-ab = cc. ,
7750     case = d ,
7751     Name-sg = Странице ,
7752     name-sg = странице ,
7753     Name-pl = Страницам ,
7754     name-pl = страницам ,
7755     Name-sg-ab = C. ,
7756     name-sg-ab = c. ,
7757     Name-pl-ab = Сс. ,
7758     name-pl-ab = cc. ,
7759     case = i ,
7760     Name-sg = Страницей ,
7761     name-sg = страницей ,
7762     Name-pl = Страницами ,
7763     name-pl = страницами ,
7764     Name-sg-ab = C. ,
7765     name-sg-ab = c. ,
7766     Name-pl-ab = Сс. ,
7767     name-pl-ab = cc. ,
7768     case = p ,
7769     Name-sg = Странице ,
7770     name-sg = странице ,
7771     Name-pl = Страницах ,
7772     name-pl = страницах ,
7773     Name-sg-ab = C. ,
7774     name-sg-ab = c. ,
7775     Name-pl-ab = Сс. ,
7776     name-pl-ab = cc. ,
7777     rangesep = {\textendash} ,
7778     rangetopair = false ,
7779     +refbounds-rb = {,,} ,
7780
7781     type = line ,

```

```

7782 gender = f ,
7783 case = n ,
7784     Name-sg = Строка ,
7785     name-sg = строка ,
7786     Name-pl = Строки ,
7787     name-pl = строки ,
7788 case = a ,
7789     Name-sg = Строку ,
7790     name-sg = строку ,
7791     Name-pl = Строки ,
7792     name-pl = строки ,
7793 case = g ,
7794     Name-sg = Строки ,
7795     name-sg = строки ,
7796     Name-pl = Строк ,
7797     name-pl = строк ,
7798 case = d ,
7799     Name-sg = Строке ,
7800     name-sg = строке ,
7801     Name-pl = Строкам ,
7802     name-pl = строкам ,
7803 case = i ,
7804     Name-sg = Строкой ,
7805     name-sg = строкой ,
7806     Name-pl = Строками ,
7807     name-pl = строками ,
7808 case = p ,
7809     Name-sg = Строке ,
7810     name-sg = строке ,
7811     Name-pl = Строках ,
7812     name-pl = строках ,
7813
7814 type = figure ,
7815 gender = m ,
7816 case = n ,
7817     Name-sg = Рисунок ,
7818     name-sg = рисунок ,
7819     Name-pl = Рисунки ,
7820     name-pl = рисунки ,
7821     Name-sg-ab = Рис. ,
7822     name-sg-ab = рис. ,
7823     Name-pl-ab = Рис. ,
7824     name-pl-ab = рис. ,
7825 case = a ,
7826     Name-sg = Рисунок ,
7827     name-sg = рисунок ,
7828     Name-pl = Рисунки ,
7829     name-pl = рисунки ,
7830     Name-sg-ab = Рис. ,
7831     name-sg-ab = рис. ,
7832     Name-pl-ab = Рис. ,
7833     name-pl-ab = рис. ,
7834 case = g ,
7835     Name-sg = Рисунка ,

```

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

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

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


```

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

```

```

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

```

```

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

```

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

```

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

```

8268 gender = n ,
8269 case = n ,
8270 Name-sg = Предложение ,
8271 name-sg = предложение ,
8272 Name-pl = Предложения ,
8273 name-pl = предложения ,
8274 Name-sg-ab = Предл. ,
8275 name-sg-ab = предл. ,
8276 Name-pl-ab = Предл. ,
8277 name-pl-ab = предл. ,
8278 case = a ,
8279 Name-sg = Предложение ,
8280 name-sg = предложение ,
8281 Name-pl = Предложения ,
8282 name-pl = предложения ,
8283 Name-sg-ab = Предл. ,
8284 name-sg-ab = предл. ,
8285 Name-pl-ab = Предл. ,
8286 name-pl-ab = предл. ,
8287 case = g ,
8288 Name-sg = Предложения ,
8289 name-sg = предложения ,
8290 Name-pl = Предложений ,
8291 name-pl = предложений ,
8292 Name-sg-ab = Предл. ,
8293 name-sg-ab = предл. ,
8294 Name-pl-ab = Предл. ,
8295 name-pl-ab = предл. ,
8296 case = d ,
8297 Name-sg = Предложению ,
8298 name-sg = предложению ,
8299 Name-pl = Предложениям ,
8300 name-pl = предложениям ,
8301 Name-sg-ab = Предл. ,
8302 name-sg-ab = предл. ,
8303 Name-pl-ab = Предл. ,
8304 name-pl-ab = предл. ,
8305 case = i ,
8306 Name-sg = Предложением ,
8307 name-sg = предложением ,
8308 Name-pl = Предложениями ,
8309 name-pl = предложениями ,
8310 Name-sg-ab = Предл. ,
8311 name-sg-ab = предл. ,
8312 Name-pl-ab = Предл. ,
8313 name-pl-ab = предл. ,
8314 case = p ,
8315 Name-sg = Предложении ,
8316 name-sg = предложении ,
8317 Name-pl = Предложениях ,
8318 name-pl = предложениях ,
8319 Name-sg-ab = Предл. ,
8320 name-sg-ab = предл. ,
8321 Name-pl-ab = Предл. ,

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

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

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

