

The `zref-clever` package^{*}

Code documentation

Gustavo Barros[†]

2021-12-07

EXPERIMENTAL

Contents

1	Initial setup	2
2	Dependencies	3
3	<code>zref</code> setup	3
4	Plumbing	7
4.1	Auxiliary	7
4.2	Messages	8
4.3	Data extraction	10
4.4	Reference format	11
4.5	Languages	13
4.6	Dictionaries	17
4.7	Options	24
5	Configuration	40
5.1	<code>\zcsetup</code>	40
5.2	<code>\zcRefTypeSetup</code>	40
5.3	<code>\zcLanguageSetup</code>	42
6	User interface	46
6.1	<code>\zceref</code>	46
6.2	<code>\zcpageref</code>	47
7	Sorting	48
8	Typesetting	54

^{*}This file describes v0.1.1-alpha, released 2021-12-07.

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

9	Compatibility	80
9.1	appendix	80
9.2	appendices	81
9.3	memoir	82
9.4	KOMA	85
9.5	amsmath	86
9.6	mathtools	88
9.7	breqn	89
9.8	listings	90
9.9	enumitem	91
9.10	subcaption	91
9.11	subfig	92
10	Dictionaries	92
10.1	English	93
10.2	German	96
10.3	French	105
10.4	Portuguese	109
10.5	Spanish	114
	Index	118

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`, even though I'd have loved to have used `\bool_case_true:...`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the dictionaries (which became the default input encoding in the 2018-04-01 release). Finally, a couple of changes came with the 2021-11-15 kernel release, which are important here. First, a fix was made to the new hook management system (`ltxcmdhooks`), with implications to the hook we add to `\appendix` (by Phelype Oleinik at <https://tex.stackexchange.com/q/617905> and <https://github.com/latex3/latex2e/pull/699>). Second, the support for `\currentcounter` has been improved, including `\footnote` and `amsmath` (by Frank Mittelbach and Ulrike Fischer at <https://github.com/latex3/latex2e/issues/687>). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut at the 2021-11-15 kernel release.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
9     'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
```

```

10      \MessageBreak Loading will abort!%
11    }%
12  \endinput
13 }%

Identify the package.
14 \ProvidesExplPackage {zref-clever} {2021-12-07} {0.1.1-alpha}
15 {Clever LaTeX cross-references based on zref}

```

2 Dependencies

Required packages. Besides these, `zref-hyperref`, `zref-titleref`, and `zref-check` may also be loaded depending on user options.

```

16 \RequirePackage { zref-base }
17 \RequirePackage { zref-user }
18 \RequirePackage { zref-abspage }
19 \RequirePackage { l3keys2e }
20 \RequirePackage { ifdraft }

```

3 zref setup

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

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

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

Since the default/`thecounter` and `page` properties store the “*printed* representation” of their respective counters, for sorting and compressing purposes, we are also interested in their numeric values. So we store them in `zc@cntval` and `zc@pgval`. For this, we use `\c@<counter>`, which contains the counter’s numerical value (see ‘`texdoc source2e`’, section ‘`ltcounts.dtx`’).

```

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

```

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

Furthermore, even if it is true that most of the definitions of counters, and hence of their reset behavior, is likely to be defined in the preamble, this is not necessarily true. Users can create counters, `newtheorems` mid-document, and alter their reset behavior along the way. Was that not the case, we could just store the desired information at

`\begindocument` in a variable and retrieve it when needed. But since it is, we need to store the information with the label, with the values as current when the label is set.

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

The procedure below examines a set of counters, those in `\l__zrefclever_counter_resettters_seq`, and for each of them retrieves the set of counters it resets, as stored in `\c1@⟨counter⟩`, looking for the counter for which we are trying to set a label (`\l__zrefclever_current_counter_tl`, by default `\@currentcounter`, passed as an argument to the functions). There is one relevant caveat to this procedure: `\l__zrefclever_counter_resettters_seq` is populated by hand with the “usual suspects”, there is no way (that I know of) to ensure it is exhaustive. However, it is not that difficult to create a reasonable “usual suspects” list which, of course, should include the counters for the sectioning commands to start with, and it is easy to add more counters to this list if needed, with the option `counterresetters`. Unfortunately, not all counters are created alike, or reset alike. Some counters, even some kernel ones, get reset by other mechanisms (notably, the `enumerate` environment counters do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means). Therefore, inspecting `\c1@⟨counter⟩` cannot possibly fully account for all of the automatic counter resetting which takes place in the document. And there’s also no other “general rule” we could grab on for this, as far as I know. So we provide a way to manually tell `zref-clever` of these cases, by means of the `counterresetby` option, whose information is stored in `\l__zrefclever_counter_resetby_prop`. This manual specification has precedence over the search through `\l__zrefclever_counter_resettters_seq`, and should be handled with care, since there is no possible verification mechanism for this.

`__zrefclever_get_enclosing_counters_value:n`

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

```

\__zrefclever_get_enclosing_counters_value:n {⟨counter⟩}

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

```

```

59   {
60     { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
61     \_zrefclever_get_enclosing_counters_value:e
62     { \_zrefclever_counter_reset_by:n {#1} }
63   }
64 }

```

Both `e` and `f` expansions work for this particular recursive call. I'll stay with the `e` variant, since conceptually it is what I want (`x` itself is not expandable), and this package is anyway not compatible with older kernels for which the performance penalty of the `e` expansion would ensue (helpful comment by Enrico Gregorio, aka 'egreg' at https://tex.stackexchange.com/q/611370/#comment1529282_611385).

```

65 \cs_generate_variant:Nn \_zrefclever_get_enclosing_counters_value:n { e }

```

(End definition for `_zrefclever_get_enclosing_counters_value:n`.)

`_zrefclever_counter_reset_by:n`

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

```

    \_zrefclever_counter_reset_by:n { \langle counter \rangle }

66 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
67   {
68     \bool_if:nTF
69       { \prop_if_in_p:Nn \l__zrefclever_counter_resetby_prop {#1} }
70       { \prop_item:Nn \l__zrefclever_counter_resetby_prop {#1} }
71       {
72         \seq_map_tokens:Nn \l__zrefclever_counter_resettors_seq
73         { \_zrefclever_counter_reset_by_aux:nn {#1} }
74       }
75   }
76 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
77   {
78     \cs_if_exist:cT { c@ #2 }
79     {
80       \tl_if_empty:cF { c1@ #2 }
81       {
82         \tl_map_tokens:cn { c1@ #2 }
83         { \_zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
84       }
85     }
86   }
87 \cs_new:Npn \_zrefclever_counter_reset_by_auxi:nnn #1#2#3
88   {
89     \str_if_eq:nnT {#2} {#3}
90     { \tl_map_break:n { \seq_map_break:n {#1} } }
91   }

```

(End definition for `_zrefclever_counter_reset_by:n`.)

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

```

92 \zref@newprop { zc@enclval }
93 {

```

```

94     \__zrefclever_get_enclosing_counters_value:e
95     \l__zrefclever_current_counter_tl
96   }
97   \zref@addprop \ZREF@mainlist { zc@enclval }

```

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

```

98 \tl_new:N \g__zrefclever_page_format_tl
99 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
100 \AddToHook { shipout / before }
101 {
102   \group_begin:
103   \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
104   \tl_gset:Nx \g__zrefclever_page_format_tl { \thepage }
105   \group_end:
106 }
107 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
108 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

Still some other properties which we don’t need to handle at the data provision side, but need to cater for at the retrieval side, are the ones from the `zref-xr` module, which are added to the labels imported from external documents, and needed to construct hyperlinks to them and to distinguish them from the current document ones at sorting and compressing: `urluse`, `url` and `externaldocument`.

4 Plumbing

4.1 Auxiliary

`__zrefclever_if_package_loaded:n` Just a convenience, since sometimes we just need one of the branches, and it is particularly easy to miss the empty F branch after a long T one.

```

109 \prg_new_conditional:Npnn \__zrefclever_if_package_loaded:n #1 { T , F , TF }
110 { \IfPackageLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }
111 \prg_new_conditional:Npnn \__zrefclever_if_class_loaded:n #1 { T , F , TF }
112 { \IfClassLoadedTF {#1} { \prg_return_true: } { \prg_return_false: } }

```

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

4.2 Messages

```

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

```



```

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

```

```

217 }
218 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
219 {
220   Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:..~
221   To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
222   '\iow_char:N\zcref'.
223 }
224 \msg_new:nnn { zref-clever } { missing-hyperref }
225 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
226 \msg_new:nnn { zref-clever } { titleref-preamble-only }
227 {
228   Option~'titleref'~only~available~in~the~preamble~\msg_line_context:..~
229   Did~you~mean~'ref=title'?.
230 }
231 \msg_new:nnn { zref-clever } { option-preamble-only }
232 { Option~'#1'~only~available~in~the~preamble~\msg_line_context:.. }
233 \msg_new:nnn { zref-clever } { unknown-compat-module }
234 {
235   Unknown~compatibility~module~'#1'~given~to~option~'nocompat'.~
236   Nothing~to~do.
237 }
238 \msg_new:nnn { zref-clever } { missing-zref-check }
239 {
240   Option~'check'~requested~\msg_line_context:..~
241   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
242 }
243 \msg_new:nnn { zref-clever } { missing-type }
244 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
245 \msg_new:nnn { zref-clever } { missing-property }
246 { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:.. }
247 \msg_new:nnn { zref-clever } { missing-name }
248 { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:.. }
249 \msg_new:nnn { zref-clever } { missing-string }
250 {
251   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:..~
252   But~we~should~have:~throw~a~rock~at~the~maintainer.
253 }
254 \msg_new:nnn { zref-clever } { single-element-range }
255 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
256 \msg_new:nnn { zref-clever } { compat-package }
257 { Loaded~support~for~'#1'~package. }
258 \msg_new:nnn { zref-clever } { compat-class }
259 { Loaded~support~for~'#1'~documentclass. }

```

4.3 Data extraction

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

```

    \_zrefclever_def_extract:Nnnn {\langle tl val \rangle}
    {\langle label \rangle} {\langle prop \rangle} {\langle default \rangle}

260 \cs_new_protected:Npn \_zrefclever_def_extract:Nnnn #1#2#3#4
261 {

```

```

262 \exp_args:NNo \exp_args:NNo \tl_set:Nn #1
263 { \zref@extractdefault {#2} {#3} {#4} }
264 }
265 \cs_generate_variant:Nn \__zrefclever_def_extract:Nnnn { NVnn }

```

(End definition for __zrefclever_def_extract:Nnnn.)

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

```

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

266 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
267 {
268   \exp_args:NNo \exp_args:No
269   \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
270 }
271 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvnn , Vvn }

```

(End definition for __zrefclever_extract_unexp:nnn.)

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

```

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

272 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
273 { \zref@extractdefault {#1} {#2} {#3} }

```

(End definition for __zrefclever_extract:nnn.)

4.4 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_ref_string:nN, __zrefclever_get_ref_font:nN, and __zrefclever_type_name_setup: which are the basic functions to retrieve proper values for reference format settings. The “fallback” settings are stored in \g__zrefclever_fallback_dict_prop.

\l__zrefclever_setup_type_tl Store “current” type, language, and declension cases in different places for option and translation handling, notably in __zrefclever_provide_dictionary:n, \zcRefTypeSetup, \l__zrefclever_dict_decl_case_tl and \zcLanguageSetup. But also for translations retrieval, in __zrefclever_get_type_transl:nnnN and __zrefclever_get_default_transl:nnN.

```

\l__zrefclever_dict_language_tl
\l__zrefclever_dict_decl_case_tl
\l__zrefclever_dict_declension_seq
\l__zrefclever_dict_gender_seq

274 \tl_new:N \l__zrefclever_setup_type_tl
275 \tl_new:N \l__zrefclever_dict_language_tl
276 \tl_new:N \l__zrefclever_dict_decl_case_tl
277 \seq_new:N \l__zrefclever_dict_declension_seq
278 \seq_new:N \l__zrefclever_dict_gender_seq

```

(End definition for \l__zrefclever_setup_type_tl and others.)

f_options_necessarily_not_type_specific_seq
ever_ref_options_possibly_type_specific_seq
\c__zrefclever_ref_options_type_names_seq
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

Lists of reference format related 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.

```

279 \seq_const_from_clist:Nn
280 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
281 {
282     tpairsep ,
283     tlistsep ,
284     tlastsep ,
285     notesep ,
286 }
287 \seq_const_from_clist:Nn
288 \c__zrefclever_ref_options_possibly_type_specific_seq
289 {
290     namesep ,
291     pairsep ,
292     listsep ,
293     lastsep ,
294     rangesep ,
295     refpre ,
296     refpos ,
297 }

```

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_ref_string:nN`, but by `__zrefclever_type_name_setup:.`

```

298 \seq_const_from_clist:Nn
299 \c__zrefclever_ref_options_type_names_seq
300 {
301     Name-sg ,
302     name-sg ,
303     Name-pl ,
304     name-pl ,
305     Name-sg-ab ,
306     name-sg-ab ,
307     Name-pl-ab ,
308     name-pl-ab ,
309 }

```

`\c__zrefclever_ref_options_font_seq` are technically “possibly type-specific”, but are not “language-specific”, so we separate them.

```

310 \seq_const_from_clist:Nn
311 \c__zrefclever_ref_options_font_seq
312 {
313     namefont ,
314     reffont ,
315 }

```

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

```

316 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
317 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
318 \c__zrefclever_ref_options_possibly_type_specific_seq
319 \c__zrefclever_ref_options_type_names_seq
320 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
321 \c__zrefclever_ref_options_typesetup_seq

```

```

322 \c__zrefclever_ref_options_font_seq
323 \seq_new:N \c__zrefclever_ref_options_reference_seq
324 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
325 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
326 \c__zrefclever_ref_options_possibly_type_specific_seq
327 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
328 \c__zrefclever_ref_options_reference_seq
329 \c__zrefclever_ref_options_font_seq

```

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

4.5 Languages

`\g__zrefclever_languages_prop`

Stores the names of known languages and the mapping from “language name” to “dictionary name”. Whether or not a language or alias is known to `zref-clever` is decided by its presence in this property list. A “base language” (loose concept here, meaning just “the name we gave for the dictionary 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 “dictionary name”, in other words, it is an “alias to itself”.

```

330 \prop_new:N \g__zrefclever_languages_prop

```

(End definition for `\g__zrefclever_languages_prop`.)

`\zcDeclareLanguage`

Declare a new language for use with `zref-clever`. $\langle language \rangle$ is taken to be both the “language name” and the “dictionary name”. $[\langle options \rangle]$ receive a `k=v` set of options, with three valid options. The first, `declension`, takes the noun declension cases prefixes for $\langle language \rangle$ as a comma separated list, whose first element is taken to be the default case. The second, `gender`, receives the genders for $\langle language \rangle$ as comma separated list. The third, `allcaps`, receives no value, and indicates that for $\langle language \rangle$ all nouns must be capitalized for grammatical reasons, in which case, the `cap` option is disregarded for $\langle language \rangle$. If $\langle language \rangle$ is already known, just warn. This implies a particular restriction regarding $[\langle options \rangle]$, namely that these options, when defined by the package, cannot be redefined by the user. This is deliberate, otherwise the built-in dictionaries would become much too sensitive to this particular user input, and unnecessarily so. `\zcDeclareLanguage` is preamble only.

```

\zcDeclareLanguage [ $\langle options \rangle$ ] { $\langle language \rangle$ }

331 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
332 {
333   \group_begin:
334   \tl_if_empty:nF {#2}
335   {
336     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
337     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
338     {
339       \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
340       \prop_new:c { g__zrefclever_dict_#2_prop }
341       \tl_set:Nn \l__zrefclever_dict_language_tl {#2}
342       \keys_set:nn { zref-clever / declarelang } {#1}
343     }
344   }
345   \group_end:

```

```

346 }
347 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

\zcDeclareLanguageAlias Declare *<language alias>* to be an alias of *<aliased language>*. *<aliased language>* must be already known to zref-clever, as stored in \g__zrefclever_languages_prop. \zcDeclareLanguageAlias is preamble only.

```

\zcDeclareLanguageAlias {<language alias>} {<aliased language>}

348 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
349 {
350   \tl_if_empty:nF {#1}
351   {
352     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
353     {
354       \exp_args:NNnx
355       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
356       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
357     }
358     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
359   }
360 }
361 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

362 \keys_define:nn { zref-clever / declarelang }
363 {
364   declension .code:n =
365   {
366     \prop_gput:cnn
367     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
368     { declension } {#1}
369   } ,
370   declension .value_required:n = true ,
371   gender .code:n =
372   {
373     \prop_gput:cnn
374     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
375     { gender } {#1}
376   } ,
377   gender .value_required:n = true ,
378   allcaps .code:n =
379   {
380     \prop_gput:cnn
381     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
382     { allcaps } { true }
383   } ,
384   allcaps .value_forbidden:n = true ,
385 }

```

_zrefclever_process_language_options: Auxiliary function for _zrefclever_zceref:nnn, responsible for processing options from \zcDeclareLanguage. It is necessary to separate them from the reference options machinery 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`). Hence, we must validate these options 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.

```

386 \cs_new_protected:Npn \__zrefclever_process_language_options:
387 {
388   \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
389     { \l__zrefclever_ref_language_tl }
390     \l__zrefclever_dict_language_tl
391   {

```

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.

```

392     \exp_args:NNx \seq_set_from_clist:Nn
393       \l__zrefclever_dict_declension_seq
394     {
395       \prop_item:cn
396       {
397         g__zrefclever_dict_
398         \l__zrefclever_dict_language_tl _prop
399       }
400       { declension }
401     }
402   \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
403   {
404     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
405     {
406       \msg_warning:nxxx { zref-clever }
407       { language-no-decl-ref }
408       { \l__zrefclever_ref_language_tl }
409       { \l__zrefclever_ref_decl_case_tl }
410       \tl_clear:N \l__zrefclever_ref_decl_case_tl
411     }
412   }
413   {
414     \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
415     {
416       \seq_get_left:NN \l__zrefclever_dict_declension_seq
417       \l__zrefclever_ref_decl_case_tl
418     }
419     {
420       \seq_if_in:NVF \l__zrefclever_dict_declension_seq
421       \l__zrefclever_ref_decl_case_tl
422       {
423         \msg_warning:nxxx { zref-clever }
424         { unknown-decl-case }
425         { \l__zrefclever_ref_decl_case_tl }
426         { \l__zrefclever_ref_language_tl }
427         \seq_get_left:NN \l__zrefclever_dict_declension_seq
428         \l__zrefclever_ref_decl_case_tl

```

```

429         }
430     }
431 }

```

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.

```

432     \exp_args:NNx \seq_set_from_clist:Nn
433     \l__zrefclever_dict_gender_seq
434     {
435         \prop_item:cn
436         {
437             g__zrefclever_dict_
438             \l__zrefclever_dict_language_tl _prop
439         }
440         { gender }
441     }
442     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
443     {
444         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
445         {
446             \msg_warning:nxxx { zref-clever }
447             { language-no-gender }
448             { \l__zrefclever_ref_language_tl }
449             { g }
450             { \l__zrefclever_ref_gender_tl }
451             \tl_clear:N \l__zrefclever_ref_gender_tl
452         }
453     }
454     {
455         \tl_if_empty:NF \l__zrefclever_ref_gender_tl
456         {
457             \seq_if_in:NVF \l__zrefclever_dict_gender_seq
458             \l__zrefclever_ref_gender_tl
459             {
460                 \msg_warning:nxxx { zref-clever }
461                 { gender-not-declared }
462                 { \l__zrefclever_ref_language_tl }
463                 { \l__zrefclever_ref_gender_tl }
464                 \tl_clear:N \l__zrefclever_ref_gender_tl
465             }
466         }
467     }

```

Ensure \l__zrefclever_capitalize_bool is set to true when the language was declared with allcaps option.