```

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

```

```

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

```

```

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

```

8646 Name-sg-ab = Упр. ,
8647 name-sg-ab = упр. ,
8648 Name-pl-ab = Упр. ,
8649 name-pl-ab = упр. ,
8650 case = p ,
8651 Name-sg = Упражнения ,
8652 name-sg = упражнения ,
8653 Name-pl = Упражнениях ,
8654 name-pl = упражнениях ,
8655 Name-sg-ab = Упр. ,
8656 name-sg-ab = упр. ,
8657 Name-pl-ab = Упр. ,
8658 name-pl-ab = упр. ,
8659
8660 type = solution ,
8661 gender = n ,
8662 case = n ,
8663 Name-sg = Решение ,
8664 name-sg = решение ,
8665 Name-pl = Решения ,
8666 name-pl = решения ,
8667 case = a ,
8668 Name-sg = Решение ,
8669 name-sg = решение ,
8670 Name-pl = Решения ,
8671 name-pl = решения ,
8672 case = g ,
8673 Name-sg = Решения ,
8674 name-sg = решения ,
8675 Name-pl = Решений ,
8676 name-pl = решений ,
8677 case = d ,
8678 Name-sg = Решению ,
8679 name-sg = решению ,
8680 Name-pl = Решениям ,
8681 name-pl = решениям ,
8682 case = i ,
8683 Name-sg = Решением ,
8684 name-sg = решением ,
8685 Name-pl = Решениями ,
8686 name-pl = решениями ,
8687 case = p ,
8688 Name-sg = Решении ,
8689 name-sg = решении ,
8690 Name-pl = Решениях ,
8691 name-pl = решениях ,
8692 </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>	120, 2061, 2105, 2128, 2159, 2161, 2199, 2272, 2314, 2468, 2481, 2489, 5379, 5400, 5416, 5440, 5455, 5480, 5498, 5532, 5568, 5593
<code>\AddToHookWithArguments</code>	2443
<code>\alph</code>	5513, 5584
<code>\appendix</code>	2, 128, 129, 131, 138
<code>\appendixname</code>	128, 140
<code>\AtEndOfPackage</code>	2479
B	
<code>\babelname</code>	2115
<code>\babelprovide</code>	30, 56
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, 4974, 4976, 5535, 5596
<code>\bool_if:nTF</code> 86, 3585, 3595, 3619, 3636, 3651, 3716, 3724, 4365, 4742, 4783, 4864, 4881
<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, 4986
<code>\bool_lazy_and:nnTF</code> ..	49, 54, 2615, 3442, 3463, 4246, 4317, 4695, 4959, 5001
<code>\bool_lazy_any:nTF</code>	5138, 5147
<code>\bool_lazy_or:nnTF</code>	1396, 1429, 1986, 2561, 2828, 3330, 3355, 3446, 4947, 5554
<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, 5497, 5542, 5567
<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, 5145, 5162
<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, 4935, 4942, 5064, 5123, 5161, 5198, 5205, 5206, 5224, 5241, 5243, 5511, 5547, 5581
<code>\bool_until_do:Nn</code>	
.....	1881, 1935, 1980, 3617, 3842
<code>\bool_while_do:nn</code>	5626
C	
<code>\chaptername</code>	140
clist commands:	
<code>\clist_map_inline:nn</code> ...	651, 699, 716, 1035, 2241, 2457, 2967, 5514, 5586
<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, 5181, 5285, 5318, 5351
<code>\cs_if_exist:NTF</code>	
...	24, 27, 66, 75, 96, 5355, 5367, 5419
<code>\cs_if_exist_p:N</code>	
.....	51, 56, 4248, 4319, 4697, 5628
<code>\cs_if_exist_use:N</code>	5479, 5666
<code>\cs_new:Npn</code>	64, 73, 84, 94, 105, 301, 307, 309, 311, 314, 325, 336, 348, 350, 738, 4738, 4778, 4816, 5171
<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, 5165, 5182, 5253, 5286, 5319, 5548		3486, 4402, 4408, 4758, 4770, 4798, 4809, 4840, 4849, 4858, 4873, 4906, 4921	
<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, 5184, 5190, 5212, 5216, 5231			
<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,		
H			
<code>\hyperlink</code>	5168		
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, 5194, 5200, 5220, 5226, 5644			
<code>\int_compare_p:nNn</code> ...	1399, 1432, 1987, 1989, 2564, 2831, 3333, 3358, 3718, 3726, 4465, 4951, 4962, 4991, 5158		
<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, 5642		