```

468     \str_if_eq:eeT
469     {
470         \prop_item:cn
471         {
472             g__zrefclever_dict_
473             \l__zrefclever_dict_language_tl _prop
474         }
475         { allcaps }
476     }

```



```

477         { true }
478         { \bool_set_true:N \l__zrefclever_capitalize_bool }
479     }
480     {

```

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

```

481     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
482     {
483         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
484         { \l__zrefclever_ref_decl_case_tl }
485         { \l__zrefclever_ref_language_tl }
486         \tl_clear:N \l__zrefclever_ref_decl_case_tl
487     }
488     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
489     {
490         \msg_warning:nnxxx { zref-clever }
491         { language-no-gender }
492         { \l__zrefclever_ref_language_tl }
493         { g }
494         { \l__zrefclever_ref_gender_tl }
495         \tl_clear:N \l__zrefclever_ref_gender_tl
496     }
497 }
498 }

```

(End definition for `__zrefclever_process_language_options:.`)

4.6 Dictionaries

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

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

not in this form, but actually resemble “configuration files” of sorts, which means they are read and processed somehow else than with just `\input`. So we do the more or less the same here. It seems a reasonable way to ensure we can load dictionaries 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 dictionary files are a set of *key-value options* which are read from the file, and fed to `\keys_set:nn{zref-clever/dictionary}` by `__zrefclever_provide_dictionary:n`. And they use the same syntax and options as `\zcLanguageSetup` does. The dictionary file itself is read with `\ExplSyntaxOn` with the usual implications for white-space and catcodes.

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

Provide

`\g__zrefclever_loaded_dictionaries_seq` Used to keep track of whether a dictionary has already been loaded or not.

```

499 \seq_new:N \g__zrefclever_loaded_dictionaries_seq

(End definition for \g__zrefclever_loaded_dictionaries_seq.)

```

`\l__zrefclever_load_dict_verbose_bool` Controls whether `__zrefclever_provide_dictionary:n` fails silently or verbosely in case of unknown languages or dictionaries not found.

```

500 \bool_new:N \l__zrefclever_load_dict_verbose_bool

(End definition for \l__zrefclever_load_dict_verbose_bool.)

```

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

```

\__zrefclever_provide_dictionary:n {<language>}

501 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
502 {
503   \group_begin:
504   \@bsphack
505   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
506   \l__zrefclever_dict_language_tl
507   {
508     \seq_if_in:NVF
509     \g__zrefclever_loaded_dictionaries_seq
510     \l__zrefclever_dict_language_tl
511     {
512       \exp_args:Nx \file_get:nnNTF
513       { zref-clever- \l__zrefclever_dict_language_tl .dict }
514       { \ExplSyntaxOn }
515       \l_tmpa_tl
516       {
517         \tl_clear:N \l__zrefclever_setup_type_tl
518         \exp_args:NNx \seq_set_from_clist:Nn
519         \l__zrefclever_dict_declension_seq

```

```

520         {
521             \prop_item:cn
522             {
523                 g__zrefclever_dict_
524                 \l__zrefclever_dict_language_tl _prop
525             }
526             { declension }
527         }
528         \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
529         { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
530         {
531             \seq_get_left:NN \l__zrefclever_dict_declension_seq
532             \l__zrefclever_dict_decl_case_tl
533         }
534         \exp_args:NNx \seq_set_from_clist:Nn
535         \l__zrefclever_dict_gender_seq
536         {
537             \prop_item:cn
538             {
539                 g__zrefclever_dict_
540                 \l__zrefclever_dict_language_tl _prop
541             }
542             { gender }
543         }
544         \keys_set:nV { zref-clever / dictionary } \l_tmpa_tl
545         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
546         \l__zrefclever_dict_language_tl
547         \msg_note:nnx { zref-clever } { dict-loaded }
548         { \l__zrefclever_dict_language_tl }
549     }
550     {
551         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
552         {
553             \msg_warning:nnx { zref-clever } { dict-not-available }
554             { \l__zrefclever_dict_language_tl }
555         }
556     }

```

Even if we don't have the actual dictionary, 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, because users cannot really provide the dictionary files (well, technically they could, but we are working so they don't need to, and have better ways to do what they want). And if the users had provided some translations themselves, by means of `\zcLanguageSetup`, everything would be in place, and they could use the `lang` option multiple times, and the `dict-not-available` warning would never go away.

```

556         \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
557         \l__zrefclever_dict_language_tl
558     }
559 }
560 }
561 {
562     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
563     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
564 }
565 \@esphack

```

```

566 \group_end:
567 }
568 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }

```

(End definition for __zrefclever_provide_dictionary:n.)

__zrefclever_provide_dictionary_verbose:n Does the same as __zrefclever_provide_dictionary:n, but warns if the loading of the dictionary has failed.

```

\__zrefclever_provide_dictionary_verbose:n {<language>}

569 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
570 {
571   \group_begin:
572   \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
573   \__zrefclever_provide_dictionary:n {#1}
574   \group_end:
575 }
576 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn A couple of auxiliary functions for the of zref-clever/dictionary keys set in __zrefclever_provide_dictionary:n. They respectively “provide” (i.e. set if it value does not exist, do nothing if it already does) “type-specific” and “default” translations. Both receive <key> and <translation> as arguments, but __zrefclever_provide_dict_type_transl:nn relies on the current value of \l__zrefclever_setup_type_tl, as set by the type key.

```

\__zrefclever_provide_dict_type_transl:nn {<key>} {<translation>}
\__zrefclever_provide_dict_default_transl:nn {<key>} {<translation>}

577 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
578 {
579   \exp_args:Nnx \prop_gput_if_new:cnn
580   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
581   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
582 }
583 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
584 {
585   \prop_gput_if_new:cnn
586   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
587   { default- #1 } {#2}
588 }

```

(End definition for __zrefclever_provide_dict_type_transl:nn and __zrefclever_provide_dict_default_transl:nn.)

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

```

589 \keys_define:nn { zref-clever / dictionary }
590 {
591   type .code:n =

```

```

592     {
593         \tl_if_empty:nTF {#1}
594         { \tl_clear:N \l__zrefclever_setup_type_tl }
595         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
596     } ,
597     case .code:n =
598     {
599         \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
600         {
601             \msg_info:nxxx { zref-clever } { language-no-decl-setup }
602             { \l__zrefclever_dict_language_tl } {#1}
603         }
604         {
605             \seq_if_in:NnTF \l__zrefclever_dict_declension_seq {#1}
606             { \tl_set:Nn \l__zrefclever_dict_decl_case_tl {#1} }
607             {
608                 \msg_info:nxxx { zref-clever } { unknown-decl-case }
609                 {#1} { \l__zrefclever_dict_language_tl }
610                 \seq_get_left:NN \l__zrefclever_dict_declension_seq
611                 \l__zrefclever_dict_decl_case_tl
612             }
613         }
614     } ,
615     case .value_required:n = true ,
616     gender .code:n =
617     {
618         \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
619         {
620             \msg_info:nxxxx { zref-clever } { language-no-gender }
621             { \l__zrefclever_dict_language_tl } { gender } {#1}
622         }
623         {
624             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
625             {
626                 \msg_info:nnn { zref-clever }
627                 { option-only-type-specific } { gender }
628             }
629             {
630                 \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
631                 { \__zrefclever_provide_dict_type_transl:nn { gender } {#1} }
632                 {
633                     \msg_info:nxxx { zref-clever } { gender-not-declared }
634                     { \l__zrefclever_dict_language_tl } {#1}
635                 }
636             }
637         }
638     } ,
639     gender .value_required:n = true ,
640 }
641 \seq_map_inline:Nn
642 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
643 {
644     \keys_define:nn { zref-clever / dictionary }
645     {

```

```

646         #1 .value_required:n = true ,
647         #1 .code:n =
648         {
649             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
650             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
651             {
652                 \msg_info:nnn { zref-clever }
653                 { option-not-type-specific } {#1}
654             }
655         } ,
656     }
657 }
658 \seq_map_inline:Nn
659 \c__zrefclever_ref_options_possibly_type_specific_seq
660 {
661     \keys_define:nn { zref-clever / dictionary }
662     {
663         #1 .value_required:n = true ,
664         #1 .code:n =
665         {
666             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
667             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
668             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
669         } ,
670     }
671 }
672 \seq_map_inline:Nn
673 \c__zrefclever_ref_options_type_names_seq
674 {
675     \keys_define:nn { zref-clever / dictionary }
676     {
677         #1 .value_required:n = true ,
678         #1 .code:n =
679         {
680             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
681             {
682                 \msg_info:nnn { zref-clever }
683                 { option-only-type-specific } {#1}
684             }
685             {
686                 \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
687                 { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
688                 {
689                     \__zrefclever_provide_dict_type_transl:nn
690                     { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
691                 }
692             }
693         } ,
694     }
695 }

```

Fallback

All “strings” queried with `_zrefclever_get_ref_string:nN` – in practice, those in either `\c_zrefclever_ref_options_necessarily_not_type_specific_seq` or `\c_zrefclever_ref_options_possibly_type_specific_seq` – must have their values set for “fallback”, even if to empty ones, since this is what will be retrieved in the absence of a proper translation, which will be the case if `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”. Also “font” options – those in `\c_zrefclever_ref_options_font_seq`, and queried with `_zrefclever_get_ref_font:nN` – do not need to be provided here, since the later function sets an empty value if the option is not found.

```

696 \prop_new:N \g__zrefclever_fallback_dict_prop
697 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
698 {
699   tpairsep = {,~} ,
700   tlistsep = {,~} ,
701   tlastsep = {,~} ,
702   notesep  = {~} ,
703   namesep  = {\nobreakspace} ,
704   pairsep  = {,~} ,
705   listsep  = {,~} ,
706   lastsep  = {,~} ,
707   rangesep = {\textendash} ,
708   refpre   = {} ,
709   refpos   = {} ,
710 }
```

Get translations

`_zrefclever_get_type_transl:nnnNF` Get type-specific translation of $\langle key \rangle$ for $\langle type \rangle$ and $\langle language \rangle$, and store it in $\langle tl variable \rangle$ if found. If not found, leave the $\langle false code \rangle$ on the stream, in which case the value of $\langle tl variable \rangle$ should not be relied upon.

```

\__zrefclever_get_type_transl:nnnNF {<language>} {<type>} {<key>}
{<tl variable>} {<>false code>}

711 \prg_new_protected_conditional:Npnn
712 \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
713 {
714   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
715   \l__zrefclever_dict_language_tl
716   {
717     \prop_get:cnNTF
718     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
719     { type- #2 - #3 } #4
720     { \prg_return_true: }
721     { \prg_return_false: }
722   }
723   { \prg_return_false: }
724 }
725 \prg_generate_conditional_variant:Nnn
726 \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }
```

(End definition for `_zrefclever_get_type_transl:nnnNF`.)

`_zrefclever_get_default_transl:nnNF` Get default translation of $\langle key \rangle$ for $\langle language \rangle$, and store it in $\langle tl\ variable \rangle$ if found. If not found, leave the $\langle false\ code \rangle$ on the stream, in which case the value of $\langle tl\ variable \rangle$ should not be relied upon.

```

    \_zrefclever_get_default_transl:nnNF {\langle language \rangle} {\langle key \rangle}
      {\langle tl\ variable \rangle} {\langle false\ code \rangle}

727 \prg_new_protected_conditional:Npnn
728 \_zrefclever_get_default_transl:nnN #1#2#3 { F }
729 {
730   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
731     \l__zrefclever_dict_language_tl
732     {
733       \prop_get:cnNTF
734         { \g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
735         { default- #2 } #3
736         { \prg_return_true: }
737         { \prg_return_false: }
738       }
739     { \prg_return_false: }
740   }
741 \prg_generate_conditional_variant:Nnn
742 \_zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for `_zrefclever_get_default_transl:nnNF`.)

`_zrefclever_get_fallback_transl:nnNF` Get fallback translation of $\langle key \rangle$, and store it in $\langle tl\ variable \rangle$ if found. If not found, leave the $\langle false\ code \rangle$ on the stream, in which case the value of $\langle tl\ variable \rangle$ should not be relied upon.

```

    \_zrefclever_get_fallback_transl:nnNF {\langle key \rangle}
      {\langle tl\ variable \rangle} {\langle false\ code \rangle}

743 % {\langle key \rangle} <tl var to set>
744 \prg_new_protected_conditional:Npnn
745 \_zrefclever_get_fallback_transl:nnN #1#2 { F }
746 {
747   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
748     { #1 } #2
749     { \prg_return_true: }
750     { \prg_return_false: }
751   }

```

(End definition for `_zrefclever_get_fallback_transl:nnNF`.)

4.7 Options

Auxiliary

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

```

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

```



```

752 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
753 {
754   \tl_if_empty:nTF {#3}
755     { \prop_remove:Nn #1 {#2} }
756     { \prop_put:Nnn #1 {#2} {#3} }
757 }

```

(End definition for __zrefclever_prop_put_non_empty:Nnn.)

ref option

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

```

758 \tl_new:N \l__zrefclever_ref_property_tl
759 \keys_define:nn { zref-clever / reference }
760 {
761   ref .code:n =
762   {
763     \zref@ifpropundefined {#1}
764     {
765       \msg_warning:nnn { zref-clever } { zref-property-undefined } {#1}
766       \tl_set:Nn \l__zrefclever_ref_property_tl { default }
767     }
768     { \tl_set:Nn \l__zrefclever_ref_property_tl {#1} }
769   } ,
770   ref .initial:n = default ,
771   ref .value_required:n = true ,
772   page .meta:n = { ref = page },
773   page .value_forbidden:n = true ,
774 }

```

typeset option

```

775 \bool_new:N \l__zrefclever_typeset_ref_bool
776 \bool_new:N \l__zrefclever_typeset_name_bool
777 \keys_define:nn { zref-clever / reference }
778 {
779   typeset .choice: ,
780   typeset / both .code:n =
781   {
782     \bool_set_true:N \l__zrefclever_typeset_ref_bool
783     \bool_set_true:N \l__zrefclever_typeset_name_bool
784   } ,
785   typeset / ref .code:n =
786   {
787     \bool_set_true:N \l__zrefclever_typeset_ref_bool
788     \bool_set_false:N \l__zrefclever_typeset_name_bool

```

```

789     } ,
790     typeset / name .code:n =
791     {
792         \bool_set_false:N \l__zrefclever_typeset_ref_bool
793         \bool_set_true:N \l__zrefclever_typeset_name_bool
794     } ,
795     typeset .initial:n = both ,
796     typeset .value_required:n = true ,
797
798     noname .meta:n = { typeset = ref } ,
799     noname .value_forbidden:n = true ,
800     noref .meta:n = { typeset = name } ,
801     noref .value_forbidden:n = true ,
802 }

```

sort option

```

803 \bool_new:N \l__zrefclever_typeset_sort_bool
804 \keys_define:nn { zref-clever / reference }
805 {
806     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
807     sort .initial:n = true ,
808     sort .default:n = true ,
809     nosort .meta:n = { sort = false } ,
810     nosort .value_forbidden:n = true ,
811 }

```

typesort option

\l__zrefclever_typesort_seq is stored reversed, since the sort priorities are computed in the negative range in __zrefclever_sort_default_different_types:nn, so that we can implicitly rely on ‘0’ being the “last value”, and spare creating an integer variable using \seq_map_indexed_inline:Nn.

```

812 \seq_new:N \l__zrefclever_typesort_seq
813 \keys_define:nn { zref-clever / reference }
814 {
815     typesort .code:n =
816     {
817         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
818         \seq_reverse:N \l__zrefclever_typesort_seq
819     } ,
820     typesort .initial:n =
821     { part , chapter , section , paragraph } ,
822     typesort .value_required:n = true ,
823     notypesort .code:n =
824     { \seq_clear:N \l__zrefclever_typesort_seq } ,
825     notypesort .value_forbidden:n = true ,
826 }

```

comp option

```

827 \bool_new:N \l__zrefclever_typeset_compress_bool
828 \keys_define:nn { zref-clever / reference }
829 {
830     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,

```

```

831     comp .initial:n = true ,
832     comp .default:n = true ,
833     nocomp .meta:n = { comp = false },
834     nocomp .value_forbidden:n = true ,
835 }

```

range option

```

836 \bool_new:N \l__zrefclever_typeset_range_bool
837 \keys_define:nn { zref-clever / reference }
838 {
839     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
840     range .initial:n = false ,
841     range .default:n = true ,
842 }

```

cap and capfirst options

```

843 \bool_new:N \l__zrefclever_capitalize_bool
844 \bool_new:N \l__zrefclever_capitalize_first_bool
845 \keys_define:nn { zref-clever / reference }
846 {
847     cap .bool_set:N = \l__zrefclever_capitalize_bool ,
848     cap .initial:n = false ,
849     cap .default:n = true ,
850     nocap .meta:n = { cap = false },
851     nocap .value_forbidden:n = true ,
852
853     capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
854     capfirst .initial:n = false ,
855     capfirst .default:n = true ,
856 }

```

abbrev and noabbrevfirst options

```

857 \bool_new:N \l__zrefclever_abbrev_bool
858 \bool_new:N \l__zrefclever_noabbrev_first_bool
859 \keys_define:nn { zref-clever / reference }
860 {
861     abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
862     abbrev .initial:n = false ,
863     abbrev .default:n = true ,
864     noabbrev .meta:n = { abbrev = false },
865     noabbrev .value_forbidden:n = true ,
866
867     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
868     noabbrevfirst .initial:n = false ,
869     noabbrevfirst .default:n = true ,
870 }

```

S option

```

871 \keys_define:nn { zref-clever / reference }
872 {
873     S .meta:n =
874     { capfirst = true , noabbrevfirst = true },
875     S .value_forbidden:n = true ,
876 }

```

hyperref option

```
877 \bool_new:N \l__zrefclever_use_hyperref_bool
878 \bool_new:N \l__zrefclever_warn_hyperref_bool
879 \keys_define:nn { zref-clever / reference }
880 {
881   hyperref .choice: ,
882   hyperref / auto .code:n =
883   {
884     \bool_set_true:N \l__zrefclever_use_hyperref_bool
885     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
886   } ,
887   hyperref / true .code:n =
888   {
889     \bool_set_true:N \l__zrefclever_use_hyperref_bool
890     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
891   } ,
892   hyperref / false .code:n =
893   {
894     \bool_set_false:N \l__zrefclever_use_hyperref_bool
895     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
896   } ,
897   hyperref .initial:n = auto ,
898   hyperref .default:n = auto
899 }
900 \AddToHook { begindocument }
901 {
902   \__zrefclever_if_package_loaded:nTF { hyperref }
903   {
904     \bool_if:NT \l__zrefclever_use_hyperref_bool
905     { \RequirePackage { zref-hyperref } }
906   }
907   {
908     \bool_if:NT \l__zrefclever_warn_hyperref_bool
909     { \msg_warning:nn { zref-clever } { missing-hyperref } }
910     \bool_set_false:N \l__zrefclever_use_hyperref_bool
911   }
912   \keys_define:nn { zref-clever / reference }
913   {
914     hyperref .code:n =
915     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
916   }
917 }
```

nameinlink option

```
918 \str_new:N \l__zrefclever_nameinlink_str
919 \keys_define:nn { zref-clever / reference }
920 {
921   nameinlink .choice: ,
922   nameinlink / true .code:n =
923   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
924   nameinlink / false .code:n =
925   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
926   nameinlink / single .code:n =
```

```

927     { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
928     nameinlink / tsingle .code:n =
929     { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
930     nameinlink .initial:n = tsingle ,
931     nameinlink .default:n = true ,
932 }

```

preposinlink option

```

933 \bool_new:N \l__zrefclever_preposinlink_bool
934 \keys_define:nn { zref-clever / reference }
935 {
936     preposinlink .bool_set:N = \l__zrefclever_preposinlink_bool ,
937     preposinlink .initial:n = false ,
938     preposinlink .default:n = true ,
939 }

```

lang option

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

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

```

940 \tl_new:N \l__zrefclever_ref_language_tl
941 \tl_new:N \l__zrefclever_current_language_tl
942 \tl_new:N \l__zrefclever_main_language_tl
943 \AddToHook { begindocument }
944 {
945     \__zrefclever_if_package_loaded:nTF { babel }
946     {
947         \tl_set:Nn \l__zrefclever_current_language_tl { \language }
948         \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }

```

```

949     }
950   {
951     \__zrefclever_if_package_loaded:nTF { polyglossia }
952     {
953       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
954       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
955     }
956     {
957       \tl_set:Nn \l__zrefclever_current_language_tl { english }
958       \tl_set:Nn \l__zrefclever_main_language_tl { english }
959     }
960   }

```

Provide default value for `\l__zrefclever_ref_language_tl` corresponding to option **current**, but do so outside of the `l3keys` machinery (that is, instead of using `.initial:n`), so that we are able to distinguish when the user actually gave the option, in which case the dictionary loading is done verbosely, from when we are setting the default value (here), in which case the dictionary loading is done silently.

```

961     \tl_set:Nn \l__zrefclever_ref_language_tl
962     { \l__zrefclever_current_language_tl }
963   }
964 \keys_define:nn { zref-clever / reference }
965 {
966   lang .code:n =
967   {
968     \AddToHook { begindocument }
969     {
970       \str_case:nnF {#1}
971       {
972         { current }
973         {
974           \tl_set:Nn \l__zrefclever_ref_language_tl
975           { \l__zrefclever_current_language_tl }
976           \__zrefclever_provide_dictionary_verbosely:x
977           { \l__zrefclever_ref_language_tl }
978         }
979
980         { main }
981         {
982           \tl_set:Nn \l__zrefclever_ref_language_tl
983           { \l__zrefclever_main_language_tl }
984           \__zrefclever_provide_dictionary_verbosely:x
985           { \l__zrefclever_ref_language_tl }
986         }
987       }
988     }
989     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
990     { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
991     {
992       \msg_warning:nnn { zref-clever }
993       { unknown-language-opt } {#1}
994       \tl_set:Nn \l__zrefclever_ref_language_tl
995       { \l__zrefclever_current_language_tl }
996     }

```

```

997         \__zrefclever_provide_dictionary_verbose:x
998         { \l__zrefclever_ref_language_tl }
999     }
1000 }
1001 } ,
1002 lang .value_required:n = true ,
1003 }
1004 \AddToHook { begindocument / before }
1005 {
1006     \AddToHook { begindocument }
1007     {

```

If any `lang` option has been given by the user, the corresponding language is already loaded, otherwise, ensure the default one (`current`) gets loaded early, but not verbosely.

```

1008     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body. Also, drop the verbose dictionary loading in the document body, as it can become intrusive depending on the use case, and does not provide much “juice” anyway: in `\zcref` missing names warnings will already ensue.

```

1009     \keys_define:nn { zref-clever / reference }
1010     {
1011         lang .code:n =
1012         {
1013             \str_case:nnF {#1}
1014             {
1015                 { current }
1016                 {
1017                     \tl_set:Nn \l__zrefclever_ref_language_tl
1018                     { \l__zrefclever_current_language_tl }
1019                     \__zrefclever_provide_dictionary:x
1020                     { \l__zrefclever_ref_language_tl }
1021                 }
1022             }
1023             { main }
1024             {
1025                 \tl_set:Nn \l__zrefclever_ref_language_tl
1026                 { \l__zrefclever_main_language_tl }
1027                 \__zrefclever_provide_dictionary:x
1028                 { \l__zrefclever_ref_language_tl }
1029             }
1030         }
1031     }
1032     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1033     { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
1034     {
1035         \msg_warning:nnn { zref-clever }
1036         { unknown-language-opt } {#1}
1037         \tl_set:Nn \l__zrefclever_ref_language_tl
1038         { \l__zrefclever_current_language_tl }
1039     }
1040     \__zrefclever_provide_dictionary:x
1041     { \l__zrefclever_ref_language_tl }
1042 }

```

```

1043         } ,
1044         lang .value_required:n = true ,
1045     }
1046 }
1047 }

```

d option

For setting the declension case. Short for convenience and for not polluting the markup too much given that, for languages that need it, it may get to be used frequently.

@samcarter and Alan Munn provided useful comments about declension on the TeX.SX chat. Also, Florent Rougon's efforts in this area, with the xcref package (<https://github.com/frougon/xcref>), have been an insightful source to frame the problem in general terms.

```

1048 \tl_new:N \l__zrefclever_ref_decl_case_tl
1049 \keys_define:nn { zref-clever / reference }
1050 {
1051     d .code:n =
1052     { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1053 }
1054 \AddToHook { begindocument }
1055 {
1056     \keys_define:nn { zref-clever / reference }
1057     {

```

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

```

1058         d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1059         d .value_required:n = true ,
1060     }
1061 }

```

nudge & co. options

```

1062 \bool_new:N \l__zrefclever_nudge_enabled_bool
1063 \bool_new:N \l__zrefclever_nudge_multitype_bool
1064 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1065 \bool_new:N \l__zrefclever_nudge_singular_bool
1066 \bool_new:N \l__zrefclever_nudge_gender_bool
1067 \tl_new:N \l__zrefclever_ref_gender_tl
1068 \keys_define:nn { zref-clever / reference }
1069 {
1070     nudge .choice: ,
1071     nudge / true .code:n =
1072     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1073     nudge / false .code:n =
1074     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1075     nudge / ifdraft .code:n =
1076     {
1077         \ifdraft
1078         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1079         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1080     } ,

```



```

1081 nudge / ifffinal .code:n =
1082 {
1083     \ifoptionfinal
1084     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1085     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1086 },
1087 nudge .initial:n = false ,
1088 nudge .default:n = true ,
1089 nonnudge .meta:n = { nudge = false } ,
1090 nonnudge .value_forbidden:n = true ,
1091 nudgeif .code:n =
1092 {
1093     \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1094     \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1095     \bool_set_false:N \l__zrefclever_nudge_gender_bool
1096     \clist_map_inline:nn {#1}
1097     {
1098         \str_case:nnF {##1}
1099         {
1100             { multitype }
1101             { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }
1102             { comptosing }
1103             { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1104             { gender }
1105             { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1106             { all }
1107             {
1108                 \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1109                 \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1110                 \bool_set_true:N \l__zrefclever_nudge_gender_bool
1111             }
1112         }
1113     }
1114     \msg_warning:nnn { zref-clever }
1115     { nudgeif-unknown-value } {##1}
1116 }
1117 },
1118 nudgeif .value_required:n = true ,
1119 nudgeif .initial:n = all ,
1120 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1121 sg .initial:n = false ,
1122 sg .default:n = true ,
1123 g .code:n =
1124 { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1125 }
1126 \AddToHook { begindocument }
1127 {
1128     \keys_define:nn { zref-clever / reference }
1129     {
1130

```

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

```

1131     g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1132     g .value_required:n = true ,

```

```

1133     }
1134 }

```

font option

`font` can't be used as a package option, since the options get expanded by L^AT_EX before being passed to the package (see <https://tex.stackexchange.com/a/489570>). It can be set in `\zcref` and, for global settings, with `\zcsetup`. Note that, technically, the “raw” options are already available as `\@raw@opt@<package>.sty` (helpful comment by David Carlisle at <https://tex.stackexchange.com/a/618439>).

```

1135 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1136 \keys_define:nn { zref-clever / reference }
1137 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

1138 \keys_define:nn { zref-clever / reference }
1139 {
1140     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1141     titleref .value_forbidden:n = true ,
1142 }
1143 \AddToHook { begindocument }
1144 {
1145     \keys_define:nn { zref-clever / reference }
1146     {
1147         titleref .code:n =
1148             { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1149     }
1150 }

```

note option

```

1151 \tl_new:N \l__zrefclever_zcref_note_tl
1152 \keys_define:nn { zref-clever / reference }
1153 {
1154     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
1155     note .value_required:n = true ,
1156 }

```

check option

Integration with `zref-check`.

```

1157 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1158 \bool_new:N \l__zrefclever_zcref_with_check_bool
1159 \keys_define:nn { zref-clever / reference }
1160 {
1161     check .code:n = { \RequirePackage { zref-check } } ,
1162     check .value_forbidden:n = true ,
1163 }
1164 \AddToHook { begindocument }
1165 {
1166     \__zrefclever_if_package_loaded:nTF { zref-check }
1167     {
1168         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1169         \keys_define:nn { zref-clever / reference }
1170         {

```

```

1171         check .code:n =
1172             {
1173                 \bool_set_true:N \l__zrefclever_zceref_with_check_bool
1174                 \keys_set:nn { zref-check / zcheck } {#1}
1175             } ,
1176         check .value_required:n = true ,
1177     }
1178 }
1179 {
1180     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1181     \keys_define:nn { zref-clever / reference }
1182     {
1183         check .value_forbidden:n = false ,
1184         check .code:n =
1185             { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1186     }
1187 }
1188 }

```

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`.

```

1189 \prop_new:N \l__zrefclever_counter_type_prop
1190 \keys_define:nn { zref-clever / label }
1191 {
1192     countertype .code:n =
1193     {
1194         \keyval_parse:nnn
1195         {
1196             \msg_warning:nnnn { zref-clever }
1197             { key-requires-value } { countertype }
1198         }
1199         {
1200             \__zrefclever_prop_put_non_empty:Nnn
1201             \l__zrefclever_counter_type_prop
1202         }
1203         {#1}
1204     } ,
1205     countertype .value_required:n = true ,
1206     countertype .initial:n =
1207     {
1208         subsection      = section ,
1209         subsubsection   = section ,
1210         subparagraph    = paragraph ,
1211         enumi           = item ,
1212         enumii          = item ,
1213         enumiii         = item ,
1214         enumiv          = item ,
1215         mpfootnote      = footnote ,
1216     } ,

```

1217 }

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

counterresetters option

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

```

1218 \seq_new:N \l__zrefclever_counter_resetters_seq
1219 \keys_define:nn { zref-clever / label }
1220 {
1221   counterresetters .code:n =
1222   {
1223     \clist_map_inline:nn {#1}
1224     {
1225       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1226       {
1227         \seq_put_right:Nn
1228         \l__zrefclever_counter_resetters_seq {##1}
1229       }
1230     }
1231   } ,
1232   counterresetters .initial:n =
1233   {
1234     part ,
1235     chapter ,
1236     section ,
1237     subsection ,
1238     subsubsection ,
1239     paragraph ,
1240     subparagraph ,
1241   } ,
1242   counterresetters .value_required:n = true ,
1243 }
```

counterresetby option

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

```
1244 \prop_new:N \l__zrefclever_counter_resetby_prop
1245 \keys_define:nn { zref-clever / label }
1246 {
1247   counterresetby .code:n =
1248   {
1249     \keyval_parse:nnn
1250     {
1251       \msg_warning:nnn { zref-clever }
1252       { key-requires-value } { counterresetby }
1253     }
1254     {
1255       \__zrefclever_prop_put_non_empty:Nnn
1256       \l__zrefclever_counter_resetby_prop
1257     }
1258     {#1}
1259   } ,
1260   counterresetby .value_required:n = true ,
1261   counterresetby .initial:n =
1262   {
```

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.

```
1263   enumii = enumi ,
1264   enumiii = enumii ,
1265   enumiv = enumiii ,
1266 } ,
1267 }
```

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.

```
1268 \tl_new:N \l__zrefclever_current_counter_tl
1269 \keys_define:nn { zref-clever / label }
1270 {
1271   currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1272   currentcounter .value_required:n = true ,
1273   currentcounter .initial:n = \@currentcounter ,
1274 }
```

nocompat option

```

1275 \bool_new:N \g__zrefclever_nocompat_bool
1276 \seq_new:N \g__zrefclever_nocompat_modules_seq
1277 \keys_define:nn { zref-clever / reference }
1278 {
1279   nocompat .code:n =
1280   {
1281     \tl_if_empty:nTF {#1}
1282     { \bool_gset_true:N \g__zrefclever_nocompat_bool }
1283     {
1284       \clist_map_inline:nn {#1}
1285       {
1286         \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
1287         {
1288           \seq_gput_right:Nn
1289             \g__zrefclever_nocompat_modules_seq {##1}
1290         }
1291       }
1292     }
1293   },
1294 }
1295 \AddToHook { begindocument }
1296 {
1297   \keys_define:nn { zref-clever / reference }
1298   {
1299     nocompat .code:n =
1300     {
1301       \msg_warning:nnn { zref-clever }
1302       { option-preamble-only } { nocompat }
1303     }
1304   }
1305 }
1306 \AtEndOfPackage
1307 {
1308   \AddToHook { begindocument }
1309   {
1310     \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
1311     { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
1312   }
1313 }

```

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

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

```

1314 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
1315 {
1316   \AddToHook { begindocument }
1317   {
1318     \bool_if:NF \g__zrefclever_nocompat_bool
1319     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
1320     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
1321   }
1322 }

```

(End definition for __zrefclever_compat_module:nn.)

Reference options

This is a set of options related to reference typesetting which receive equal treatment and, hence, are handled in batch. Since we are dealing with options to be passed to `\zcref` or to `\zcsetup` or at load time, only “not necessarily type-specific” options are pertinent here. However, they *may* either be type-specific or language-specific, and thus must be stored in a property list, `\l__zrefclever_ref_options_prop`, in order to be retrieved from the option *name* by `__zrefclever_get_ref_string:nN` and `__zrefclever_get_ref_font:nN` according to context and precedence rules.

The keys are set so that any value, including an empty one, is added to `\l__zrefclever_ref_options_prop`, while a key with *no value* removes the property from the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.2.

```

1323 \prop_new:N \l__zrefclever_ref_options_prop
1324 \seq_map_inline:Nn
1325   \c__zrefclever_ref_options_reference_seq
1326   {
1327     \keys_define:nn { zref-clever / reference }
1328     {
1329       #1 .default:V = \c_novalue_tl ,
1330       #1 .code:n =
1331       {
1332         \tl_if_novalue:nTF {##1}
1333         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1334         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1335       } ,
1336     }
1337 }

```

Package options

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

```

1338 \keys_define:nn { }
1339 {
1340   zref-clever / zcsetup .inherit:n =
1341   {

```

```

1342         zref-clever / label ,
1343         zref-clever / reference ,
1344     }
1345 }

```

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

```

1346 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 \zcsetup

\zcsetup Provide \zcsetup.

```

\zcsetup{<options>}

```

```

1347 \NewDocumentCommand \zcsetup { m }
1348 { \__zrefclever_zcsetup:n {#1} }

```

(End definition for \zcsetup.)

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

```

\__zrefclever_zcsetup:n{<options>}

```

```

1349 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
1350 { \keys_set:nn { zref-clever / zcsetup } {#1} }
1351 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }

```

(End definition for __zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

\zcRefTypeSetup is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any translation, hence they override any language-specific setting, either done at \zcLanguageSetup or by the package’s dictionaries. 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>}

1352 \NewDocumentCommand \zcRefTypeSetup { m m }
1353 {
1354     \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1355     { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1356     \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1357     \keys_set:nn { zref-clever / typesetup } {#2}
1358 }

```


(End definition for `\zcRefTypeSetup`.)

Inside `\zcRefTypeSetup` any of the options *can* receive empty values, and those values, if they exist in the property list, will override translations, regardless of their emptiness. In principle, we could live with the situation of, once a setting has been made in `\l__zrefclever_type_<type>options_prop` or in `\l__zrefclever_ref_options_prop` it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the translations at any given point. So both in `\zcRefTypeSetup` and in setting reference options (see Section 4.7), 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 at `\keys_set:nn` by means of the `.default:V` property of the key in `\keys_define:nn`. 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>.

```

1359 \seq_map_inline:Nn
1360   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1361   {
1362     \keys_define:nn { zref-clever / typesetup }
1363     {
1364       #1 .code:n =
1365       {
1366         \msg_warning:nnn { zref-clever }
1367           { option-not-type-specific } {#1}
1368       } ,
1369     }
1370   }

1371 \seq_map_inline:Nn
1372   \c__zrefclever_ref_options_typesetup_seq
1373   {
1374     \keys_define:nn { zref-clever / typesetup }
1375     {
1376       #1 .default:V = \c_novalue_tl ,
1377       #1 .code:n =
1378       {
1379         \tl_if_novalue:nTF {##1}
1380         {
1381           \prop_remove:cn
1382           {
1383             l__zrefclever_type_
1384             \l__zrefclever_setup_type_tl_options_prop
1385           }
1386           {#1}
1387         }
1388         {
1389           \prop_put:cnn
1390           {
1391             l__zrefclever_type_
1392             \l__zrefclever_setup_type_tl_options_prop
1393           }
1394           {#1} {##1}

```

```

1395         }
1396     } ,
1397 }
1398 }

```

5.3 \zcLanguageSetup

\zcLanguageSetup is the main user interface for “language-specific” reference formatting, be it “type-specific” or not. The difference between the two cases is captured by the `type` key, which works as a sort of a “switch”. Inside the `<options>` argument of \zcLanguageSetup, any options made before the first `type` key declare “default” (non type-specific) translations. When the `type` key is given with a value, the options following it will set “type-specific” translations for that type. The current type can be switched off by an empty `type` key. \zcLanguageSetup is preamble only.

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}

1399 \NewDocumentCommand \zcLanguageSetup { m m }
1400 {
1401   \group_begin:
1402   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1403   \l__zrefclever_dict_language_tl
1404   {
1405     \tl_clear:N \l__zrefclever_setup_type_tl
1406     \exp_args:NNx \seq_set_from_clist:Nn
1407     \l__zrefclever_dict_declension_seq
1408     {
1409       \prop_item:cn
1410       {
1411         g__zrefclever_dict_
1412         \l__zrefclever_dict_language_tl _prop
1413       }
1414       { declension }
1415     }
1416     \seq_if_empty:NTF \l__zrefclever_dict_declension_seq
1417     { \tl_clear:N \l__zrefclever_dict_decl_case_tl }
1418     {
1419       \seq_get_left:NN \l__zrefclever_dict_declension_seq
1420       \l__zrefclever_dict_decl_case_tl
1421     }
1422     \exp_args:NNx \seq_set_from_clist:Nn
1423     \l__zrefclever_dict_gender_seq
1424     {
1425       \prop_item:cn
1426       {
1427         g__zrefclever_dict_
1428         \l__zrefclever_dict_language_tl _prop
1429       }
1430       { gender }
1431     }
1432     \keys_set:nn { zref-clever / langsetup } {#2}
1433   }
1434   { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1435   \group_end:

```

```

1436 }
1437 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

_zrefclever_declare_type_transl:nnnn A couple of auxiliary functions for the of zref-clever/translation keys set in
_zrefclever_declare_default_transl:nnn \zcLanguageSetup. They respectively declare (unconditionally set) “type-specific” and
“default” translations.

```

      \_zrefclever_declare_type_transl:nnnn {\langle language\rangle} {\langle type\rangle}
      {\langle key\rangle} {\langle translation\rangle}
      \_zrefclever_declare_default_transl:nnn {\langle language\rangle}
      {\langle key\rangle} {\langle translation\rangle}

1438 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
1439 {
1440   \prop_gput:cnn { g\_zrefclever_dict_ #1 _prop }
1441   { type- #2 - #3 } {#4}
1442 }
1443 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn , VVxn }
1444 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
1445 {
1446   \prop_gput:cnn { g\_zrefclever_dict_ #1 _prop }
1447   { default- #2 } {#3}
1448 }
1449 \cs_generate_variant:Nn \_zrefclever_declare_default_transl:nnn { Vnn }

```

(End definition for _zrefclever_declare_type_transl:nnnn and _zrefclever_declare_default_transl:nnn.)