<code>\int_new:N</code>	118, 139, 3495, 3496, 3758, 3759, 3760, 3773, 3774		
<code>\int_rand:n</code>	322, 333, 344		
<code>\int_set:Nn</code>	3703, 3705, 3709, 3712, 5625		
<code>\int_to_roman:n</code>	5629, 5636, 5637, 5640		
<code>\int_use:N</code>	52, 57, 62, 77, 128		
<code>\int_zero:N</code>			
3696, 3697, 3818, 3819, 3820, 3821, 3822, 4478, 4479, 4481, 4482, 4727, 4728			
iow commands:			
<code>\iow_char:N</code>			
..... 143, 158, 159, 164, 165, 170, 171, 176, 177, 229, 246, 293, 2289, 2298			
<code>\iow_newline:</code>	287		
K			
keys commands:			
<code>\l_keys_choice_tl</code>	823		
<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,			

<code>\ProvidesFile</code>	30		
R			
<code>\refstepcounter</code> ..	3, 4, 132, 133, 135, 136		
regex commands:			
<code>\regex_match:nnTF</code>	1906		
<code>\renewlist</code>	136		
<code>\RequirePackage</code>	16, 17, 18, 19, 2066		
<code>\restoreapp</code>	129		
<code>\roman</code>	8		
S			
<code>\scantokens</code>	129		
seq commands:			
<code>\seq_clear:N</code>	467, 842, 879, 960, 973, 1034, 1642, 2558, 2825, 2902, 2915, 2966, 3500, 5312		
<code>\seq_const_from_clist:Nn</code>	22		
<code>\seq_count:N</code>	1400, 1414, 1433, 1448, 2565, 2575, 2832, 2845, 3334, 3348, 3359, 3373		
<code>\seq_gclear:N</code> ..	1393, 1426, 3327, 3352		
<code>\seq_gconcat:NNN</code>	644, 648		
<code>\seq_get_left:NN</code>	857, 868, 964, 1012, 2906, 2944, 3852		
<code>\seq_gput_right:Nn</code> ...	975, 981, 2461		
<code>\seq_gremove_all:Nn</code>	2493		
<code>\seq_gset_eq:NN</code>	460, 1062		
<code>\seq_gset_from_clist:Nn</code>	587, 596, 608, 623, 631, 792, 807		
<code>\seq_gset_split:Nnn</code>	445		
<code>\seq_if_empty:NTF</code> ..	843, 880, 961, 1001, 1022, 2903, 2933, 2954, 3846, 5013		
<code>\seq_if_exist:NTF</code> ..	448, 458, 465, 476		
<code>\seq_if_in:NnTF</code> 861, 895, 939, 1007, 1037, 2459, 2492, 2939, 2969, 3544, 5009			
<code>\seq_item:Nn</code>	4748, 4753, 4759, 4761, 4764, 4765, 4771, 4772, 4789, 4794, 4799, 4801, 4804, 4805, 4810, 4811, 4842, 4843, 4850, 4852, 4887, 4899, 4907, 4910, 4914, 4915, 4922, 4923		
<code>\seq_map_break:n</code>	108, 3737, 3740		
<code>\seq_map_function:NN</code>	3503		
<code>\seq_map_indexed_inline:Nn</code> ..	45, 3698		
<code>\seq_map_inline:Nn</code>	1076, 1101, 1338, 1377, 1455, 2483, 2496, 2544, 2581, 2630, 2642, 2808, 2851, 2990, 3014, 3277, 3317, 3379, 3734, 5550		
<code>\seq_map_tokens:Nn</code>	90		
<code>\seq_new:N</code>	136, 137, 449, 459, 584, 585, 586, 595, 607, 622, 630, 643, 647, 787, 802, 932, 1054, 1630, 2388, 2449, 3478, 3497, 3755, 3772, 3795, 3796, 3797, 3798, 3799, 3800, 3801, 3802, 3803, 3804, 3805		
<code>\seq_pop_left:NN</code>	3844		
<code>\seq_put_right:Nn</code> ...	1038, 2970, 3547		
<code>\seq_reverse:N</code>	1636		
<code>\seq_set_eq:NN</code>	450, 494, 3810, 4036, 4047, 4058, 4108, 4144, 4180, 4214, 4230, 4312, 4346, 4357, 4578, 4623, 4645, 4674		
<code>\seq_set_from_clist:Nn</code>	1635, 2392, 3438		
<code>\seq_set_split:Nnn</code>	443		
<code>\seq_sort:Nn</code>	88, 3506		
<code>\seq_use:Nn</code>	5027		
<code>\setcounter</code> ..	5381, 5402, 5417, 5442, 5456		
<code>\sidefootnote</code>	132		
sort commands:			
<code>\sort_return_same:</code>	89, 93, 3513, 3518, 3592, 3612, 3633, 3648, 3662, 3687, 3722, 3737, 3753		
<code>\sort_return_swapped:</code>	89, 93, 3526, 3602, 3611, 3632, 3647, 3663, 3686, 3730, 3740, 3752		
<code>\space</code>	11		
<code>\stepcounter</code>	135		
str commands:			
<code>\str_case:nn</code>	45		
<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>	5143, 5149, 5151, 5155, 5555, 5556		
<code>\str_new:N</code>	2081		
<code>\str_set:Nn</code>	2086, 2088, 2090, 2092		