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

```

1450 \keys_define:nn { zref-clever / langsetup }
1451 {
1452   type .code:n =
1453   {
1454     \tl_if_empty:NTF {#1}
1455     { \tl_clear:N \l\_zrefclever_setup_type_tl }
1456     { \tl_set:Nn \l\_zrefclever_setup_type_tl {#1} }
1457   } ,
1458   case .code:n =
1459   {
1460     \seq_if_empty:NTF \l\_zrefclever_dict_declension_seq
1461     {
1462       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
1463       { \l\_zrefclever_dict_language_tl } {#1}
1464     }
1465     {
1466       \seq_if_in:NnTF \l\_zrefclever_dict_declension_seq {#1}
1467       { \tl_set:Nn \l\_zrefclever_dict_decl_case_tl {#1} }
1468       {
1469         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
1470         {#1} { \l\_zrefclever_dict_language_tl }
1471         \seq_get_left:NN \l\_zrefclever_dict_declension_seq
1472         \l\_zrefclever_dict_decl_case_tl
1473       }

```

```

1474     }
1475   } ,
1476   case .value_required:n = true ,
1477   gender .code:n =
1478   {
1479     \seq_if_empty:NTF \l__zrefclever_dict_gender_seq
1480     {
1481       \msg_warning:nnxxx { zref-clever } { language-no-gender }
1482       { \l__zrefclever_dict_language_tl } { gender } {#1}
1483     }
1484     {
1485       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1486       {
1487         \msg_warning:nnn { zref-clever }
1488         { option-only-type-specific } { gender }
1489       }
1490       {
1491         \seq_if_in:NnTF \l__zrefclever_dict_gender_seq {#1}
1492         {
1493           \__zrefclever_declare_type_transl:VWnn
1494           \l__zrefclever_dict_language_tl
1495           \l__zrefclever_setup_type_tl
1496           { gender } {#1}
1497         }
1498         {
1499           \msg_warning:nnxx { zref-clever } { gender-not-declared }
1500           { \l__zrefclever_dict_language_tl } {#1}
1501         }
1502       }
1503     }
1504   } ,
1505   gender .value_required:n = true ,
1506 }
1507 \seq_map_inline:Nn
1508 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1509 {
1510   \keys_define:nn { zref-clever / langsetup }
1511   {
1512     #1 .value_required:n = true ,
1513     #1 .code:n =
1514     {
1515       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1516       {
1517         \__zrefclever_declare_default_transl:Vnn
1518         \l__zrefclever_dict_language_tl
1519         {#1} {##1}
1520       }
1521       {
1522         \msg_warning:nnn { zref-clever }
1523         { option-not-type-specific } {#1}
1524       }
1525     } ,
1526   }
1527 }

```

```

1528 \seq_map_inline:Nn
1529 \c__zrefclever_ref_options_possibly_type_specific_seq
1530 {
1531   \keys_define:nn { zref-clever / langsetup }
1532   {
1533     #1 .value_required:n = true ,
1534     #1 .code:n =
1535     {
1536       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1537       {
1538         \__zrefclever_declare_default_transl:Vnn
1539         \l__zrefclever_dict_language_tl
1540         {#1} {##1}
1541       }
1542       {
1543         \__zrefclever_declare_type_transl:VVnn
1544         \l__zrefclever_dict_language_tl
1545         \l__zrefclever_setup_type_tl
1546         {#1} {##1}
1547       }
1548     } ,
1549   }
1550 }
1551 \seq_map_inline:Nn
1552 \c__zrefclever_ref_options_type_names_seq
1553 {
1554   \keys_define:nn { zref-clever / langsetup }
1555   {
1556     #1 .value_required:n = true ,
1557     #1 .code:n =
1558     {
1559       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1560       {
1561         \msg_warning:nnn { zref-clever }
1562         { option-only-type-specific } {#1}
1563       }
1564       {
1565         \tl_if_empty:NTF \l__zrefclever_dict_decl_case_tl
1566         {
1567           \__zrefclever_declare_type_transl:VVnn
1568           \l__zrefclever_dict_language_tl
1569           \l__zrefclever_setup_type_tl
1570           {#1} {##1}
1571         }
1572         {
1573           \__zrefclever_declare_type_transl:VVxn
1574           \l__zrefclever_dict_language_tl
1575           \l__zrefclever_setup_type_tl
1576           { \l__zrefclever_dict_decl_case_tl - #1 } {##1}
1577         }
1578       }
1579     } ,
1580   }
1581 }

```

6 User interface

6.1 `\zcref`

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

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

```
1582 \NewDocumentCommand \zcref { s O { } m }
1583 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }
```

(End definition for `\zcref`.)

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

```
\__zrefclever_zcref:nnnn {<labels>} {<*>} {<options>}
```

```
1584 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1585 {
1586   \group_begin:
```

Set options.

```
1587   \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```
1588   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1589   \bool_set:Nn \l__zrefclever_link_star_bool {#2}
```

Ensure dictionary 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_dictionary:x` does nothing if the dictionary is already loaded.

```
1590   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Process `\zcDeclareLanguage` options.

```
1591   \__zrefclever_process_language_options:
```

Integration with `zref-check`.

```
1592   \bool_lazy_and:nnT
1593   { \l__zrefclever_zrefcheck_available_bool }
1594   { \l__zrefclever_zcref_with_check_bool }
1595   { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1596   \bool_lazy_or:nnT
1597   { \l__zrefclever_typeset_sort_bool }
1598   { \l__zrefclever_typeset_range_bool }
1599   { \__zrefclever_sort_labels: }
```

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

```
1600   \group_begin:
1601   \l__zrefclever_ref_typeset_font_tl
1602   \__zrefclever_typeset_refs:
1603   \group_end:
```

Typeset note.

```

1604     \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1605     {
1606         \__zrefclever_get_ref_string:nN { noteseq } \l_tmpa_tl
1607         \l_tmpa_tl
1608         \l__zrefclever_zcref_note_tl
1609     }

```

Integration with zref-check.

```

1610     \bool_lazy_and:nnT
1611     { \l__zrefclever_zrefcheck_available_bool }
1612     { \l__zrefclever_zcref_with_check_bool }
1613     {
1614         \zrefcheck_zcref_end_label_maybe:
1615         \zrefcheck_zcref_run_checks_on_labels:n
1616         { \l__zrefclever_zcref_labels_seq }
1617     }

```

Integration with mathtools.

```

1618     \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1619     {
1620         \__zrefclever_mathtools_showonlyrefs:n
1621         { \l__zrefclever_zcref_labels_seq }
1622     }
1623     \group_end:
1624 }

```

(End definition for __zrefclever_zcref:nnnn.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1625 \seq_new:N \l__zrefclever_zcref_labels_seq
1626 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for \l__zrefclever_zcref_labels_seq and \l__zrefclever_link_star_bool.)

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

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

```

1627 \NewDocumentCommand \zcpageref { s O { } m }
1628 {
1629     \IfBooleanTF {#1}
1630     { \zcref*[#2, ref = page] {#3} }
1631     { \zcref [ #2, ref = page] {#3} }
1632 }

```

(End definition for \zcpageref.)

7 Sorting

Sorting is certainly a “big task” for `zref-clever` but, in the end, it boils down to “carefully done branching”, and quite some of it. The sorting of “page” references is very much lightened by the availability of `abspage`, from the `zref-abspage` module, which offers “just what we need” for our purposes. The sorting of “default” references falls on two main cases: i) labels of the same type; ii) labels of different types. The first case is sorted according to the priorities set by the `typesort` option or, if that is silent for the case, by the order in which labels were given by the user in `\zceref`. 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` 1633 `\tl_new:N \l__zrefclever_label_type_a_tl`

`\l_zrefclever_label_enclval_b_tl` 1634 `\tl_new:N \l__zrefclever_label_type_b_tl`

`\l_zrefclever_label_extdoc_a_tl` 1635 `\tl_new:N \l__zrefclever_label_enclval_a_tl`

`\l_zrefclever_label_extdoc_b_tl` 1636 `\tl_new:N \l__zrefclever_label_enclval_b_tl`

1637 `\tl_new:N \l__zrefclever_label_extdoc_a_tl`

1638 `\tl_new:N \l__zrefclever_label_extdoc_b_tl`

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

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

1639 `\bool_new:N \l__zrefclever_sort_decided_bool`

(End definition for `\l__zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

1640 `\int_new:N \l__zrefclever_sort_prior_a_int`

1641 `\int_new:N \l__zrefclever_sort_prior_b_int`

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

`\l_zrefclever_label_types_seq` Stores the order in which reference types appear in the label list supplied by the user in `\zceref`. 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`.

1642 `\seq_new:N \l__zrefclever_label_types_seq`

(End definition for `\l__zrefclever_label_types_seq`.)

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

1643 `\cs_new_protected:Npn __zrefclever_sort_labels:`

1644 `{`

Store label types sequence.

```

1645 \seq_clear:N \l__zrefclever_label_types_seq
1646 \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1647 {
1648   \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1649   \__zrefclever_label_type_put_new_right:n
1650 }

```

Sort.

```

1651 \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1652 {
1653   \zref@ifrefundefined {##1}
1654   {
1655     \zref@ifrefundefined {##2}
1656     {
1657       % Neither label is defined.
1658       \sort_return_same:
1659     }
1660     {
1661       % The second label is defined, but the first isn't, leave the
1662       % undefined first (to be more visible).
1663       \sort_return_same:
1664     }
1665   }
1666   {
1667     \zref@ifrefundefined {##2}
1668     {
1669       % The first label is defined, but the second isn't, bring the
1670       % second forward.
1671       \sort_return_swapped:
1672     }
1673     {
1674       % The interesting case: both labels are defined. References
1675       % to the "default" property or to the "page" are quite
1676       % different with regard to sorting, so we branch them here to
1677       % specialized functions.
1678       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1679       { \__zrefclever_sort_page:nn {##1} {##2} }
1680       { \__zrefclever_sort_default:nn {##1} {##2} }
1681     }
1682   }
1683 }
1684 }

```

(End definition for __zrefclever_sort_labels:.)

__zrefclever_label_type_put_new_right:n

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

```

        \_zrefclever_label_type_put_new_right:n {\label}}
1685 \cs_new_protected:Npn \_zrefclever_label_type_put_new_right:n #1
1686 {
1687   \_zrefclever_def_extract:Nnnn
1688   \l__zrefclever_label_type_a_tl {#1} {zc@type} {\c_empty_tl}
1689   \seq_if_in:NVF \l__zrefclever_label_types_seq
1690   \l__zrefclever_label_type_a_tl
1691   {
1692     \seq_put_right:NV \l__zrefclever_label_types_seq
1693     \l__zrefclever_label_type_a_tl
1694   }
1695 }

```

(End definition for _zrefclever_label_type_put_new_right:n.)

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

```

        \_zrefclever_sort_default:nn {\label a} {\label b}
1696 \cs_new_protected:Npn \_zrefclever_sort_default:nn #1#2
1697 {
1698   \_zrefclever_def_extract:Nnnn
1699   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
1700   \_zrefclever_def_extract:Nnnn
1701   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
1702
1703   \tl_if_eq:NNTF
1704   \l__zrefclever_label_type_a_tl
1705   \l__zrefclever_label_type_b_tl
1706   { \_zrefclever_sort_default_same_type:nn {#1} {#2} }
1707   { \_zrefclever_sort_default_different_types:nn {#1} {#2} }
1708 }

```

(End definition for _zrefclever_sort_default:nn.)

```

\_zrefclever_sort_default_same_type:nn        \_zrefclever_sort_default_same_type:nn {\label a} {\label b}
1709 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1710 {
1711   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_a_tl
1712   {#1} {zc@enclval} {\c_empty_tl}
1713   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1714   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_enclval_b_tl
1715   {#2} {zc@enclval} {\c_empty_tl}
1716   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1717   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
1718   {#1} {externaldocument} {\c_empty_tl}
1719   \_zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
1720   {#2} {externaldocument} {\c_empty_tl}
1721
1722   \bool_set_false:N \l__zrefclever_sort_decided_bool

```

```

1723
1724 % First we check if there's any "external document" difference (coming
1725 % from 'zref-xr') and, if so, sort based on that.
1726 \tl_if_eq:NNF
1727   \l__zrefclever_label_extdoc_a_tl
1728   \l__zrefclever_label_extdoc_b_tl
1729 {
1730   \bool_if:nTF
1731   {
1732     \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1733     ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1734   }
1735   {
1736     \bool_set_true:N \l__zrefclever_sort_decided_bool
1737     \sort_return_same:
1738   }
1739   {
1740     \bool_if:nTF
1741     {
1742       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1743       \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1744     }
1745     {
1746       \bool_set_true:N \l__zrefclever_sort_decided_bool
1747       \sort_return_swapped:
1748     }
1749     {
1750       \bool_set_true:N \l__zrefclever_sort_decided_bool
1751       % Two different "external documents": last resort, sort by the
1752       % document name itself.
1753       \str_compare:eNeTF
1754       { \l__zrefclever_label_extdoc_b_tl } <
1755       { \l__zrefclever_label_extdoc_a_tl }
1756       { \sort_return_swapped: }
1757       { \sort_return_same: }
1758     }
1759   }
1760 }
1761
1762 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1763 {
1764   \bool_if:nTF
1765   {
1766     % Both are empty: neither label has any (further) "enclosing
1767     % counters" (left).
1768     \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
1769     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1770   }
1771   {
1772     \bool_set_true:N \l__zrefclever_sort_decided_bool
1773     \int_compare:nNnTF
1774     { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
1775     >
1776     { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }

```

```

1777     { \sort_return_swapped: }
1778     { \sort_return_same:   }
1779   }
1780   {
1781     \bool_if:nTF
1782     {
1783       % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1784       \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
1785     }
1786     {
1787       \bool_set_true:N \l__zrefclever_sort_decided_bool
1788       \int_compare:nNnTF
1789       { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
1790       >
1791       { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1792       { \sort_return_swapped: }
1793       { \sort_return_same:   }
1794     }
1795   }
1796   \bool_if:nTF
1797   {
1798     % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1799     \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
1800   }
1801   {
1802     \bool_set_true:N \l__zrefclever_sort_decided_bool
1803     \int_compare:nNnTF
1804     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1805     <
1806     { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
1807     { \sort_return_same:   }
1808     { \sort_return_swapped: }
1809   }
1810   {
1811     % Neither is empty: we can compare the values of the
1812     % current enclosing counter in the loop, if they are
1813     % equal, we are still in the loop, if they are not, a
1814     % sorting decision can be made directly.
1815     \int_compare:nNnTF
1816     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1817     =
1818     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1819     {
1820       \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1821       { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1822       \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1823       { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1824     }
1825     {
1826       \bool_set_true:N \l__zrefclever_sort_decided_bool
1827       \int_compare:nNnTF
1828       { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1829       >
1830       { \tl_head:N \l__zrefclever_label_enclval_b_tl }

```

```

1831         { \sort_return_swapped: }
1832         { \sort_return_same: }
1833     }
1834 }
1835 }
1836 }
1837 }
1838 }

```

(End definition for `_zrefclever_sort_default_same_type:nn`.)

`_zrefclever_sort_default_different_types:nn`

```

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

```

```

1839 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1840 {

```

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

```

1841     \int_zero:N \l__zrefclever_sort_prior_a_int
1842     \int_zero:N \l__zrefclever_sort_prior_b_int
1843     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1844     {
1845         \tl_if_eq:nnTF {##2} {{othertypes}}
1846         {
1847             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1848             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1849             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1850             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1851         }
1852         {
1853             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1854             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1855             {
1856                 \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1857                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1858             }
1859         }
1860     }

```

Then do the actual sorting.

```

1861     \bool_if:nTF
1862     {
1863         \int_compare_p:nNn
1864         { \l__zrefclever_sort_prior_a_int } <
1865         { \l__zrefclever_sort_prior_b_int }
1866     }
1867     { \sort_return_same: }
1868     {
1869         \bool_if:nTF
1870         {
1871             \int_compare_p:nNn
1872             { \l__zrefclever_sort_prior_a_int } >
1873             { \l__zrefclever_sort_prior_b_int }
1874         }

```

```

1875         { \sort_return_swapped: }
1876     {
1877         % Sort priorities are equal: the type that occurs first in
1878         % ‘labels’, as given by the user, is kept (or brought) forward.
1879         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1880         {
1881             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1882             { \seq_map_break:n { \sort_return_same: } }
1883             {
1884                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1885                 { \seq_map_break:n { \sort_return_swapped: } }
1886             }
1887         }
1888     }
1889 }
1890 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```

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

1891 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1892 {
1893     \int_compare:nNnTF
1894     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
1895     >
1896     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
1897     { \sort_return_swapped: }
1898     { \sort_return_same: }
1899 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

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

While processing the label stack (kept in `\l__zrefclever_typeset_labels_seq`), `__zrefclever_typeset_refs:` “sees” two labels, and two labels only, the “current” one (kept in `\l__zrefclever_label_a_tl`), and the “next” one (kept in `\l__zrefclever_label_b_tl`). However, the typesetting needs (a lot) more information than just these

two immediate labels to make a number of critical decisions. Some examples: i) We cannot know if labels “current” and “next” of the same type are a “pair”, or just “elements in a list”, until we examine the label after “next”; ii) If the “next” label is of the same type as the “current”, and it is in immediate sequence to it, it potentially forms a “range”, but we cannot know if “next” is actually the end of the range until we examined an arbitrary number of labels, and found one which is not in sequence from the previous one; iii) When processing a type block, the “name” comes first, however, we only know if that name should be plural, or if it should be included in the hyperlink, after processing an arbitrary number of labels and find one of a different type. One could naively assume that just examining “next” would be enough for this, since we can know if it is of the same type or not. Alas, “there be ranges”, and a compression operation may boil down to a single element, so we have to process the whole type block to know how its name should be typeset; iv) Similar issues apply to lists of type blocks, each of which is of arbitrary length: we can only know if two type blocks form a “pair” or are “elements in a list” when we finish the block. Etc. etc. etc.

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

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

Range compression, in particular, relies heavily on counting to be able to distinguish relevant cases. `\l__zrefclever_range_count_int` counts the number of elements in the current sequential “streak”, and `\l__zrefclever_range_same_count_int` counts the number of *equal* elements in that same “streak”. The difference between the two allows us to distinguish the cases in which a range actually “skips” a number in the sequence, in which case we should use a range separator, from when they are after all just contiguous, in which case a pair separator is called for. Since, as usual, we can only know this when a arbitrary long “streak” finishes, we have to store the label which (potentially) begins a range (kept in `\l__zrefclever_range_beg_label_tl`). `\l__zrefclever_next_maybe_range_bool` signals when “next” is potentially a range with “current”, and `\l__zrefclever_next_is_same_bool` when their values are actually equal.

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this, suggested by Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes (and good ones at that) see <https://tex.stackexchange.com/q/611370>. Yet another alternative would be an option receiving the label(s) not to be compressed, this would be a repetition, but would keep the syntax clean. All in all, probably the best is simply not to allow individual inhibition of compression. We can already control compression of each `\zcref` call with existing options, this should be enough. I don’t think the small extra

flexibility individual label control for this would grant is worth the syntax disruption it would entail. Anyway, it would be easy to deal with this in case the need arose, by just adding another condition (coming from whatever the chosen syntax was) when we check for `__zrefclever_labels_in_sequence:nn` in `__zrefclever_typeset_refs_not_last_of_type:.` But I remain unconvinced of the pertinence of doing so.

Variables

<code>\l_zrefclever_typeset_labels_seq</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main stack control.
<code>\l_zrefclever_typeset_last_bool</code>	1900 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l_zrefclever_last_of_type_bool</code>	1901 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	1902 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>
	(End definition for <code>\l_zrefclever_typeset_labels_seq</code> , <code>\l_zrefclever_typeset_last_bool</code> , and <code>\l_zrefclever_last_of_type_bool</code> .)
<code>\l_zrefclever_type_count_int</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main counters.
<code>\l_zrefclever_label_count_int</code>	1903 <code>\int_new:N \l__zrefclever_type_count_int</code>
	1904 <code>\int_new:N \l__zrefclever_label_count_int</code>
	(End definition for <code>\l_zrefclever_type_count_int</code> and <code>\l_zrefclever_label_count_int</code> .)
<code>\l__zrefclever_label_a_tl</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main “queue” control and storage.
<code>\l__zrefclever_label_b_tl</code>	
<code>\l_zrefclever_typeset_queue_prev_tl</code>	1905 <code>\tl_new:N \l__zrefclever_label_a_tl</code>
<code>\l_zrefclever_typeset_queue_curr_tl</code>	1906 <code>\tl_new:N \l__zrefclever_label_b_tl</code>
<code>\l_zrefclever_type_first_label_tl</code>	1907 <code>\tl_new:N \l__zrefclever_typeset_queue_prev_tl</code>
<code>\l_zrefclever_type_first_label_type_tl</code>	1908 <code>\tl_new:N \l__zrefclever_typeset_queue_curr_tl</code>
	1909 <code>\tl_new:N \l__zrefclever_type_first_label_tl</code>
	1910 <code>\tl_new:N \l__zrefclever_type_first_label_type_tl</code>
	(End definition for <code>\l__zrefclever_label_a_tl</code> and others.)
<code>\l__zrefclever_type_name_tl</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : type name handling.
<code>\l_zrefclever_name_in_link_bool</code>	1911 <code>\tl_new:N \l__zrefclever_type_name_tl</code>
<code>\l_zrefclever_name_format_tl</code>	1912 <code>\bool_new:N \l__zrefclever_name_in_link_bool</code>
<code>\l_zrefclever_name_format_fallback_tl</code>	1913 <code>\tl_new:N \l__zrefclever_name_format_tl</code>
<code>\l_zrefclever_type_name_gender_tl</code>	1914 <code>\tl_new:N \l__zrefclever_name_format_fallback_tl</code>
	1915 <code>\tl_new:N \l__zrefclever_type_name_gender_tl</code>
	(End definition for <code>\l__zrefclever_type_name_tl</code> and others.)
<code>\l_zrefclever_range_count_int</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : range handling.
<code>\l_zrefclever_range_same_count_int</code>	1916 <code>\int_new:N \l__zrefclever_range_count_int</code>
<code>\l_zrefclever_range_beg_label_tl</code>	1917 <code>\int_new:N \l__zrefclever_range_same_count_int</code>
<code>\l_zrefclever_next_maybe_range_bool</code>	1918 <code>\tl_new:N \l__zrefclever_range_beg_label_tl</code>
<code>\l_zrefclever_next_is_same_bool</code>	1919 <code>\bool_new:N \l__zrefclever_next_maybe_range_bool</code>
	1920 <code>\bool_new:N \l__zrefclever_next_is_same_bool</code>
	(End definition for <code>\l_zrefclever_range_count_int</code> and others.)