<code>\subparagraph</code>	138		
<code>\subref</code>	137		
<code>\subsection</code>	138		
<code>\subsubsection</code>	62		
<code>\subsubsections</code>	138		
<code>\subsubsubsection</code>	62		
T			
<code>\tablename</code>	140		
<code>\tag</code>	125, 133, 135		
TeX and L ^A T _E X 2 _ε commands:			
<code>\@Alph</code>	128		
<code>\@addtoreset</code>	5		
<code>\@bsphack</code>	936		
<code>\@capttype</code>	5479, 5666		
<code>\@chapapp</code>	128		
<code>\@currentcounter</code>	2-5, 64, 132, 133, 136, 27, 28, 55, 56, 57, 2433, 2434		
<code>\@currentlabel</code>	3, 125, 136		
<code>\@elt</code>	5		

<code>\@esphack</code>	986	<code>\tl_gset_eq:NN</code>	125
<code>\@ifl@t@r</code>	5	<code>\tl_head:N</code>	3646, 3659, 3671, 3673, 3683, 3685
<code>\@onlypreamble</code>	768, 782, 2921	<code>\tl_head:n</code>	1884, 1885, 1938, 1939, 1983, 1984, 1990
<code>\bbl@loaded</code>	56	<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, 4956, 4978, 5036, 5047, 5090
<code>\bbl@main@language</code>	25, 2110	<code>\tl_if_empty:nTF</code>	755, 771, 994, 1286, 1563, 1572, 1735, 2454, 2749, 2926, 3198, 5167
<code>\c@lstnumber</code>	136	<code>\tl_if_empty_p:N</code>	50, 55, 2617, 4247, 4318, 4696, 4989, 5003, 5142, 5152, 5156
<code>\c@page</code>	7	<code>\tl_if_empty_p:n</code>	1397, 1430, 2562, 2829, 3331, 3356, 3587, 3588, 3597, 3598, 3623, 3624, 3639, 3654
<code>\caption@subtyphook</code>	5667	<code>\tl_if_eq:NNTF</code>	122, 3558, 3581, 3888
<code>\hyper@link</code>	114, 123	<code>\tl_if_eq:NnTF</code>	1793, 3501, 3533, 3708, 3711, 3736, 3739, 3856, 3904, 4938, 5188
<code>\hyper@linkfile</code>	5169	<code>\tl_if_eq:nnTF</code>	1783, 1795, 1807, 1817, 1883, 1937, 1982, 3700, 5184, 5190, 5212, 5216, 5231
<code>\lst@AddToHook</code>	5616	<code>\tl_if_exist:NTF</code>	354, 364, 374, 381, 388, 399, 415, 425, 744
<code>\ltx@gobble</code>	64	<code>\tl_if_exist_p:N</code>	2616
<code>\MT@newlabel</code>	134	<code>\tl_if_novalue:NTF</code>	2504, 2552, 2650, 2816
<code>\zref@addprop</code>	63, 21, 31, 46, 61, 63, 115, 129	<code>\tl_map_break:n</code>	108
<code>\zref@default</code>	114, 4735, 4737	<code>\tl_map_tokens:Nn</code>	100
<code>\zref@extractdefault</code>	12, 123, 298, 304, 308	<code>\tl_new:N</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
<code>\zref@ifpropundefined</code>	43, 1292, 1579, 1741, 2755, 3204, 5173, 5583	<code>\tl_put_left:Nn</code>	4368, 4375, 4418, 5049, 5050, 5092, 5094, 5096, 5098
<code>\zref@ifrefcontainsprop</code>	43, 46, 1769, 1777, 1836, 1854, 1866, 1914, 1959, 3909, 4740, 4823, 4877, 5176	<code>\tl_put_right:Nn</code>	4052, 4075, 4083, 4101, 4114, 4153, 4162, 4184, 4192, 4199, 4223, 4237, 4254, 4264, 4562, 4585, 4597, 4631, 4653, 4662, 4682, 4703, 4713, 4957, 4958, 4969, 5667
<code>\zref@ifrefundefined</code>	3508, 3510, 3522, 3877, 3879, 3884, 3901, 4283, 4294, 4495, 4818, 4932	<code>\tl_reverse:N</code>	3568, 3571
<code>\zref@label</code>	64, 2446	<code>\tl_set:Nn</code>	297, 356, 737, 762, 952, 996, 1008, 1576, 1583, 1585, 1774
<code>\zref@localaddprop</code>	5481, 5536, 5597, 5668		
<code>\ZREF@mainlist</code>	21, 31, 46, 61, 63, 115, 129, 5481, 5536, 5597, 5668		
<code>\zref@newprop</code>	6, 8, 20, 22, 32, 47, 62, 110, 116, 128, 5478, 5513, 5584, 5665		
<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, 5757, 6021, 6633, 6855, 7069, 7292, 7777		
<code>\thechapter</code>	128, 130		
<code>\thelstnumber</code>	136		
<code>\thepage</code>	7, 8, 122, 125		
<code>\thesection</code>	128		
tl commands:			
<code>\c_novalue_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, 4934, 4941, 4971, 5063, 5122, 5279		
<code>\tl_const:Nn</code>	1544		
<code>\tl_gclear:N</code>	383, 427		
<code>\tl_gset:Nn</code>	376, 417, 761, 776		


```

\__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, 5319, 5319, 5351
\__zrefclever_get_rf_opt_-
seq:nnnN ..... 21, 126, 3972,
3976, 3980, 3984, 3988, 3992, 3996,
4000, 4004, 4008, 5005, 5286, 5286, 5318
\__zrefclever_get_rf_opt_tl:nnnN
. 21, 23, 46, 126, 3456, 3827, 3831,
3835, 3924, 3928, 3932, 3936, 3940,
3944, 3948, 3952, 3956, 5253, 5253, 5285
\__zrefclever_hyperlink:nnn . 123,
4391, 4749, 4790, 4827, 4889, 5165, 5165
\l__zrefclever_hyperlink_bool ..
.... 2036, 2043, 2048, 2053, 2065,
2071, 2078, 3484, 4744, 4785, 4883, 5140
\l__zrefclever_hyperref_warn_-
bool .. 2037, 2044, 2049, 2054, 2069
\__zrefclever_if_class_loaded:n
..... 130, 132
\__zrefclever_if_class_loaded:nTF
..... 5462
\__zrefclever_if_package_-
loaded:n ..... 130, 130
\__zrefclever_if_package_-
loaded:nTF .....
.... 2063, 2107, 2113, 2316, 5398,
5495, 5545, 5565, 5605, 5623, 5650, 5673
\__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, 4992
\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
\l__zrefclever_label_type_a_tl .
..... 3457, 3488, 3543,
3545, 3548, 3554, 3559, 3708, 3736,
3828, 3832, 3836, 3858, 3863, 3889,
3904, 3925, 3929, 3933, 3937, 3941,
3945, 3949, 3953, 3957, 3961, 3965,
3969, 3973, 3977, 3981, 3985, 3989,
3993, 3997, 4001, 4005, 4009, 4035, 4511
\l__zrefclever_label_type_b_tl .
..... 3488,
3556, 3560, 3711, 3739, 3859, 3866, 3890
\__zrefclever_label_type_put_-
new_right:n 87, 89, 3504, 3540, 3540
\l__zrefclever_label_types_seq .
..... 88, 3497, 3500, 3544, 3547, 3734
\l__zrefclever_labelhook_bool ..
..... 2436, 2439, 2445
\__zrefclever_labels_in_sequence:nn
... 48, 95, 124, 4297, 4498, 5182, 5182
\l__zrefclever_lang_decl_case_tl
.. 581, 962, 965, 1008, 1013, 1352,
1369, 2904, 2907, 2940, 2945, 3291, 3308
\l__zrefclever_lang_declension_-
seq ..... 581,
841, 842, 843, 857, 861, 868, 959,
960, 961, 964, 1001, 1007, 1012,
2901, 2902, 2903, 2906, 2933, 2939, 2944
\l__zrefclever_lang_gender_seq .
..... 581, 878, 879, 880, 895, 972,
973, 1022, 1037, 2914, 2915, 2954, 2969

```

<code>__zrefclever_language_if_-</code>	<code>\l__zrefclever_noabbrev_first_-</code>
declared:n 26, 742, 749, 751	bool 2023, 2026, 4964
<code>__zrefclever_language_if_-</code>	<code>\g__zrefclever_nocompat_bool . . .</code>
declared:n(TF) 25 2448, 2455, 2491
<code>__zrefclever_language_if_-</code>	<code>\l__zrefclever_nocompat_bool 65</code>
declared:nTF 316, 327, 338,	<code>\g__zrefclever_nocompat_modules_-</code>
742, 757, 773, 833, 937, 2146, 2183, 2892	seq 2449, 2459, 2462, 2483, 2492, 2493
<code>__zrefclever_language_varname:n</code>	<code>\l__zrefclever_nocompat_modules_-</code>
. 25, 319, 330,	seq 65
341, 738, 738, 741, 744, 760, 761,	<code>\l__zrefclever_nudge_comptosing_-</code>
775, 776, 777, 941, 946, 976, 978, 982	bool . . 2209, 2239, 2248, 2254, 4988
<code>\l__zrefclever_last_of_type_bool</code>	<code>\l__zrefclever_nudge_enabled_-</code>
. 94, 3755,	bool 2207, 2217,
3875, 3880, 3881, 3885, 3891, 3892, 4015	2219, 2223, 2224, 2229, 2230, 4463, 4974
<code>\l__zrefclever_lastsep_tl</code>	<code>\l__zrefclever_nudge_gender_bool</code>
3780, 3939, 4085, 4120, 4164, 4201, 4239 2211, 2240, 2250, 2255, 5002
<code>\l__zrefclever_link_star_bool . . .</code>	<code>\l__zrefclever_nudge_multitype_-</code>
. . . 3439, 3478, 4745, 4786, 4884, 5141	bool . . 2208, 2238, 2246, 2253, 4464
<code>\l__zrefclever_listsep_tl</code>	<code>\l__zrefclever_nudge_singular_-</code>
. 3780, 3935, 4116, 4194,	bool 2210, 2266, 4976
4564, 4587, 4599, 4633, 4655, 4664, 4684	<code>__zrefclever_opt_bool_get:NN . . .</code>
<code>\g__zrefclever_loaded_langfiles_-</code> 562, 572
seq 932, 940, 975, 981	<code>__zrefclever_opt_bool_get:NN(TF)</code>
<code>\l__zrefclever_main_language_tl</code> 20
25, 55, 735, 2110, 2116, 2120, 2141, 2178	<code>__zrefclever_opt_bool_get:NNTF</code>
<code>\l__zrefclever_mathtools_loaded_-</code> 562, 5322, 5327, 5332, 5337, 5342
bool 3471, 5542, 5547	<code>__zrefclever_opt_bool_gset_-</code>
<code>__zrefclever_mathtools_showonlyrefs:n</code>	false:N 19,
. 3473, 5548	528, 555, 561, 1510, 1527, 3410, 3418
<code>__zrefclever_name_default:</code>	<code>__zrefclever_opt_bool_gset_-</code>
. 4734, 4736, 4866	true:N 19,