`\l__zrefclever_tpairsep_tl` Auxiliary variables for `__zrefclever_typeset_refs`: separators, `refpre/pos` and font options.

```

1921 \tl_new:N \l__zrefclever_tpairsep_tl
1922 \tl_new:N \l__zrefclever_tlistsep_tl
1923 \tl_new:N \l__zrefclever_tlastsep_tl
1924 \tl_new:N \l__zrefclever_namesep_tl
1925 \tl_new:N \l__zrefclever_pairsep_tl
1926 \tl_new:N \l__zrefclever_listsep_tl
1927 \tl_new:N \l__zrefclever_lastsep_tl
1928 \tl_new:N \l__zrefclever_rangesep_tl
1929 \tl_new:N \l__zrefclever_refpre_tl
1930 \tl_new:N \l__zrefclever_refpos_tl
1931 \tl_new:N \l__zrefclever_namefont_tl
1932 \tl_new:N \l__zrefclever_reffont_tl

```

(End definition for `\l__zrefclever_tpairsep_tl` 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`.

```
1933 \bool_new:N \l__zrefclever_verbose_testing_bool
```

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```

1934 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1935 {
1936   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1937   \l__zrefclever_zcref_labels_seq
1938   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1939   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1940   \tl_clear:N \l__zrefclever_type_first_label_tl
1941   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1942   \tl_clear:N \l__zrefclever_range_beg_label_tl
1943   \int_zero:N \l__zrefclever_label_count_int
1944   \int_zero:N \l__zrefclever_type_count_int
1945   \int_zero:N \l__zrefclever_range_count_int
1946   \int_zero:N \l__zrefclever_range_same_count_int
1947
1948   % Get type block options (not type-specific).
1949   \__zrefclever_get_ref_string:nN { tpairsep }
1950   \l__zrefclever_tpairsep_tl
1951   \__zrefclever_get_ref_string:nN { tlistsep }
1952   \l__zrefclever_tlistsep_tl
1953   \__zrefclever_get_ref_string:nN { tlastsep }
1954   \l__zrefclever_tlastsep_tl
1955
1956   % Process label stack.
1957   \bool_set_false:N \l__zrefclever_typeset_last_bool
1958   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1959   {
1960     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq

```

```

1961 \l__zrefclever_label_a_tl
1962 \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1963 {
1964   \tl_clear:N \l__zrefclever_label_b_tl
1965   \bool_set_true:N \l__zrefclever_typeset_last_bool
1966 }
1967 {
1968   \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1969   \l__zrefclever_label_b_tl
1970 }
1971
1972 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1973 {
1974   \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1975   \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1976 }
1977 {
1978   \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_a_tl
1979   \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
1980   \__zrefclever_def_extract:Nvnn \l__zrefclever_label_type_b_tl
1981   \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
1982 }
1983
1984 % First, we establish whether the "current label" (i.e. 'a') is the
1985 % last one of its type. This can happen because the "next label"
1986 % (i.e. 'b') is of a different type (or different definition status),
1987 % or because we are at the end of the list.
1988 \bool_if:NTF \l__zrefclever_typeset_last_bool
1989 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1990 {
1991   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1992   {
1993     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1994     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1995     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1996   }
1997   {
1998     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1999     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2000     {
2001       % Neither is undefined, we must check the types.
2002       \tl_if_eq:NNTF
2003         \l__zrefclever_label_type_a_tl
2004         \l__zrefclever_label_type_b_tl
2005         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2006         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2007     }
2008   }
2009 }
2010
2011 % Handle warnings in case of reference or type undefined.
2012 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
2013 \zref@refused { \l__zrefclever_label_a_tl }
2014 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"

```

```

2015 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2016 {}
2017 {
2018   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
2019   {
2020     \msg_warning:nxx { zref-clever } { missing-type }
2021     { \l__zrefclever_label_a_tl }
2022   }
2023   \zref@ifrefcontainsprop
2024   { \l__zrefclever_label_a_tl }
2025   { \l__zrefclever_ref_property_tl }
2026   { }
2027   {
2028     \msg_warning:nxxx { zref-clever } { missing-property }
2029     { \l__zrefclever_ref_property_tl }
2030     { \l__zrefclever_label_a_tl }
2031   }
2032 }
2033
2034 % Get type-specific separators, refpre/pos and font options, once per
2035 % type.
2036 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
2037 {
2038   \__zrefclever_get_ref_string:nN { namesep }
2039   \l__zrefclever_namesep_tl
2040   \__zrefclever_get_ref_string:nN { pairsep }
2041   \l__zrefclever_pairsep_tl
2042   \__zrefclever_get_ref_string:nN { listsep }
2043   \l__zrefclever_listsep_tl
2044   \__zrefclever_get_ref_string:nN { lastsep }
2045   \l__zrefclever_lastsep_tl
2046   \__zrefclever_get_ref_string:nN { rangesep }
2047   \l__zrefclever_rangesep_tl
2048   \__zrefclever_get_ref_string:nN { refpre }
2049   \l__zrefclever_refpre_tl
2050   \__zrefclever_get_ref_string:nN { refpos }
2051   \l__zrefclever_refpos_tl
2052   \__zrefclever_get_ref_font:nN { namefont }
2053   \l__zrefclever_namefont_tl
2054   \__zrefclever_get_ref_font:nN { reffont }
2055   \l__zrefclever_reffont_tl
2056 }
2057
2058 % Here we send this to a couple of auxiliary functions.
2059 \bool_if:NTF \l__zrefclever_last_of_type_bool
2060 % There exists no next label of the same type as the current.
2061 { \__zrefclever_typeset_refs_last_of_type: }
2062 % There exists a next label of the same type as the current.
2063 { \__zrefclever_typeset_refs_not_last_of_type: }
2064 }
2065 }

```

(End definition for `__zrefclever_typeset_refs:.`)

This is actually the one meaningful “big branching” we can do while processing the

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

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

```

2066 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
2067 {
2068   % Process the current label to the current queue.
2069   \int_case:nnF { \l__zrefclever_label_count_int }
2070   {
2071     % It is the last label of its type, but also the first one, and that's
2072     % what matters here: just store it.
2073     % Test: 'zc-typeset01.lvt': "Last of type: single"
2074     { 0 }
2075     {
2076       \tl_set:NV \l__zrefclever_type_first_label_tl
2077       \l__zrefclever_label_a_tl
2078       \tl_set:NV \l__zrefclever_type_first_label_type_tl
2079       \l__zrefclever_label_type_a_tl
2080     }
2081
2082     % The last is the second: we have a pair (if not repeated).
2083     % Test: 'zc-typeset01.lvt': "Last of type: pair"
2084     { 1 }
2085     {
2086       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
2087       {
2088         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2089         {
2090           \exp_not:V \l__zrefclever_pairsep_tl
2091           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2092         }
2093       }
2094     }
2095   }
2096   % Last is third or more of its type: without repetition, we'd have the
2097   % last element on a list, but control for possible repetition.
2098   {
2099     \int_case:nnF { \l__zrefclever_range_count_int }
2100     {
2101       % There was no range going on.
2102       % Test: 'zc-typeset01.lvt': "Last of type: not range"
2103       { 0 }
2104       {
2105         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2106         {
2107           \exp_not:V \l__zrefclever_lastsep_tl
2108           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl

```

```

2109     }
2110 }
2111 % Last in the range is also the second in it.
2112 % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
2113 { 1 }
2114 {
2115   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2116   {
2117     % We know 'range_beg_label' is not empty, since this is the
2118     % second element in the range, but the third or more in the
2119     % type list.
2120     \exp_not:V \l__zrefclever_listsep_tl
2121     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2122     \int_compare:nNnF
2123       { \l__zrefclever_range_same_count_int } = { 1 }
2124       {
2125         \exp_not:V \l__zrefclever_lastsep_tl
2126         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2127       }
2128   }
2129 }
2130 }
2131 % Last in the range is third or more in it.
2132 {
2133   \int_case:nnF
2134   {
2135     \l__zrefclever_range_count_int -
2136     \l__zrefclever_range_same_count_int
2137   }
2138   {
2139     % Repetition, not a range.
2140     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
2141     { 0 }
2142     {
2143       % If 'range_beg_label' is empty, it means it was also the
2144       % first of the type, and hence was already handled.
2145       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2146       {
2147         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2148         {
2149           \exp_not:V \l__zrefclever_lastsep_tl
2150           \__zrefclever_get_ref:V
2151             \l__zrefclever_range_beg_label_tl
2152         }
2153       }
2154     }
2155     % A 'range', but with no skipped value, treat as list.
2156     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
2157     { 1 }
2158     {
2159       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2160       {
2161         % Ditto.
2162         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl

```

```

2163         {
2164             \exp_not:V \l__zrefclever_listsep_tl
2165             \__zrefclever_get_ref:V
2166             \l__zrefclever_range_beg_label_tl
2167         }
2168         \exp_not:V \l__zrefclever_lastsep_tl
2169         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2170     }
2171 }
2172 }
2173 {
2174     % An actual range.
2175     % Test: 'zc-typeset01.lvt': "Last of type: range"
2176     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2177     {
2178         % Ditto.
2179         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2180         {
2181             \exp_not:V \l__zrefclever_lastsep_tl
2182             \__zrefclever_get_ref:V
2183             \l__zrefclever_range_beg_label_tl
2184         }
2185         \exp_not:V \l__zrefclever_rangesep_tl
2186         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2187     }
2188 }
2189 }
2190 }
2191
2192 % Handle "range" option. The idea is simple: if the queue is not empty,
2193 % we replace it with the end of the range (or pair). We can still
2194 % retrieve the end of the range from 'label_a' since we know to be
2195 % processing the last label of its type at this point.
2196 \bool_if:NT \l__zrefclever_typeset_range_bool
2197 {
2198     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2199     {
2200         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2201         { }
2202         {
2203             \msg_warning:nxx { zref-clever } { single-element-range }
2204             { \l__zrefclever_type_first_label_type_tl }
2205         }
2206     }
2207     {
2208         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2209         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2210         { }
2211         {
2212             \__zrefclever_labels_in_sequence:nn
2213             { \l__zrefclever_type_first_label_tl }
2214             { \l__zrefclever_label_a_tl }
2215         }
2216         % Test: 'zc-typeset01.lvt': "Last of type: option range"

```

```

2217 % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
2218 \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2219 {
2220   \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2221     { \exp_not:V \l__zrefclever_pairsep_tl }
2222     { \exp_not:V \l__zrefclever_rangesep_tl }
2223   \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2224 }
2225 }
2226 }
2227
2228 % Now that the type block is finished, we can add the name and the first
2229 % ref to the queue. Also, if "typeset" option is not "both", handle it
2230 % here as well.
2231 \__zrefclever_type_name_setup:
2232 \bool_if:NTF
2233 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2234 {
2235   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2236     { \__zrefclever_get_ref_first: }
2237 }
2238 {
2239   \bool_if:NTF \l__zrefclever_typeset_ref_bool
2240   {
2241     % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
2242     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2243       { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2244   }
2245   {
2246     \bool_if:NTF \l__zrefclever_typeset_name_bool
2247     {
2248       % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
2249       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2250       {
2251         \bool_if:NTF \l__zrefclever_name_in_link_bool
2252         {
2253           \exp_not:N \group_begin:
2254           \exp_not:V \l__zrefclever_namefont_tl
2255           % It's two '@s', but escaped for DocStrip.
2256           \exp_not:N \hyper@@link
2257           {
2258             \__zrefclever_extract_url_unexp:V
2259             \l__zrefclever_type_first_label_tl
2260           }
2261           {
2262             \__zrefclever_extract_unexp:Vnn
2263             \l__zrefclever_type_first_label_tl
2264             { anchor } { }
2265           }
2266           { \exp_not:V \l__zrefclever_type_name_tl }
2267           \exp_not:N \group_end:
2268         }
2269         {
2270           \exp_not:N \group_begin:

```

```

2271         \exp_not:V \l__zrefclever_namefont_tl
2272         \exp_not:V \l__zrefclever_type_name_tl
2273         \exp_not:N \group_end:
2274     }
2275 }
2276 }
2277 {
2278     % Logically, this case would correspond to "typeset=none", but
2279     % it should not occur, given that the options are set up to
2280     % typeset either "ref" or "name". Still, leave here a
2281     % sensible fallback, equal to the behavior of "both".
2282     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
2283     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2284         { \l__zrefclever_get_ref_first: }
2285 }
2286 }
2287 }
2288
2289 % Typeset the previous type block, if there is one.
2290 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
2291 {
2292     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
2293     { \l__zrefclever_tlistsep_tl }
2294     \l__zrefclever_typeset_queue_prev_tl
2295 }
2296
2297 % Extra log for testing.
2298 \bool_if:NT \l__zrefclever_verbose_testing_bool
2299 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
2300
2301 % Wrap up loop, or prepare for next iteration.
2302 \bool_if:NTF \l__zrefclever_typeset_last_bool
2303 {
2304     % We are finishing, typeset the current queue.
2305     \int_case:nnF { \l__zrefclever_type_count_int }
2306     {
2307         % Single type.
2308         % Test: 'zc-typeset01.lvt': "Last of type: single type"
2309         { 0 }
2310         { \l__zrefclever_typeset_queue_curr_tl }
2311         % Pair of types.
2312         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
2313         { 1 }
2314         {
2315             \l__zrefclever_tpairsep_tl
2316             \l__zrefclever_typeset_queue_curr_tl
2317         }
2318     }
2319     {
2320         % Last in list of types.
2321         % Test: 'zc-typeset01.lvt': "Last of type: list of types"
2322         \l__zrefclever_tlastsep_tl
2323         \l__zrefclever_typeset_queue_curr_tl
2324     }

```



```

2325 % And nudge in case of multitype reference.
2326 \bool_lazy_all:nT
2327 {
2328   { \l__zrefclever_nudge_enabled_bool }
2329   { \l__zrefclever_nudge_multitype_bool }
2330   { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
2331 }
2332 { \msg_warning:nn { zref-clever } { nudge-multitype } }
2333 }
2334 {
2335   % There are further labels, set variables for next iteration.
2336   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2337   \l__zrefclever_typeset_queue_curr_tl
2338   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2339   \tl_clear:N \l__zrefclever_type_first_label_tl
2340   \tl_clear:N \l__zrefclever_type_first_label_type_tl
2341   \tl_clear:N \l__zrefclever_range_beg_label_tl
2342   \int_zero:N \l__zrefclever_label_count_int
2343   \int_incr:N \l__zrefclever_type_count_int
2344   \int_zero:N \l__zrefclever_range_count_int
2345   \int_zero:N \l__zrefclever_range_same_count_int
2346 }
2347 }

```

(End definition for __zrefclever_typeset_refs_last_of_type:.)

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

```

2348 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
2349 {
2350   % Signal if next label may form a range with the current one (only
2351   % considered if compression is enabled in the first place).
2352   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2353   \bool_set_false:N \l__zrefclever_next_is_same_bool
2354   \bool_if:NT \l__zrefclever_typeset_compress_bool
2355   {
2356     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2357     { }
2358     {
2359       \__zrefclever_labels_in_sequence:nn
2360       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2361     }
2362   }
2363
2364   % Process the current label to the current queue.
2365   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2366   {
2367     % Current label is the first of its type (also not the last, but it
2368     % doesn't matter here): just store the label.
2369     \tl_set:NV \l__zrefclever_type_first_label_tl
2370     \l__zrefclever_label_a_tl
2371     \tl_set:NV \l__zrefclever_type_first_label_type_tl
2372     \l__zrefclever_label_type_a_tl
2373
2374     % If the next label may be part of a range, we set 'range_beg_label'

```

```

2375 % to "empty" (we deal with it as the "first", and must do it there, to
2376 % handle hyperlinking), but also step the range counters.
2377 % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
2378 \bool_if:NT \l__zrefclever_next_maybe_range_bool
2379 {
2380   \tl_clear:N \l__zrefclever_range_beg_label_tl
2381   \int_incr:N \l__zrefclever_range_count_int
2382   \bool_if:NT \l__zrefclever_next_is_same_bool
2383     { \int_incr:N \l__zrefclever_range_same_count_int }
2384 }
2385 }
2386 {
2387 % Current label is neither the first (nor the last) of its type.
2388 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2389 {
2390   % Starting, or continuing a range.
2391   \int_compare:nNnTF
2392     { \l__zrefclever_range_count_int } = { 0 }
2393   {
2394     % There was no range going, we are starting one.
2395     \tl_set:NV \l__zrefclever_range_beg_label_tl
2396       \l__zrefclever_label_a_tl
2397     \int_incr:N \l__zrefclever_range_count_int
2398     \bool_if:NT \l__zrefclever_next_is_same_bool
2399       { \int_incr:N \l__zrefclever_range_same_count_int }
2400   }
2401   {
2402     % Second or more in the range, but not the last.
2403     \int_incr:N \l__zrefclever_range_count_int
2404     \bool_if:NT \l__zrefclever_next_is_same_bool
2405       { \int_incr:N \l__zrefclever_range_same_count_int }
2406   }
2407 }
2408 {
2409 % Next element is not in sequence: there was no range, or we are
2410 % closing one.
2411 \int_case:nnF { \l__zrefclever_range_count_int }
2412 {
2413   % There was no range going on.
2414   % Test: 'zc-typeset01.lvt': "Not last of type: no range"
2415   { 0 }
2416   {
2417     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2418       {
2419         \exp_not:V \l__zrefclever_listsep_tl
2420         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2421       }
2422   }
2423   % Last is second in the range: if 'range_same_count' is also
2424   % '1', it's a repetition (drop it), otherwise, it's a "pair
2425   % within a list", treat as list.
2426   % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
2427   % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
2428   { 1 }

```

```

2429 {
2430   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2431   {
2432     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2433     {
2434       \exp_not:V \l__zrefclever_listsep_tl
2435       \__zrefclever_get_ref:V
2436       \l__zrefclever_range_beg_label_tl
2437     }
2438     \int_compare:nNnF
2439     { \l__zrefclever_range_same_count_int } = { 1 }
2440     {
2441       \exp_not:V \l__zrefclever_listsep_tl
2442       \__zrefclever_get_ref:V
2443       \l__zrefclever_label_a_tl
2444     }
2445   }
2446 }
2447 }
2448 {
2449   % Last is third or more in the range: if 'range_count' and
2450   % 'range_same_count' are the same, its a repetition (drop it),
2451   % if they differ by '1', its a list, if they differ by more,
2452   % it is a real range.
2453   \int_case:nnF
2454   {
2455     \l__zrefclever_range_count_int -
2456     \l__zrefclever_range_same_count_int
2457   }
2458   {
2459     % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
2460     { 0 }
2461     {
2462       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2463       {
2464         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2465         {
2466           \exp_not:V \l__zrefclever_listsep_tl
2467           \__zrefclever_get_ref:V
2468           \l__zrefclever_range_beg_label_tl
2469         }
2470       }
2471     }
2472     % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
2473     { 1 }
2474     {
2475       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2476       {
2477         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2478         {
2479           \exp_not:V \l__zrefclever_listsep_tl
2480           \__zrefclever_get_ref:V
2481           \l__zrefclever_range_beg_label_tl
2482         }

```

```

2483         \exp_not:V \l__zrefclever_listsep_tl
2484         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2485     }
2486 }
2487 }
2488 {
2489     % Test: 'zc-typeset01.lvt': "Not last of type: range"
2490     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2491     {
2492         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2493         {
2494             \exp_not:V \l__zrefclever_listsep_tl
2495             \__zrefclever_get_ref:V
2496             \l__zrefclever_range_beg_label_tl
2497         }
2498         \exp_not:V \l__zrefclever_rangesept_tl
2499         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2500     }
2501 }
2502 }
2503 % Reset counters.
2504 \int_zero:N \l__zrefclever_range_count_int
2505 \int_zero:N \l__zrefclever_range_same_count_int
2506 }
2507 }
2508 % Step label counter for next iteration.
2509 \int_incr:N \l__zrefclever_label_count_int
2510 }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:.`)

Auxiliary functions

`__zrefclever_get_ref:n` and `__zrefclever_get_ref_first:` are the two functions which actually build the reference blocks for typesetting. `__zrefclever_get_ref:n` 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 ncial 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:n` 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.

```

2511 \cs_new_protected:Npn \__zrefclever_ref_default:
2512 { \zref@default }
2513 \cs_new_protected:Npn \__zrefclever_name_default:
2514 { \zref@default }

```

(End definition for `__zrefclever_ref_default:` and `__zrefclever_name_default:.`)

`__zrefclever_get_ref:n` Handles a complete reference block to be accumulated in the “queue”, including “pre” and “pos” elements, and hyperlinking. For use with all labels, except the first of its type, which is done by `__zrefclever_get_ref_first:.`

```

\__zrefclever_get_ref:n {<label>}

2515 \cs_new:Npn \__zrefclever_get_ref:n #1
2516 {
2517   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2518   {
2519     \bool_if:nTF
2520     {
2521       \l__zrefclever_use_hyperref_bool &&
2522       ! \l__zrefclever_link_star_bool
2523     }
2524     {
2525       \bool_if:NF \l__zrefclever_preposinlink_bool
2526       { \exp_not:V \l__zrefclever_refpre_tl }
2527       % It's two '@s', but escaped for DocStrip.
2528       \exp_not:N \hyper@@link
2529       { \__zrefclever_extract_url_unexp:n {#1} }
2530       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2531       {
2532         \bool_if:NT \l__zrefclever_preposinlink_bool
2533         { \exp_not:V \l__zrefclever_refpre_tl }
2534         \exp_not:N \group_begin:
2535         \exp_not:V \l__zrefclever_reffont_tl
2536         \__zrefclever_extract_unexp:nvn {#1}
2537         { \l__zrefclever_ref_property_tl } { }
2538         \exp_not:N \group_end:
2539         \bool_if:NT \l__zrefclever_preposinlink_bool
2540         { \exp_not:V \l__zrefclever_refpos_tl }
2541       }
2542       \bool_if:NF \l__zrefclever_preposinlink_bool
2543       { \exp_not:V \l__zrefclever_refpos_tl }
2544     }
2545     {
2546       \exp_not:V \l__zrefclever_refpre_tl
2547       \exp_not:N \group_begin:
2548       \exp_not:V \l__zrefclever_reffont_tl
2549       \__zrefclever_extract_unexp:nvn {#1}
2550       { \l__zrefclever_ref_property_tl } { }
2551       \exp_not:N \group_end:

```

```

2552         \exp_not:V \l__zrefclever_refpos_tl
2553     }
2554 }
2555 { \__zrefclever_ref_default: }
2556 }
2557 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

__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.