<code>\l__zrefclever_name_format_-</code>	528, 548, 554, 1472, 1489, 3389, 3397
fallback_tl 3767,	<code>__zrefclever_opt_bool_if:N . 573, 580</code>
4967, 4971, 5036, 5085, 5097, 5099, 5117	<code>__zrefclever_opt_bool_if:N(TF) . 20</code>
<code>\l__zrefclever_name_format_tl</code>	<code>__zrefclever_opt_bool_if:N(TF) . .</code>
. 3767, 4953, 4954, 4957, 573, 906
4958, 4968, 4969, 5042, 5049, 5050,	<code>__zrefclever_opt_bool_if_set:N</code>
5058, 5066, 5076, 5093, 5094, 5107, 5127 512, 527
<code>\l__zrefclever_name_in_link_bool</code>	<code>__zrefclever_opt_bool_if_-</code>
. 116,	set:N(TF) 19
119, 3767, 4387, 4821, 5145, 5161, 5162	<code>__zrefclever_opt_bool_if_-</code>
<code>\l__zrefclever_namefont_tl</code>	set:N(TF)
3780, 3947, 4390, 4406, 4838, 4856, 4871	. 512, 564, 575, 1465, 1481, 1503, 1519
<code>\l__zrefclever_nameinlink_str</code>	<code>__zrefclever_opt_bool_set_-</code>
. 2081, 2086,	false:N 19, 528, 538, 547, 2594, 2868
2088, 2090, 2092, 5143, 5149, 5151, 5155	<code>__zrefclever_opt_bool_set_-</code>
<code>\l__zrefclever_namesep_tl</code>	true:N . 19, 528, 528, 537, 2589, 2859
. . . 3780, 3927, 4841, 4859, 4867, 4875	<code>__zrefclever_opt_bool_unset:N</code>
<code>\l__zrefclever_next_is_same_bool</code> 18, 501, 501, 511, 2599, 2877
. 94, 124,	<code>__zrefclever_opt_seq_get:NN 490, 500</code>
3773, 4492, 4525, 4542, 4548, 5206, 5244	<code>__zrefclever_opt_seq_get:NN(TF) 18</code>
<code>\l__zrefclever_next_maybe_range_-</code>	<code>__zrefclever_opt_seq_get:NNTF</code>
bool . . 94, 124, 3773, 4291, 4304, 490, 836, 873, 954, 967,
4491, 4518, 4531, 5198, 5205, 5224, 5242	2896, 2909, 5289, 5294, 5299, 5304, 5309

__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:NTF
 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:nn 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, 5038, 5053, 5072, 5081,
 5102, 5112, 5256, 5261, 5266, 5271, 5276
 __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:NTF
 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:nn
 .. 14, 348, 348, 1545, 5277, 5310, 5343
 __zrefclever_opt_varname_-
 general:nn 13,
 309, 309, 1665, 1670, 1679, 1685,
 1694, 1700, 1709, 1715, 1724, 1729,
 1749, 1755, 2507, 2511, 2555, 2569,
 2590, 2595, 2600, 5257, 5290, 5323
 __zrefclever_opt_varname_lang_-
 default:nnn 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, 5272, 5305, 5338
 __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,
 5055, 5104, 5114, 5267, 5300, 5333
 __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, 5040, 5074, 5083, 5262, 5295, 5328
 \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	1567, 1576, 1583, 1585, 1769, 1793,
__zrefclever_process_language_-		1837, 1854, 1866, 1873, 1878, 1914,
settings:	58, 59, 831, 831, 3441	1921, 1926, 1959, 1966, 1971, 3533,
__zrefclever_prop_put_non_-		3856, 3911, 3915, 4740, 4825, 4879, 5188
empty:Nnn	43, 1561, 1561, 2370, 2416	\l__zrefclever_ref_propserity_tl
__zrefclever_provide_langfile:n		3501
21, 31, 32, 85, 933, 933, 989, 2152, 3440		\l__zrefclever_ref_typeset_font_-
\l__zrefclever_range_beg_is_-		tl
first_bool	3773, 3823, 4142, 4178, 4212,	2280, 2282, 3451
4483, 4520, 4576, 4621, 4643, 4672, 4725		\l__zrefclever_refbounds_first_-
\l__zrefclever_range_beg_label_-		pb_seq
tl	94, 3773, 3816, 4105, 4118, 4157,	3795,
4166, 4196, 4227, 4241, 4251, 4476,		3983, 4059, 4109, 4181, 4232, 4313
4521, 4538, 4589, 4635, 4657, 4686, 4700		\l__zrefclever_refbounds_first_-
\l__zrefclever_range_count_int		rb_seq
94, 3773, 3821, 4067, 4131, 4481,		3795, 3987, 4215, 4347, 4676
4524, 4535, 4541, 4547, 4555, 4614, 4727		\l__zrefclever_refbounds_first_-
\l__zrefclever_range_end_ref_tl		seq
3773, 3817, 4253, 4259,		3795, 3975, 4358, 4579, 4625, 4647
4328, 4334, 4477, 4523, 4540, 4702, 4708		\l__zrefclever_refbounds_first_-
\l__zrefclever_range_same_count_-		sg_seq
int	94,	3795, 3979, 4037, 4048, 4145
3773, 3822, 4045, 4096, 4132, 4482,		\l__zrefclever_refbounds_last_-
4526, 4543, 4549, 4594, 4615, 4728		pe_seq
\l__zrefclever_rangeseq_tl	3780,	3795,
3943, 4256, 4266, 4331, 4341, 4705, 4715		4007, 4056, 4079, 4106, 4158, 4188, 4310
\l__zrefclever_rangetopair_bool		\l__zrefclever_refbounds_last_-