```

2558 \cs_new:Npn \__zrefclever_get_ref_first:
2559 {
2560     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2561     { \__zrefclever_ref_default: }
2562     {
2563         \bool_if:NTF \l__zrefclever_name_in_link_bool
2564         {
2565             \zref@ifrefcontainsprop
2566             { \l__zrefclever_type_first_label_tl }
2567             { \l__zrefclever_ref_property_tl }
2568             {
2569                 % It's two '@s', but escaped for DocStrip.
2570                 \exp_not:N \hyper@@link
2571                 {
2572                     \__zrefclever_extract_url_unexp:V
2573                     \l__zrefclever_type_first_label_tl
2574                 }
2575                 {
2576                     \__zrefclever_extract_unexp:Vnn
2577                     \l__zrefclever_type_first_label_tl { anchor } { }
2578                 }
2579             }
2580             {
2581                 \exp_not:N \group_begin:
2582                 \exp_not:V \l__zrefclever_namefont_tl
2583                 \exp_not:V \l__zrefclever_type_name_tl
2584                 \exp_not:N \group_end:
2585                 \exp_not:V \l__zrefclever_namesep_tl
2586                 \exp_not:V \l__zrefclever_refpre_tl
2587                 \exp_not:N \group_begin:
2588                 \exp_not:V \l__zrefclever_reffont_tl
2589                 \__zrefclever_extract_unexp:Vnn
2590                 \l__zrefclever_type_first_label_tl
2591                 { \l__zrefclever_ref_property_tl } { }
2592                 \exp_not:N \group_end:
2593                 \bool_if:NT \l__zrefclever_preposinlink_bool
2594                 { \exp_not:V \l__zrefclever_refpos_tl }

```

```

2595         \bool_if:NF \l__zrefclever_preposinlink_bool
2596         { \exp_not:V \l__zrefclever_refpos_tl }
2597     }
2598     {
2599         \exp_not:N \group_begin:
2600         \exp_not:V \l__zrefclever_namefont_tl
2601         \exp_not:V \l__zrefclever_type_name_tl
2602         \exp_not:N \group_end:
2603         \exp_not:V \l__zrefclever_namesep_tl
2604         \__zrefclever_ref_default:
2605     }
2606 }
2607 {
2608     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2609     {
2610         \__zrefclever_name_default:
2611         \exp_not:V \l__zrefclever_namesep_tl
2612     }
2613     {
2614         \exp_not:N \group_begin:
2615         \exp_not:V \l__zrefclever_namefont_tl
2616         \exp_not:V \l__zrefclever_type_name_tl
2617         \exp_not:N \group_end:
2618         \exp_not:V \l__zrefclever_namesep_tl
2619     }
2620 \zref@ifrefcontainsprop
2621 { \l__zrefclever_type_first_label_tl }
2622 { \l__zrefclever_ref_property_tl }
2623 {
2624     \bool_if:nTF
2625     {
2626         \l__zrefclever_use_hyperref_bool &&
2627         ! \l__zrefclever_link_star_bool
2628     }
2629     {
2630         \bool_if:NF \l__zrefclever_preposinlink_bool
2631         { \exp_not:V \l__zrefclever_refpre_tl }
2632         % It's two '@s', but escaped for DocStrip.
2633         \exp_not:N \hyper@@link
2634         {
2635             \__zrefclever_extract_url_unexp:V
2636             \l__zrefclever_type_first_label_tl
2637         }
2638         {
2639             \__zrefclever_extract_unexp:Vnn
2640             \l__zrefclever_type_first_label_tl { anchor } { }
2641         }
2642         {
2643             \bool_if:NT \l__zrefclever_preposinlink_bool
2644             { \exp_not:V \l__zrefclever_refpre_tl }
2645             \exp_not:N \group_begin:
2646             \exp_not:V \l__zrefclever_reffont_tl
2647             \__zrefclever_extract_unexp:Vnn
2648             \l__zrefclever_type_first_label_tl

```

```

2649         { \l__zrefclever_ref_property_tl } { }
2650         \exp_not:N \group_end:
2651         \bool_if:NT \l__zrefclever_preposinlink_bool
2652         { \exp_not:V \l__zrefclever_refpos_tl }
2653     }
2654     \bool_if:NF \l__zrefclever_preposinlink_bool
2655     { \exp_not:V \l__zrefclever_refpos_tl }
2656 }
2657 {
2658     \exp_not:V \l__zrefclever_refpre_tl
2659     \exp_not:N \group_begin:
2660     \exp_not:V \l__zrefclever_reffont_tl
2661     \__zrefclever_extract_unexp:Vvn
2662     \l__zrefclever_type_first_label_tl
2663     { \l__zrefclever_ref_property_tl } { }
2664     \exp_not:N \group_end:
2665     \exp_not:V \l__zrefclever_refpos_tl
2666 }
2667 }
2668 { \__zrefclever_ref_default: }
2669 }
2670 }
2671 }

```

(End definition for __zrefclever_get_ref_first:.)

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

```

2672 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2673 {
2674     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2675     { \tl_clear:N \l__zrefclever_type_name_tl }
2676     {
2677         \tl_if_eq:NnTF
2678         \l__zrefclever_type_first_label_type_tl { zc@missingtype }
2679         { \tl_clear:N \l__zrefclever_type_name_tl }
2680         {
2681             % Determine whether we should use capitalization, abbreviation,
2682             % and plural.
2683             \bool_lazy_or:nnTF
2684             { \l__zrefclever_capitalize_bool }
2685             {
2686                 \l__zrefclever_capitalize_first_bool &&
2687                 \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }

```



```

2688     }
2689     { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2690     { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2691 % If the queue is empty, we have a singular, otherwise, plural.
2692 \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2693   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2694   { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2695 \bool_lazy_and:nnTF
2696   { \l__zrefclever_abbrev_bool }
2697   {
2698     ! \int_compare_p:nNn
2699       { \l__zrefclever_type_count_int } = { 0 } ||
2700     ! \l__zrefclever_noabbrev_first_bool
2701   }
2702   {
2703     \tl_set:NV \l__zrefclever_name_format_fallback_tl
2704       \l__zrefclever_name_format_tl
2705     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2706   }
2707   { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2708
2709 % Handle number and gender nudges.
2710 \bool_if:NT \l__zrefclever_nudge_enabled_bool
2711   {
2712     \bool_if:NTF \l__zrefclever_nudge_singular_bool
2713       {
2714         \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2715           {
2716             \msg_warning:nnx { zref-clever }
2717               { nudge-plural-when-sg }
2718             { \l__zrefclever_type_first_label_type_tl }
2719           }
2720       }
2721       {
2722         \bool_lazy_all:nT
2723           {
2724             { \l__zrefclever_nudge_comptosing_bool }
2725             { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2726             {
2727               \int_compare_p:nNn
2728                 { \l__zrefclever_label_count_int } > { 0 }
2729             }
2730           }
2731           {
2732             \msg_warning:nnx { zref-clever }
2733               { nudge-comptosing }
2734             { \l__zrefclever_type_first_label_type_tl }
2735           }
2736       }
2737     \bool_lazy_and:nnT
2738       { \l__zrefclever_nudge_gender_bool }
2739       { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
2740       {
2741         \__zrefclever_get_type_transl:xxnNF

```

```

2742 { \l__zrefclever_ref_language_tl }
2743 { \l__zrefclever_type_first_label_type_tl }
2744 { gender }
2745 \l__zrefclever_type_name_gender_tl
2746 { \tl_clear:N \l__zrefclever_type_name_gender_tl }
2747 \tl_if_eq:NNF
2748 \l__zrefclever_ref_gender_tl
2749 \l__zrefclever_type_name_gender_tl
2750 {
2751   \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
2752   {
2753     \msg_warning:nnxxx { zref-clever }
2754     { nudge-gender-not-declared-for-type }
2755     { \l__zrefclever_ref_gender_tl }
2756     { \l__zrefclever_type_first_label_type_tl }
2757     { \l__zrefclever_ref_language_tl }
2758   }
2759   {
2760     \msg_warning:nnxxxx { zref-clever }
2761     { nudge-gender-mismatch }
2762     { \l__zrefclever_type_first_label_type_tl }
2763     { \l__zrefclever_ref_gender_tl }
2764     { \l__zrefclever_type_name_gender_tl }
2765     { \l__zrefclever_ref_language_tl }
2766   }
2767 }
2768 }
2769 }
2770
2771 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2772 {
2773   \prop_get:cVNF
2774   {
2775     \l__zrefclever_type_
2776     \l__zrefclever_type_first_label_type_tl _options_prop
2777   }
2778   \l__zrefclever_name_format_tl
2779   \l__zrefclever_type_name_tl
2780   {
2781     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2782     {
2783       \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
2784       \tl_put_left:NV \l__zrefclever_name_format_tl
2785         \l__zrefclever_ref_decl_case_tl
2786     }
2787     \__zrefclever_get_type_transl:xxxNF
2788     { \l__zrefclever_ref_language_tl }
2789     { \l__zrefclever_type_first_label_type_tl }
2790     { \l__zrefclever_name_format_tl }
2791     \l__zrefclever_type_name_tl
2792     {
2793       \tl_clear:N \l__zrefclever_type_name_tl
2794       \msg_warning:nnxx { zref-clever } { missing-name }
2795       { \l__zrefclever_name_format_tl }

```

```

2796         { \l__zrefclever_type_first_label_type_tl }
2797     }
2798 }
2799 {
2800 \prop_get:cVNF
2801 {
2802     l__zrefclever_type_
2803     \l__zrefclever_type_first_label_type_tl _options_prop
2804 }
2805 \l__zrefclever_name_format_tl
2806 \l__zrefclever_type_name_tl
2807 {
2808     \prop_get:cVNF
2809     {
2810         l__zrefclever_type_
2811         \l__zrefclever_type_first_label_type_tl _options_prop
2812     }
2813     \l__zrefclever_name_format_fallback_tl
2814     \l__zrefclever_type_name_tl
2815     {
2816         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
2817         {
2818             \tl_put_left:Nn
2819                 \l__zrefclever_name_format_tl { - }
2820             \tl_put_left:NV \l__zrefclever_name_format_tl
2821                 \l__zrefclever_ref_decl_case_tl
2822             \tl_put_left:Nn
2823                 \l__zrefclever_name_format_fallback_tl { - }
2824             \tl_put_left:NV
2825                 \l__zrefclever_name_format_fallback_tl
2826                 \l__zrefclever_ref_decl_case_tl
2827         }
2828     }
2829     \__zrefclever_get_type_transl:xxxNF
2830     { \l__zrefclever_ref_language_tl }
2831     { \l__zrefclever_type_first_label_type_tl }
2832     { \l__zrefclever_name_format_tl }
2833     \l__zrefclever_type_name_tl
2834     {
2835         \__zrefclever_get_type_transl:xxxNF
2836         { \l__zrefclever_ref_language_tl }
2837         { \l__zrefclever_type_first_label_type_tl }
2838         { \l__zrefclever_name_format_fallback_tl }
2839         \l__zrefclever_type_name_tl
2840         {
2841             \tl_clear:N \l__zrefclever_type_name_tl
2842             \msg_warning:nxxx { zref-clever }
2843             { missing-name }
2844             { \l__zrefclever_name_format_tl }
2845             { \l__zrefclever_type_first_label_type_tl }
2846         }
2847     }
2848 }
2849 }

```

```

2850     }
2851   }
2852 }
2853
2854 % Signal whether the type name is to be included in the hyperlink or not.
2855 \bool_lazy_any:nTF
2856 {
2857   { ! \l__zrefclever_use_hyperref_bool }
2858   { \l__zrefclever_link_star_bool }
2859   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2860   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2861 }
2862 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2863 {
2864   \bool_lazy_any:nTF
2865   {
2866     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2867     {
2868       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2869       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2870     }
2871     {
2872       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2873       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2874       \l__zrefclever_typeset_last_bool &&
2875       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2876     }
2877   }
2878   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2879   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2880 }
2881 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_extract_url_unexp:n

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

```

2882 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
2883 {
2884   \zref@ifpropundefined { urluse }
2885   { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2886   {
2887     \zref@ifrefcontainsprop {#1} { urluse }
2888     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
2889     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
2890   }
2891 }
2892 \cs_generate_variant:Nn \__zrefclever_extract_url_unexp:n { V }

```

(End definition for __zrefclever_extract_url_unexp:n.)

__zrefclever_labels_in_sequence:nn

Auxiliary function to `__zrefclever_typeset_refs_not_last_of_type:.` Sets `\l__zrefclever_next_maybe_range_bool` to true if $\langle label\ b \rangle$ comes in immediate sequence

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

```

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

2893 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2894 {
2895   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_a_tl
2896   {#1} { externaldocument } { \c_empty_tl }
2897   \__zrefclever_def_extract:Nnnn \l__zrefclever_label_extdoc_b_tl
2898   {#2} { externaldocument } { \c_empty_tl }
2899
2900   \tl_if_eq:NNT
2901   \l__zrefclever_label_extdoc_a_tl
2902   \l__zrefclever_label_extdoc_b_tl
2903   {
2904     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2905     {
2906       \exp_args:Nxx \tl_if_eq:nnT
2907       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
2908       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
2909       {
2910         \int_compare:nNnTF
2911         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
2912         =
2913         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2914         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2915         {
2916           \int_compare:nNnTF
2917           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
2918           =
2919           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
2920           {
2921             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2922             \bool_set_true:N \l__zrefclever_next_is_same_bool
2923           }
2924         }
2925       }
2926     }
2927   }
2928   \exp_args:Nxx \tl_if_eq:nnT
2929   { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
2930   { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
2931   {
2932     \exp_args:Nxx \tl_if_eq:nnT
2933     { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
2934     { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
2935     {
2936       \int_compare:nNnTF
2937       { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
2938       =

```

```

2939 { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2940 { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2941 {
2942   \int_compare:nNt
2943     { \_zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2944     =
2945     { \_zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2946     {
2947       \bool_set_true:N
2948         \l__zrefclever_next_maybe_range_bool
2949       \exp_args:Nxx \tl_if_eq:nnT
2950         {
2951           \_zrefclever_extract_unexp:nvn {#1}
2952             { l__zrefclever_ref_property_tl } { }
2953         }
2954         {
2955           \_zrefclever_extract_unexp:nvn {#2}
2956             { l__zrefclever_ref_property_tl } { }
2957         }
2958         {
2959           \bool_set_true:N
2960             \l__zrefclever_next_is_same_bool
2961         }
2962       }
2963     }
2964   }
2965 }
2966 }
2967 }
2968 }

```

(End definition for `_zrefclever_labels_in_sequence:nn`.)

Finally, a couple of functions for retrieving options values, according to the relevant precedence rules. They both receive an *<option>* as argument, and store the retrieved value in *<tl variable>*. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of `\l__zrefclever_label_type_a_tl`, as set during the processing of the label stack. This could be easily generalized, of course, but I don't think it is worth it, `\l__zrefclever_label_type_a_tl` is indeed what we want in all practical cases. The difference between `_zrefclever_get_ref_string:nN` and `_zrefclever_get_ref_font:nN` is the kind of option each should be used for. `_zrefclever_get_ref_string:nN` is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fall-back”). `_zrefclever_get_ref_font:nN` is intended for “font” options, which cannot be “language-specific”, thus for these we just search general options and type options.

```

\_zrefclever_get_ref_string:nN      \_zrefclever_get_ref_string:nN {<option>} {<tl variable>}
2969 \cs_new_protected:Npn \_zrefclever_get_ref_string:nN #1#2
2970 {
2971   % First attempt: general options.
2972   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2973   {
2974     % If not found, try type specific options.
2975     \bool_lazy_and:nnTF

```

```

2976     {
2977         \prop_if_exist_p:c
2978         {
2979             l__zrefclever_type_
2980             \l__zrefclever_label_type_a_tl _options_prop
2981         }
2982     }
2983     {
2984         \prop_if_in_p:cn
2985         {
2986             l__zrefclever_type_
2987             \l__zrefclever_label_type_a_tl _options_prop
2988         }
2989         {#1}
2990     }
2991     {
2992         \prop_get:cnN
2993         {
2994             l__zrefclever_type_
2995             \l__zrefclever_label_type_a_tl _options_prop
2996         }
2997         {#1} #2
2998     }
2999     {
3000         % If not found, try type specific translations.
3001         \__zrefclever_get_type_transl:xxnNF
3002         { \l__zrefclever_ref_language_tl }
3003         { \l__zrefclever_label_type_a_tl }
3004         {#1} #2
3005         {
3006             % If not found, try default translations.
3007             \__zrefclever_get_default_transl:xxnNF
3008             { \l__zrefclever_ref_language_tl }
3009             {#1} #2
3010             {
3011                 % If not found, try fallback.
3012                 \__zrefclever_get_fallback_transl:nNF {#1} #2
3013                 {
3014                     \tl_clear:N #2
3015                     \msg_warning:nnn { zref-clever }
3016                     { missing-string } {#1}
3017                 }
3018             }
3019         }
3020     }
3021 }
3022 }

```

(End definition for __zrefclever_get_ref_string:nN.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
3023 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
3024 {
3025     % First attempt: general options.

```

```

3026 \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3027 {
3028   % If not found, try type specific options.
3029   \bool_if:nTF
3030   {
3031     \prop_if_exist_p:c
3032     {
3033       l__zrefclever_type_
3034       \l__zrefclever_label_type_a_tl _options_prop
3035     }
3036   }
3037   {
3038     \prop_get:cnNF
3039     {
3040       l__zrefclever_type_
3041       \l__zrefclever_label_type_a_tl _options_prop
3042     }
3043     {#1} #2
3044     { \tl_clear:N #2 }
3045   }
3046   { \tl_clear:N #2 }
3047 }
3048 }

```

(End definition for `__zrefclever_get_ref_font:nN`.)

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.