3780, 3971, 4292		re_seq
\l__zrefclever_ref_count_int	3758,	3795, 4011, 4260, 4268, 4335, 4343
3820, 4073, 4151, 4221, 4479, 4512,		\l__zrefclever_refbounds_last_-
4561, 4584, 4596, 4630, 4652, 4661, 4681		seq
\l__zrefclever_ref_decl_case_tl		3795, 4003, 4087, 4122, 4167, 4203
28, 845, 850, 851, 855,		\l__zrefclever_refbounds_mid_rb_-
858, 862, 866, 869, 914, 917, 919,		seq
2193, 2203, 5047, 5051, 5090, 5095, 5100		3795, 3995, 4228, 4242, 4687
__zrefclever_ref_default:		\l__zrefclever_refbounds_mid_re_-
4734, 4734, 4775, 4781, 4819, 4860, 4926		seq
\l__zrefclever_ref_gender_tl		3795, 3999, 4709, 4717
29, 882, 888,		\l__zrefclever_refbounds_mid_seq
889, 893, 896, 901, 902, 921, 927,		3795, 3991, 4119,
928, 2212, 2276, 5003, 5011, 5017, 5025		4197, 4566, 4590, 4602, 4636, 4658, 4666
\l__zrefclever_ref_language_tl		\l__zrefclever_reffont_tl
25, 28, 55, 733,		3780, 3951,
737, 834, 839, 849, 867, 876, 886,		4755, 4767, 4796, 4807, 4845, 4902, 4917
900, 909, 918, 925, 2134, 2140, 2145,		\l__zrefclever_reftype_override_-
2153, 2171, 2177, 2182, 3440, 3458,		tl
3829, 3833, 3837, 3926, 3930, 3934,		34, 44, 2352, 2355
3938, 3942, 3946, 3950, 3954, 3958,		\g__zrefclever_rf_opts_bool_-
3962, 3966, 3970, 3974, 3978, 3982,		maybe_type_specific_seq
3986, 3990, 3994, 3998, 4002, 4006,		53, 586, 1456, 2582, 2852, 3380
4010, 5007, 5019, 5030, 5056, 5105, 5115		\g__zrefclever_rf_opts_seq_-
\l__zrefclever_ref_property_tl		refbounds_seq
43, 48,		586, 1378, 2545, 2809, 3318
		\g__zrefclever_rf_opts_tl_maybe_-
		type_specific_seq
		586, 1102, 3015
		\g__zrefclever_rf_opts_tl_not_-
		type_specific_seq
		586, 1077, 2631, 2991
		\g__zrefclever_rf_opts_tl_-
		reference_seq
		586, 2497
		\g__zrefclever_rf_opts_tl_type_-
		names_seq
		586, 1339, 3278
		\g__zrefclever_rf_opts_tl_-
		typesetup_seq
		586, 2643

`\l_zrefclever_setup_language_tl`
..... [581](#), [762](#), [790](#), [795](#),
[805](#), [810](#), [821](#), [826](#), [952](#), [1004](#), [1011](#),
[1025](#), [1042](#), [1048](#), [1057](#), [1065](#), [1089](#),
[1114](#), [1123](#), [1146](#), [1152](#), [1160](#), [1167](#),
[1181](#), [1188](#), [1196](#), [1204](#), [1218](#), [1225](#),
[1233](#), [1241](#), [1255](#), [1262](#), [1270](#), [1278](#),
[1303](#), [1310](#), [1319](#), [1327](#), [1357](#), [1367](#),
[1390](#), [1406](#), [1422](#), [1439](#), [1468](#), [1475](#),
[1484](#), [1492](#), [1506](#), [1513](#), [1522](#), [1530](#),
[2895](#), [2936](#), [2943](#), [2957](#), [2974](#), [2980](#),
[3003](#), [3027](#), [3035](#), [3058](#), [3064](#), [3072](#),
[3079](#), [3093](#), [3100](#), [3108](#), [3116](#), [3130](#),
[3137](#), [3145](#), [3153](#), [3167](#), [3174](#), [3182](#),
[3190](#), [3215](#), [3222](#), [3231](#), [3239](#), [3296](#),
[3306](#), [3340](#), [3365](#), [3392](#), [3400](#), [3413](#), [3421](#)
`\l_zrefclever_setup_type_tl` ...
... [581](#), [953](#), [995](#), [996](#), [1028](#), [1049](#),
[1058](#), [1066](#), [1084](#), [1109](#), [1124](#), [1141](#),
[1161](#), [1168](#), [1176](#), [1197](#), [1205](#), [1213](#),
[1234](#), [1242](#), [1250](#), [1271](#), [1279](#), [1298](#),
[1320](#), [1328](#), [1346](#), [1358](#), [1368](#), [1385](#),
[1423](#), [1440](#), [1463](#), [1485](#), [1493](#), [1501](#),
[1523](#), [1531](#), [2626](#), [2628](#), [2655](#), [2662](#),
[2680](#), [2685](#), [2694](#), [2700](#), [2709](#), [2715](#),
[2724](#), [2730](#), [2739](#), [2744](#), [2764](#), [2771](#),
[2821](#), [2838](#), [2862](#), [2871](#), [2880](#), [2894](#),
[2927](#), [2928](#), [2960](#), [2981](#), [2998](#), [3022](#),
[3036](#), [3053](#), [3073](#), [3080](#), [3088](#), [3109](#),
[3117](#), [3125](#), [3146](#), [3154](#), [3162](#), [3183](#),
[3191](#), [3210](#), [3232](#), [3240](#), [3285](#), [3297](#),
[3307](#), [3325](#), [3366](#), [3387](#), [3401](#), [3408](#), [3422](#)
`\l_zrefclever_sort_decided_bool`
..... [3494](#), [3577](#), [3591](#),
[3601](#), [3605](#), [3617](#), [3627](#), [3642](#), [3657](#), [3681](#)
`_zrefclever_sort_default:nn` ..
..... [89](#), [3535](#), [3551](#), [3551](#)
`_zrefclever_sort_default_-`
`different_types:nn`
..... [45](#), [87](#), [92](#), [3562](#), [3694](#), [3694](#)