```

3049 \__zrefclever_compat_module:nn { appendix }
3050 {

```



```

3051 \AddToHook { cmd / appendix / before }
3052 {
3053   \__zrefclever_zcsetup:n
3054   {
3055     countertype =
3056     {
3057       chapter      = appendix ,
3058       section      = appendix ,
3059       subsection   = appendix ,
3060       subsubsection = appendix ,
3061       paragraph    = appendix ,
3062       subparagraph = appendix ,
3063     }
3064   }
3065 }
3066 }

```

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

9.2 appendices

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

```

3067 \__zrefclever_compat_module:nn { appendices }
3068 {
3069   \__zrefclever_if_package_loaded:nT { appendix }
3070   {
3071     \newcounter { zc@appendix }
3072     \newcounter { zc@save@appendix }
3073     \setcounter { zc@appendix } { 0 }
3074     \setcounter { zc@save@appendix } { 0 }
3075     \cs_if_exist:cTF { chapter }
3076     {
3077       \__zrefclever_zcsetup:n
3078       { counterresetby = { chapter = zc@appendix } }
3079     }
3080     {
3081       \cs_if_exist:cT { section }
3082       {
3083         \__zrefclever_zcsetup:n
3084         { counterresetby = { section = zc@appendix } }
3085       }
3086     }
3087     \AddToHook { env / appendices / begin }
3088     {
3089       \stepcounter { zc@save@appendix }

```

```

3090         \setcounter { zc@appendix } { \value { zc@save@appendix } }
3091         \__zrefclever_zcsetup:n
3092         {
3093             countertype =
3094             {
3095                 chapter      = appendix ,
3096                 section      = appendix ,
3097                 subsection   = appendix ,
3098                 subsubsection = appendix ,
3099                 paragraph    = appendix ,
3100                 subparagraph = appendix ,
3101             }
3102         }
3103     }
3104     \AddToHook { env / appendices / end }
3105     { \setcounter { zc@appendix } { 0 } }
3106     \AddToHook { cmd / appendix / before }
3107     {
3108         \stepcounter { zc@save@appendix }
3109         \setcounter { zc@appendix } { \value { zc@save@appendix } }
3110     }
3111     \AddToHook { env / subappendices / begin }
3112     {
3113         \__zrefclever_zcsetup:n
3114         {
3115             countertype =
3116             {
3117                 section      = appendix ,
3118                 subsection   = appendix ,
3119                 subsubsection = appendix ,
3120                 paragraph    = appendix ,
3121                 subparagraph = appendix ,
3122             } ,
3123         }
3124     }
3125     \msg_info:nnn { zref-clever } { compat-package } { appendix }
3126 }
3127 }

```

9.3 memoir

The `memoir` document class has quite a number of cross-referencing related features, mostly dealing with captions, subfloats, and notes. Some of them are implemented in ways which make difficult the use of `zref`, particularly `\zlabel`, short of redefining the whole stuff ourselves. Hopefully, these features are specialized enough to make `zref-clever` useful enough with `memoir` without much friction, but unless some support is added upstream, it is difficult not to be a little intrusive here.

1. Caption functionality which receives $\langle label \rangle$ as optional argument, namely:
 - (a) The `sidecaption` and `sidecontcaption` environments. These environments *store* the label in an internal macro, `\m@mscaplabel`, at the begin environment code (more precisely in `\@@sidecaption`), but both the call

to `\refstepcounter` and the expansion of `\m@mscaplabel` take place at `\endsidecaption`. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the `\label` argument to start with. I can see two ways to deal with these environments, none of which I really like. First, map through `\m@mscaplabel` until `\label` is found, then grab the next token which is the `\label`. This can be used to set a `\zlabel` either with a kernel environment hook, or with `\@mem@scap@afterhook` (the former requires running `\refstepcounter` on our own, since the `env/.../end` hook comes before this is done by `\endsidecaption`). Second, locally redefine `\label` to set both labels inside the environments.

- (b) The bilingual caption commands: `\bitwonumcaption`, `\bionenumcaption`, and `\bicaption`. These commands do not support setting the label in their arguments (the labels do get set, but they end up included in the `title` property of the label too). So we do the same for them as for `sidecaption`, just taking care of grouping, since we can't count on the convenience of the environment hook (luckily for us, they are scoped themselves, so we can add an extra group there).
- 2. The `\subcaptionref` command, which makes a reference to the subcaption without the number of the main caption (e.g. “(b)”, instead of “2.3(b)”), for labels set inside the `\subcaption` argument of the subcaptioning commands, namely: `\subcaption`, `\contsubcaption`, `\subbottom`, `\contsubbottom`, `\subtop`. This functionality is implemented by `memoir` by setting a *second label* with prefix `sub@{label}`, and storing there just that part of interest. With `zref` this part is easier, since we can just add an extra property and retrieve it later on. The thing is that it is hard to find a place to hook into to add the property to the `main` list, since `memoir` does not really consider the possibility of some other command setting labels. `\@memsubcaption` is the best place to hook I could find. It is used by subcaptioning commands, and only those. And there is no hope for an environment hook in this case anyway.
- 3. `memoir`'s `\footnote`, `\verbfootnote`, `\sidefootnote` and `\pagenote`, just as the regular `\footnote` until recently in the kernel, do not set `\@currentcounter` alongside `\@currentlabel`, proper referencing to them requires setting the type for it.
- 4. Note that `memoir`'s appendix features “emulates” the `appendix` package, hence the corresponding compatibility module is loaded for `memoir` even if that package is not itself loaded. The same is true for the `\appendix` command module, since it is also defined.

```

3128 \__zrefclever_compat_module:nn { memoir }
3129 {
3130   \__zrefclever_if_class_loaded:nT { memoir }
3131   {

```

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`.

```

3132 \__zrefclever_zcsetup:n
3133 {
3134   countertype =
3135   {

```

```

3136         subfigure = figure ,
3137         subtable   = table ,
3138         poemline   = line ,
3139     } ,
3140     counterresetby =
3141     {
3142         subfigure = figure ,
3143         subtable   = table ,
3144     } ,
3145 }

```

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

```

3146 \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
3147 {
3148     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
3149     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
3150     {
3151         \__zrefclever_memoir_orig_label:n {##1}
3152         \zlabel{##1}
3153     }
3154     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
3155 }
3156 \AddToHook { env / sidecaption / begin }
3157 { \__zrefclever_memoir_both_labels: }
3158 \AddToHook { env / sidecontcaption / begin }
3159 { \__zrefclever_memoir_both_labels: }
3160 \AddToHook { cmd / bitwonumcaption / before }
3161 { \group_begin: \__zrefclever_memoir_both_labels: }
3162 \AddToHook { cmd / bitwonumcaption / after }
3163 { \group_end: }
3164 \AddToHook { cmd / bionenumcaption / before }
3165 { \group_begin: \__zrefclever_memoir_both_labels: }
3166 \AddToHook { cmd / bionenumcaption / after }
3167 { \group_end: }
3168 \AddToHook { cmd / bicaption / before }
3169 { \group_begin: \__zrefclever_memoir_both_labels: }
3170 \AddToHook { cmd / bicaption / after }
3171 { \group_end: }

```

Support for subcaption reference.

```

3172 \zref@newprop { subcaption }
3173 { \cs_if_exist_use:c { @@thesub \@captype } }
3174 \AddToHook { cmd / @memsubcaption / before }
3175 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

3176 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
3177 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
3178 \AddToHook { env / minipage / begin }
3179 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
3180 \AddToHook { cmd / @makefntext / before }
3181 {
3182     \__zrefclever_zcsetup:x
3183     { currentcounter = \l__zrefclever_memoir_footnote_type_tl }

```

```

3184     }
3185     \AddToHook { cmd / @makesidefntext / before }
3186     { \_zrefclever_zcsetup:n { currentcounter = sidefootnote } }
3187     \_zrefclever_zcsetup:n
3188     {
3189         countertype =
3190         {
3191             sidefootnote = footnote ,
3192             pagenote = endnote ,
3193         } ,
3194     }
3195     \AddToHook { file / \jobname.ent / before }
3196     { \_zrefclever_zcsetup:x { currentcounter = pagenote } }
3197     \msg_info:nnn { zref-clever } { compat-class } { memoir }
3198 }
3199 }

```

9.4 KOMA

Support for KOMA-Script document classes.

```

3200 \_zrefclever_compat_module:nn { KOMA }
3201 {
3202     \cs_if_exist:NT \KOMAClassName
3203     {

```

Add support for `captionbeside` and `captionofbeside` environments. These environments *do* run some variation of `\caption` and hence `\refstepcounter`. However, this happens inside a `parbox` inside the environment, thus grouped, such that we cannot see the variables set by `\refstepcounter` when we are setting the label. `\@currentlabel` is smuggled out of the group by KOMA, but the same care is not granted for `\@currentcounter`. So we have to rely on `\@capttype`, which the underlying caption infrastructure feeds to `\refstepcounter`. Since we must use `env/.../after` hooks, care should be taken not to set the `currentcounter` option unscoped, which would be quite disastrous. For this reason, though more “invasive”, we set `\@currentcounter` instead, which at least will be set straight the next time `\refstepcounter` runs. It sounds reasonable, it is the same treatment `\@currentlabel` is receiving in this case.

```

3204     \AddToHook { env / captionbeside / after }
3205     {
3206         \tl_if_exist:NT \@capttype
3207         { \tl_set_eq:NN \@currentcounter \@capttype }
3208     }
3209     \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
3210     \AddToHook { env / captionofbeside / end }
3211     { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
3212     \AddToHook { env / captionofbeside / after }
3213     {
3214         \tl_if_eq:NnF \@currenvir { document }
3215         {
3216             \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
3217             {
3218                 \tl_set_eq:NN
3219                 \@currentcounter \g__zrefclever_koma_capttype_tl
3220             }

```

```

3221         }
3222         \tl_gclear:N \g__zrefclever_koma_capttype_tl
3223     }
3224     \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
3225 }
3226 }

```

9.5 amsmath

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

```

3227 \__zrefclever_compat_module:nn { amsmath }
3228 {
3229     \__zrefclever_if_package_loaded:nT { amsmath }
3230     {

```

First, we define a function for label setting inside `amsmath` math environments, we want it to set both `\zlabel` and `\label`. We may “get a ride”, but not steal the place altogether. This makes for potentially redundant labels, but seems a good compromise. We *must* use the lower level `\zref@label` in this context, and hence also handle protection with `\zref@wrapper@babel`, because `\zlabel` makes itself no-op when `\label` is equal to `\ltx@gobble`, and that’s precisely the case inside the `multline` environment (and, damn!, I took a beating of this detail...).

```

3231         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
3232         {
3233             \__zrefclever_orig_ltxlabel:n {#1}
3234             \zref@wrapper@babel \zref@label {#1}
3235         }

```

Then we must store the original value of `\ltx@label`, which is the macro actually responsible for setting the labels inside `amsmath`’s math environments. And, after that, redefine it to be `__zrefclever_ltxlabel:n` instead. We must handle `hyperref` here, which comes very late in the preamble, and which loads `nameref` at `begindocument`, which in turn, lets `\ltx@label` be `\label`. This has to come after `nameref`. Other classes packages also redefine `\ltx@label`, which may cause some trouble. A `grep` on `texmf-dist` returns hits for: `thm-restate.sty`, `smartref.sty`, `jmlrbook.cls`, `cleveref.sty`, `cryptocode.sty`, `nameref.sty`, `easyeqn.sty`, `empheq.sty`, `ntheorem.sty`, `nccmath.sty`, `nwejm.cls`, `nwejmart.cls`, `aguplus.sty`, `aguplus.cls`, `agupp.sty`, `amsmath.hyp`, `amsmath.sty` (surprise!), `amsmath.4ht`, `nameref.4ht`, `frenchle.sty`, `french.sty`, plus corresponding documentations and different versions of the same packages. That’s not too many, but not “just a few” either. The critical ones are explicitly handled here (`amsmath` itself, and `nameref`). A number of those I’m really not acquainted with. For `cleveref`, in particular, this procedure is not compatible with it. If we happen to come later than it and override its definition, this may be a substantial problem for `cleveref`, since it will find the label, but it won’t contain the data it is expecting. However, this should normally not occur, if the user has followed the documented recommendation for `cleveref` to load it last, or at least very late, and besides I don’t see much of an use case for using both `cleveref` and `zref-clever` together. I have documented in the user manual that this module may cause potential issues, and how to work around them. And I have made an upstream feature request for a hook, so that this could be made more cleanly at <https://github.com/latex3/hyperref/issues/212>.

```

3236         \__zrefclever_if_package_loaded:nTF { hyperref }
3237         {

```

```

3238     \AddToHook { package / nameref / after }
3239     {
3240         \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3241         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3242     }
3243 }
3244 {
3245     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3246     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3247 }

```

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).

```

3248     \AddToHook { env / subequations / begin }
3249     {
3250         \__zrefclever_zcsetup:x
3251         {
3252             counterresetby =
3253             {
3254                 parentequation =
3255                 \__zrefclever_counter_reset_by:n { equation } ,
3256                 equation = parentequation ,
3257             } ,
3258             currentcounter = parentequation ,
3259             countertype = { parentequation = equation } ,
3260         }
3261     }

```

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

```

3262     \clist_map_inline:nn
3263     {
3264         equation ,
3265         equation* ,
3266         align ,

```

```

3267         align* ,
3268         alignat ,
3269         alignat* ,
3270         flalign ,
3271         flalign* ,
3272         xalignat ,
3273         xalignat* ,
3274         gather ,
3275         gather* ,
3276         multiline ,
3277         multiline* ,
3278     }
3279     {
3280         \AddToHook { env / #1 / begin }
3281             { \__zrefclever_zcsetup:n { currentcounter = equation } }
3282     }

```

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

```

3283     \zcRefTypeSetup { equation }
3284     {
3285         reffont = \upshape ,
3286         refpre  = {\textup{({}} } ,
3287         refpos  = {\textup{)}}} ,
3288     }
3289     \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3290 }
3291 }

```

9.6 mathtools

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

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

```

3292 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3293 \__zrefclever_compat_module:nn { mathtools }
3294 {
3295     \__zrefclever_if_package_loaded:nT { mathtools }
3296     {
3297         \MH_if_boolean:nT { show_only_refs }
3298         {
3299             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3300             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
3301             {
3302                 \@bsphack
3303                 \seq_map_inline:Nn #1
3304                 {

```



```

3305         \exp_args:Nx \tl_if_eq:nnTF
3306         { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3307         { equation }
3308         {
3309             \protected@write \@auxout { }
3310             { \string \MT@newlabel {##1} }
3311         }
3312         {
3313             \exp_args:Nx \tl_if_eq:nnT
3314             { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3315             { parentequation }
3316             {
3317                 \protected@write \@auxout { }
3318                 { \string \MT@newlabel {##1} }
3319             }
3320         }
3321     }
3322     \@esphack
3323 }
3324 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3325 }
3326 }
3327 }

```

9.7 breqn

From the `breqn` documentation: “Use of the normal `\label` command instead of the `label` option works, I think, most of the time (untested)”. Indeed, light testing suggests it does work for `\zlabel` just as well. However, if it happens not to work, there was no easy alternative handle I could find. In particular, it does not seem viable to leverage the `label=` option without hacking the package internals, even if the case of doing so would not be specially tricky, just “not very civil”.

```

3328 \__zrefclever_compat_module:nn { breqn }
3329 {
3330     \__zrefclever_if_package_loaded:nT { breqn }
3331     {

```

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>).

```

3332     \AddToHook { env / dgroup / begin }
3333     {
3334         \__zrefclever_zcsetup:x
3335         {
3336             counterresetby =
3337             {
3338                 parentequation =
3339                 \__zrefclever_counter_reset_by:n { equation } ,
3340                 equation = parentequation ,
3341             } ,

```

```

3342         currentcounter = parentequation ,
3343         countertype = { parentequation = equation } ,
3344     }
3345 }
3346 \clist_map_inline:nn
3347 {
3348     dmath ,
3349     dseries ,
3350     darray ,
3351 }
3352 {
3353     \AddToHook { env / #1 / begin }
3354     { \__zrefclever_zcsetup:n { currentcounter = equation } }
3355 }
3356 \msg_info:nnn { zref-clever } { compat-package } { breqn }
3357 }
3358 }

```

9.8 listings

```

3359 \__zrefclever_compat_module:nn { listings }
3360 {
3361     \__zrefclever_if_package_loaded:nT { listings }
3362     {
3363         \__zrefclever_zcsetup:n
3364         {
3365             countertype =
3366             {
3367                 lstlisting = listing ,
3368                 lstnumber = line ,
3369             } ,
3370             counterresetby = { lstnumber = lstlisting } ,
3371         }
3372     }
3373 }

```

Set (also) a `\zlabel` with the label received in the `label=` option from the `lstlisting` environment. The *only* place to set this label is the `PreInit` hook. This hook, comes right after `\lst@MakeCaption` in `\lst@Init`, which runs `\refstepcounter` on `lstlisting`, so we must come after it. Also `listings` itself sets `\@currentlabel` to `\thelstnumber` in the `Init` hook, which comes right after the `PreInit` one in `\lst@Init`. Since, if we add to `Init` here, we go to the end of it, we'd be seeing the wrong `\@currentlabel` at that point.

```

3372 \lst@AddToHook { PreInit }
3373 { \tl_if_empty:NF \lst@label { \zlabel { \lst@label } } }

```

Set `currentcounter` to `lstnumber` in the `Init` hook, since `listings` itself sets `\@currentlabel` to `\thelstnumber` here. Note that `listings` *does use* `\refstepcounter` on `lstnumber`, but does so in the `EveryPar` hook, and there must be some grouping involved such that `\@currentcounter` ends up not being visible to the label. See section “Line numbers” of ‘`texdoc listings-devel`’ (the `.dtx`), and search for the definition of macro `\c@lstnumber`. Indeed, the fact that `listings` manually sets `\@currentlabel` to `\thelstnumber` is a signal that the work of `\refstepcounter` is being restrained somehow.

```

3374 \lst@AddToHook { Init }
3375 { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }

```

```

3376         \msg_info:nnn { zref-clever } { compat-package } { listings }
3377     }
3378 }

```

9.9 enumitem

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

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

```

3379 \__zrefclever_compat_module:nn { enumitem }
3380 {
3381     \__zrefclever_if_package_loaded:nT { enumitem }
3382     {
3383         \int_set:Nn \l_tmpa_int { 5 }
3384         \bool_while_do:nn
3385         {
3386             \cs_if_exist_p:c
3387             { c@ enum \int_to_roman:n { \l_tmpa_int } }
3388         }
3389         {
3390             \__zrefclever_zcsetup:x
3391             {
3392                 counterresetby =
3393                 {
3394                     enum \int_to_roman:n { \l_tmpa_int } =
3395                     enum \int_to_roman:n { \l_tmpa_int - 1 }
3396                 } ,
3397                 countertype =
3398                 { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3399             }
3400             \int_incr:N \l_tmpa_int
3401         }
3402         \int_compare:nNnT { \l_tmpa_int } > { 5 }
3403         { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3404     }
3405 }

```

9.10 subcaption

```

3406 \__zrefclever_compat_module:nn { subcaption }
3407 {
3408     \__zrefclever_if_package_loaded:nT { subcaption }
3409     {

```

```

3410     \__zrefclever_zcsetup:n
3411     {
3412         countertype =
3413         {
3414             subfigure = figure ,
3415             subtable = table ,
3416         } ,
3417         counterresetby =
3418         {
3419             subfigure = figure ,
3420             subtable = table ,
3421         } ,
3422     }

```

Support for `subref` reference.

```

3423     \zref@newprop { subref }
3424     { \cs_if_exist_use:c { thesub \@captype } }
3425     \tl_put_right:Nn \caption@subtypetool
3426     { \zref@localaddprop \ZREF@mainlist { subref } }
3427 }
3428 }

```

9.11 subfig

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

```

3429 \__zrefclever_compat_module:nn { subfig }
3430 {
3431     \__zrefclever_if_package_loaded:nT { subfig }
3432     {
3433         \__zrefclever_zcsetup:n
3434         {
3435             countertype =
3436             {
3437                 subfigure = figure ,
3438                 subtable = table ,
3439             } ,
3440             counterresetby =
3441             {
3442                 subfigure = figure ,
3443                 subtable = table ,
3444             } ,
3445         }
3446     }
3447 }
3448 </package>

```

10 Dictionaries

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

10.1 English

English dictionary has been initially provided by the author.