`_zrefclever_sort_default_same_-`
`type:nn` [87](#), [89](#), [3561](#), [3564](#), [3564](#)
`_zrefclever_sort_labels:`
..... [87-89](#), [93](#), [3449](#), [3498](#), [3498](#)
`_zrefclever_sort_page:nn`
..... [93](#), [3534](#), [3746](#), [3746](#)
`\l_zrefclever_sort_prior_a_int`
[3495](#), [3696](#), [3702](#), [3703](#), [3709](#), [3719](#), [3727](#)
`\l_zrefclever_sort_prior_b_int`
[3495](#), [3697](#), [3704](#), [3705](#), [3712](#), [3720](#), [3728](#)
`\l_zrefclever_tlastsep_tl`
..... [3780](#), [3838](#), [4457](#)
`\l_zrefclever_tlistsep_tl`
..... [3780](#), [3834](#), [4428](#)
`\l_zrefclever_tmpa_bool` [134](#), [1782](#),
[1804](#), [1826](#), [1830](#), [1880](#), [1881](#), [1892](#),
[1894](#), [1928](#), [1932](#), [1934](#), [1935](#), [1946](#),
[1948](#), [1973](#), [1977](#), [1979](#), [1980](#), [1998](#), [2000](#)
`\l_zrefclever_tmpa_int` [134](#),
[5625](#), [5629](#), [5636](#), [5637](#), [5640](#), [5642](#), [5644](#)
`\g_zrefclever_tmpa_seq` [134](#), [1393](#),
[1395](#), [1400](#), [1409](#), [1414](#), [1426](#), [1428](#),
[1433](#), [1443](#), [1448](#), [3327](#), [3329](#), [3334](#),
[3343](#), [3348](#), [3352](#), [3354](#), [3359](#), [3368](#), [3373](#)
`\l_zrefclever_tmpa_seq`
..... [134](#), [1034](#), [1038](#), [1070](#),
[2558](#), [2560](#), [2565](#), [2570](#), [2575](#), [2825](#),
[2827](#), [2832](#), [2840](#), [2845](#), [2966](#), [2970](#), [2985](#)
`\l_zrefclever_tmpa_tl`
..... [134](#), [950](#), [974](#),
[1870](#), [1884](#), [1887](#), [1888](#), [1918](#), [1929](#),
[1938](#), [1941](#), [1942](#), [1963](#), [1974](#), [1983](#),
[1993](#), [1994](#), [3459](#), [3460](#), [5552](#), [5555](#), [5556](#)
`\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](#), [4951](#), [4963](#), [5158](#)
`\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](#), [4932](#)
`\l_zrefclever_type_first_label_-`
`type_tl` [94](#), [119](#), [3761](#),
[3815](#), [4034](#), [4287](#), [4475](#), [4510](#), [4939](#),
[4982](#), [4998](#), [5006](#), [5018](#), [5024](#), [5041](#),
[5057](#), [5067](#), [5075](#), [5084](#), [5106](#), [5116](#), [5128](#)
`\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](#), [5008](#), [5010](#), [5013](#), [5028](#)

\l__zrefclever_type_name_- missing_bool	__zrefclever_typeset_refs: 93 , 95 , 96 , 3452 , 3808 , 3808
. 3767 , 4864 , 4935 , 4942 , 5064 , 5124	__zrefclever_typeset_refs_last_- of_type:
__zrefclever_type_name_setup: 21 , 23 , 116 , 4364 , 4930 , 4930 100 , 114 , 116 , 119 , 4017 , 4022 , 4022
\l__zrefclever_type_name_tl 116 , 119 , 3767 , 4401 , 4407 , 4839 , 4857 , 4872 , 4874 , 4934 , 4941 , 5045 , 5061 , 5063 , 5079 , 5088 , 5110 , 5120 , 5122 , 5142	__zrefclever_typeset_refs_not_- last_of_type:
\l__zrefclever_typeset_compress_- bool 1645 , 1648 , 4493 95 , 101 , 114 , 124 , 4019 , 4487 , 4487
\l__zrefclever_typeset_labels_- seq 93 , 3755 , 3810 , 3844 , 3846 , 3852	\l__zrefclever_typeset_sort_bool 1621 , 1624 , 3447
\l__zrefclever_typeset_last_bool 94 , 3755 , 3841 , 3842 , 3849 , 3874 , 4437 , 5157	\l__zrefclever_typeset_sort_seq 45 , 92 , 1630 , 1635 , 1636 , 1642 , 3698
\l__zrefclever_typeset_name_bool 1594 , 1601 , 1606 , 1611 , 4366 , 4382	\l__zrefclever_verbose_testing_- bool 3807 , 4433
\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 , 4956 , 4978 , 4989 , 5152 , 5156	__zrefclever_zceref:nnn 28 , 57 , 3433 , 3434
\l__zrefclever_typeset_queue_- prev_tl 94 , 3761 , 3812 , 4429 , 4471	__zrefclever_zceref:nnnn 85 , 87 , 3434
\l__zrefclever_typeset_range_- bool 1779 , 2009 , 2012 , 3448 , 4279	\l__zrefclever_zceref_labels_seq 87 , 88 , 3438 , 3469 , 3474 , 3478 , 3503 , 3506 , 3811
\l__zrefclever_typeset_ref_bool 1593 , 1600 , 1605 , 1610 , 4366 , 4372	\l__zrefclever_zceref_note_tl 2301 , 2304 , 3454 , 3461
	\l__zrefclever_zceref_with_check_- bool 2308 , 2325 , 3444 , 3465
	__zrefclever_zcsetup:n 69 , 2620 , 2621 , 2621 , 2623 , 5357 , 5369 , 5382 , 5403 , 5421 , 5431 , 5443 , 5464 , 5482 , 5500 , 5534 , 5570 , 5595 , 5607 , 5617 , 5632 , 5652 , 5675
	\l__zrefclever_zrefcheck_- available_bool 2307 , 2320 , 2332 , 2344 , 3443 , 3464