```
3449 <*package>
3450 \zcDeclareLanguage { english }
3451 \zcDeclareLanguageAlias { american } { english }
3452 \zcDeclareLanguageAlias { australian } { english }
3453 \zcDeclareLanguageAlias { british } { english }
3454 \zcDeclareLanguageAlias { canadian } { english }
3455 \zcDeclareLanguageAlias { newzealand } { english }
3456 \zcDeclareLanguageAlias { UKenglish } { english }
3457 \zcDeclareLanguageAlias { USenglish } { english }
3458 </package>
3459 <*dict-english>
3460 namesep = {\nobreakspace} ,
3461 pairsep = {\~and\nobreakspace} ,
3462 listsep = {\~,~} ,
3463 lastsep = {\~and\nobreakspace} ,
3464 tpairsep = {\~and\nobreakspace} ,
3465 tlistsep = {\~,~} ,
3466 tlastsep = {\~,~and\nobreakspace} ,
3467 notesep = {\~} ,
3468 rangesep = {\~to\nobreakspace} ,
3469
3470 type = book ,
3471   Name-sg = Book ,
3472   name-sg = book ,
3473   Name-pl = Books ,
3474   name-pl = books ,
3475
3476 type = part ,
3477   Name-sg = Part ,
3478   name-sg = part ,
3479   Name-pl = Parts ,
3480   name-pl = parts ,
3481
3482 type = chapter ,
3483   Name-sg = Chapter ,
3484   name-sg = chapter ,
3485   Name-pl = Chapters ,
3486   name-pl = chapters ,
3487
3488 type = section ,
3489   Name-sg = Section ,
3490   name-sg = section ,
3491   Name-pl = Sections ,
3492   name-pl = sections ,
3493
3494 type = paragraph ,
3495   Name-sg = Paragraph ,
3496   name-sg = paragraph ,
3497   Name-pl = Paragraphs ,
3498   name-pl = paragraphs ,
```

```

3499   Name-sg-ab = Par. ,
3500   name-sg-ab = par. ,
3501   Name-pl-ab = Par. ,
3502   name-pl-ab = par. ,
3503
3504   type = appendix ,
3505       Name-sg = Appendix ,
3506       name-sg = appendix ,
3507       Name-pl = Appendices ,
3508       name-pl = appendices ,
3509
3510   type = page ,
3511       Name-sg = Page ,
3512       name-sg = page ,
3513       Name-pl = Pages ,
3514       name-pl = pages ,
3515       name-sg-ab = p. ,
3516       name-pl-ab = pp. ,
3517       rangesep = {\textendash} ,
3518
3519   type = line ,
3520       Name-sg = Line ,
3521       name-sg = line ,
3522       Name-pl = Lines ,
3523       name-pl = lines ,
3524
3525   type = figure ,
3526       Name-sg = Figure ,
3527       name-sg = figure ,
3528       Name-pl = Figures ,
3529       name-pl = figures ,
3530       Name-sg-ab = Fig. ,
3531       name-sg-ab = fig. ,
3532       Name-pl-ab = Figs. ,
3533       name-pl-ab = figs. ,
3534
3535   type = table ,
3536       Name-sg = Table ,
3537       name-sg = table ,
3538       Name-pl = Tables ,
3539       name-pl = tables ,
3540
3541   type = item ,
3542       Name-sg = Item ,
3543       name-sg = item ,
3544       Name-pl = Items ,
3545       name-pl = items ,
3546
3547   type = footnote ,
3548       Name-sg = Footnote ,
3549       name-sg = footnote ,
3550       Name-pl = Footnotes ,
3551       name-pl = footnotes ,
3552

```

```

3553 type = endnote ,
3554     Name-sg = Note ,
3555     name-sg = note ,
3556     Name-pl = Notes ,
3557     name-pl = notes ,
3558
3559 type = note ,
3560     Name-sg = Note ,
3561     name-sg = note ,
3562     Name-pl = Notes ,
3563     name-pl = notes ,
3564
3565 type = equation ,
3566     Name-sg = Equation ,
3567     name-sg = equation ,
3568     Name-pl = Equations ,
3569     name-pl = equations ,
3570     Name-sg-ab = Eq. ,
3571     name-sg-ab = eq. ,
3572     Name-pl-ab = Eqs. ,
3573     name-pl-ab = eqs. ,
3574     refpre = {} ,
3575     refpos = {} ,
3576
3577 type = theorem ,
3578     Name-sg = Theorem ,
3579     name-sg = theorem ,
3580     Name-pl = Theorems ,
3581     name-pl = theorems ,
3582
3583 type = lemma ,
3584     Name-sg = Lemma ,
3585     name-sg = lemma ,
3586     Name-pl = Lemmas ,
3587     name-pl = lemmas ,
3588
3589 type = corollary ,
3590     Name-sg = Corollary ,
3591     name-sg = corollary ,
3592     Name-pl = Corollaries ,
3593     name-pl = corollaries ,
3594
3595 type = proposition ,
3596     Name-sg = Proposition ,
3597     name-sg = proposition ,
3598     Name-pl = Propositions ,
3599     name-pl = propositions ,
3600
3601 type = definition ,
3602     Name-sg = Definition ,
3603     name-sg = definition ,
3604     Name-pl = Definitions ,
3605     name-pl = definitions ,
3606

```

```

3607 type = proof ,
3608   Name-sg = Proof ,
3609   name-sg = proof ,
3610   Name-pl = Proofs ,
3611   name-pl = proofs ,
3612
3613 type = result ,
3614   Name-sg = Result ,
3615   name-sg = result ,
3616   Name-pl = Results ,
3617   name-pl = results ,
3618
3619 type = remark ,
3620   Name-sg = Remark ,
3621   name-sg = remark ,
3622   Name-pl = Remarks ,
3623   name-pl = remarks ,
3624
3625 type = example ,
3626   Name-sg = Example ,
3627   name-sg = example ,
3628   Name-pl = Examples ,
3629   name-pl = examples ,
3630
3631 type = algorithm ,
3632   Name-sg = Algorithm ,
3633   name-sg = algorithm ,
3634   Name-pl = Algorithms ,
3635   name-pl = algorithms ,
3636
3637 type = listing ,
3638   Name-sg = Listing ,
3639   name-sg = listing ,
3640   Name-pl = Listings ,
3641   name-pl = listings ,
3642
3643 type = exercise ,
3644   Name-sg = Exercise ,
3645   name-sg = exercise ,
3646   Name-pl = Exercises ,
3647   name-pl = exercises ,
3648
3649 type = solution ,
3650   Name-sg = Solution ,
3651   name-sg = solution ,
3652   Name-pl = Solutions ,
3653   name-pl = solutions ,
3654 </dict-english>

```

10.2 German

German dictionary has been initially provided by the author.

```

3655 <*package>

```



```

3656 \zcDeclareLanguage
3657 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]
3658 { german }
3659 \zcDeclareLanguageAlias { austrian      } { german }
3660 \zcDeclareLanguageAlias { germanb       } { german }
3661 \zcDeclareLanguageAlias { ngerman       } { german }
3662 \zcDeclareLanguageAlias { naustrian     } { german }
3663 \zcDeclareLanguageAlias { nswissgerman } { german }
3664 \zcDeclareLanguageAlias { swissgerman  } { german }
3665 \package
3666 <*dict-german>

3667 namesep = {\nobreakspace} ,
3668 pairsep = {\simund\nobreakspace} ,
3669 listsep = { , \sim } ,
3670 lastsep = {\simund\nobreakspace} ,
3671 tpairsep = {\simund\nobreakspace} ,
3672 tlistsep = { , \sim } ,
3673 tlastsep = {\simund\nobreakspace} ,
3674 notesep = { \sim } ,
3675 rangesep = {\simbis\nobreakspace} ,
3676
3677 type = book ,
3678 gender = n ,
3679 case = N ,
3680     Name-sg = Buch ,
3681     Name-pl = Bücher ,
3682 case = A ,
3683     Name-sg = Buch ,
3684     Name-pl = Bücher ,
3685 case = D ,
3686     Name-sg = Buch ,
3687     Name-pl = Büchern ,
3688 case = G ,
3689     Name-sg = Buches ,
3690     Name-pl = Bücher ,
3691
3692 type = part ,
3693 gender = m ,
3694 case = N ,
3695     Name-sg = Teil ,
3696     Name-pl = Teile ,
3697 case = A ,
3698     Name-sg = Teil ,
3699     Name-pl = Teile ,
3700 case = D ,
3701     Name-sg = Teil ,
3702     Name-pl = Teilen ,
3703 case = G ,
3704     Name-sg = Teiles ,
3705     Name-pl = Teile ,
3706
3707 type = chapter ,
3708 gender = n ,

```

```

3709 case = N ,
3710     Name-sg = Kapitel ,
3711     Name-pl = Kapitel ,
3712 case = A ,
3713     Name-sg = Kapitel ,
3714     Name-pl = Kapitel ,
3715 case = D ,
3716     Name-sg = Kapitel ,
3717     Name-pl = Kapiteln ,
3718 case = G ,
3719     Name-sg = Kapitels ,
3720     Name-pl = Kapitel ,
3721
3722 type = section ,
3723     gender = m ,
3724     case = N ,
3725         Name-sg = Abschnitt ,
3726         Name-pl = Abschnitte ,
3727     case = A ,
3728         Name-sg = Abschnitt ,
3729         Name-pl = Abschnitte ,
3730     case = D ,
3731         Name-sg = Abschnitt ,
3732         Name-pl = Abschnitten ,
3733     case = G ,
3734         Name-sg = Abschnitts ,
3735         Name-pl = Abschnitte ,
3736
3737 type = paragraph ,
3738     gender = m ,
3739     case = N ,
3740         Name-sg = Absatz ,
3741         Name-pl = Absätze ,
3742     case = A ,
3743         Name-sg = Absatz ,
3744         Name-pl = Absätze ,
3745     case = D ,
3746         Name-sg = Absatz ,
3747         Name-pl = Absätzen ,
3748     case = G ,
3749         Name-sg = Absatzes ,
3750         Name-pl = Absätze ,
3751
3752 type = appendix ,
3753     gender = m ,
3754     case = N ,
3755         Name-sg = Anhang ,
3756         Name-pl = Anhänge ,
3757     case = A ,
3758         Name-sg = Anhang ,
3759         Name-pl = Anhänge ,
3760     case = D ,
3761         Name-sg = Anhang ,
3762         Name-pl = Anhängen ,

```

```

3763 case = G ,
3764     Name-sg = Anhangs ,
3765     Name-pl = Anhänge ,
3766
3767 type = page ,
3768     gender = f ,
3769     case = N ,
3770     Name-sg = Seite ,
3771     Name-pl = Seiten ,
3772     case = A ,
3773     Name-sg = Seite ,
3774     Name-pl = Seiten ,
3775     case = D ,
3776     Name-sg = Seite ,
3777     Name-pl = Seiten ,
3778     case = G ,
3779     Name-sg = Seite ,
3780     Name-pl = Seiten ,
3781     rangesep = {\textendash} ,
3782
3783 type = line ,
3784     gender = f ,
3785     case = N ,
3786     Name-sg = Zeile ,
3787     Name-pl = Zeilen ,
3788     case = A ,
3789     Name-sg = Zeile ,
3790     Name-pl = Zeilen ,
3791     case = D ,
3792     Name-sg = Zeile ,
3793     Name-pl = Zeilen ,
3794     case = G ,
3795     Name-sg = Zeile ,
3796     Name-pl = Zeilen ,
3797
3798 type = figure ,
3799     gender = f ,
3800     case = N ,
3801     Name-sg = Abbildung ,
3802     Name-pl = Abbildungen ,
3803     Name-sg-ab = Abb. ,
3804     Name-pl-ab = Abb. ,
3805     case = A ,
3806     Name-sg = Abbildung ,
3807     Name-pl = Abbildungen ,
3808     Name-sg-ab = Abb. ,
3809     Name-pl-ab = Abb. ,
3810     case = D ,
3811     Name-sg = Abbildung ,
3812     Name-pl = Abbildungen ,
3813     Name-sg-ab = Abb. ,
3814     Name-pl-ab = Abb. ,
3815     case = G ,
3816     Name-sg = Abbildung ,

```

```

3817     Name-pl = Abbildungen ,
3818     Name-sg-ab = Abb. ,
3819     Name-pl-ab = Abb. ,
3820
3821 type = table ,
3822     gender = f ,
3823     case = N ,
3824     Name-sg = Tabelle ,
3825     Name-pl = Tabellen ,
3826     case = A ,
3827     Name-sg = Tabelle ,
3828     Name-pl = Tabellen ,
3829     case = D ,
3830     Name-sg = Tabelle ,
3831     Name-pl = Tabellen ,
3832     case = G ,
3833     Name-sg = Tabelle ,
3834     Name-pl = Tabellen ,
3835
3836 type = item ,
3837     gender = m ,
3838     case = N ,
3839     Name-sg = Punkt ,
3840     Name-pl = Punkte ,
3841     case = A ,
3842     Name-sg = Punkt ,
3843     Name-pl = Punkte ,
3844     case = D ,
3845     Name-sg = Punkt ,
3846     Name-pl = Punkten ,
3847     case = G ,
3848     Name-sg = Punktes ,
3849     Name-pl = Punkte ,
3850
3851 type = footnote ,
3852     gender = f ,
3853     case = N ,
3854     Name-sg = Fußnote ,
3855     Name-pl = Fußnoten ,
3856     case = A ,
3857     Name-sg = Fußnote ,
3858     Name-pl = Fußnoten ,
3859     case = D ,
3860     Name-sg = Fußnote ,
3861     Name-pl = Fußnoten ,
3862     case = G ,
3863     Name-sg = Fußnote ,
3864     Name-pl = Fußnoten ,
3865
3866 type = endnote ,
3867     gender = f ,
3868     case = N ,
3869     Name-sg = Endnote ,
3870     Name-pl = Endnoten ,

```

```

3871 case = A ,
3872     Name-sg = Endnote ,
3873     Name-pl = Endnoten ,
3874 case = D ,
3875     Name-sg = Endnote ,
3876     Name-pl = Endnoten ,
3877 case = G ,
3878     Name-sg = Endnote ,
3879     Name-pl = Endnoten ,
3880
3881 type = note ,
3882     gender = f ,
3883     case = N ,
3884         Name-sg = Anmerkung ,
3885         Name-pl = Anmerkungen ,
3886 case = A ,
3887     Name-sg = Anmerkung ,
3888     Name-pl = Anmerkungen ,
3889 case = D ,
3890     Name-sg = Anmerkung ,
3891     Name-pl = Anmerkungen ,
3892 case = G ,
3893     Name-sg = Anmerkung ,
3894     Name-pl = Anmerkungen ,
3895
3896 type = equation ,
3897     gender = f ,
3898     case = N ,
3899         Name-sg = Gleichung ,
3900         Name-pl = Gleichungen ,
3901 case = A ,
3902     Name-sg = Gleichung ,
3903     Name-pl = Gleichungen ,
3904 case = D ,
3905     Name-sg = Gleichung ,
3906     Name-pl = Gleichungen ,
3907 case = G ,
3908     Name-sg = Gleichung ,
3909     Name-pl = Gleichungen ,
3910 refpre = {() ,
3911 refpos = {}} ,
3912
3913 type = theorem ,
3914     gender = n ,
3915     case = N ,
3916         Name-sg = Theorem ,
3917         Name-pl = Theoreme ,
3918 case = A ,
3919     Name-sg = Theorem ,
3920     Name-pl = Theoreme ,
3921 case = D ,
3922     Name-sg = Theorem ,
3923     Name-pl = Theoremen ,
3924 case = G ,

```

```

3925     Name-sg = Theorems ,
3926     Name-pl = Theoreme ,
3927
3928 type = lemma ,
3929     gender = n ,
3930     case = N ,
3931     Name-sg = Lemma ,
3932     Name-pl = Lemmata ,
3933     case = A ,
3934     Name-sg = Lemma ,
3935     Name-pl = Lemmata ,
3936     case = D ,
3937     Name-sg = Lemma ,
3938     Name-pl = Lemmata ,
3939     case = G ,
3940     Name-sg = Lemmas ,
3941     Name-pl = Lemmata ,
3942
3943 type = corollary ,
3944     gender = n ,
3945     case = N ,
3946     Name-sg = Korollar ,
3947     Name-pl = Korollare ,
3948     case = A ,
3949     Name-sg = Korollar ,
3950     Name-pl = Korollare ,
3951     case = D ,
3952     Name-sg = Korollar ,
3953     Name-pl = Korollaren ,
3954     case = G ,
3955     Name-sg = Korollars ,
3956     Name-pl = Korollare ,
3957
3958 type = proposition ,
3959     gender = m ,
3960     case = N ,
3961     Name-sg = Satz ,
3962     Name-pl = Sätze ,
3963     case = A ,
3964     Name-sg = Satz ,
3965     Name-pl = Sätze ,
3966     case = D ,
3967     Name-sg = Satz ,
3968     Name-pl = Sätzen ,
3969     case = G ,
3970     Name-sg = Satzes ,
3971     Name-pl = Sätze ,
3972
3973 type = definition ,
3974     gender = f ,
3975     case = N ,
3976     Name-sg = Definition ,
3977     Name-pl = Definitionen ,
3978     case = A ,

```

```

3979     Name-sg = Definition ,
3980     Name-pl = Definitionen ,
3981     case = D ,
3982     Name-sg = Definition ,
3983     Name-pl = Definitionen ,
3984     case = G ,
3985     Name-sg = Definition ,
3986     Name-pl = Definitionen ,
3987
3988 type = proof ,
3989     gender = m ,
3990     case = N ,
3991     Name-sg = Beweis ,
3992     Name-pl = Beweise ,
3993     case = A ,
3994     Name-sg = Beweis ,
3995     Name-pl = Beweise ,
3996     case = D ,
3997     Name-sg = Beweis ,
3998     Name-pl = Beweisen ,
3999     case = G ,
4000     Name-sg = Beweises ,
4001     Name-pl = Beweise ,
4002
4003 type = result ,
4004     gender = n ,
4005     case = N ,
4006     Name-sg = Ergebnis ,
4007     Name-pl = Ergebnisse ,
4008     case = A ,
4009     Name-sg = Ergebnis ,
4010     Name-pl = Ergebnisse ,
4011     case = D ,
4012     Name-sg = Ergebnis ,
4013     Name-pl = Ergebnissen ,
4014     case = G ,
4015     Name-sg = Ergebnisses ,
4016     Name-pl = Ergebnisse ,
4017
4018 type = remark ,
4019     gender = f ,
4020     case = N ,
4021     Name-sg = Bemerkung ,
4022     Name-pl = Bemerkungen ,
4023     case = A ,
4024     Name-sg = Bemerkung ,
4025     Name-pl = Bemerkungen ,
4026     case = D ,
4027     Name-sg = Bemerkung ,
4028     Name-pl = Bemerkungen ,
4029     case = G ,
4030     Name-sg = Bemerkung ,
4031     Name-pl = Bemerkungen ,
4032

```

```

4033 type = example ,
4034     gender = n ,
4035     case = N ,
4036         Name-sg = Beispiel ,
4037         Name-pl = Beispiele ,
4038     case = A ,
4039         Name-sg = Beispiel ,
4040         Name-pl = Beispiele ,
4041     case = D ,
4042         Name-sg = Beispiel ,
4043         Name-pl = Beispielen ,
4044     case = G ,
4045         Name-sg = Beispiels ,
4046         Name-pl = Beispiele ,
4047
4048 type = algorithm ,
4049     gender = m ,
4050     case = N ,
4051         Name-sg = Algorithmus ,
4052         Name-pl = Algorithmen ,
4053     case = A ,
4054         Name-sg = Algorithmus ,
4055         Name-pl = Algorithmen ,
4056     case = D ,
4057         Name-sg = Algorithmus ,
4058         Name-pl = Algorithmen ,
4059     case = G ,
4060         Name-sg = Algorithmus ,
4061         Name-pl = Algorithmen ,
4062
4063 type = listing ,
4064     gender = n ,
4065     case = N ,
4066         Name-sg = Listing ,
4067         Name-pl = Listings ,
4068     case = A ,
4069         Name-sg = Listing ,
4070         Name-pl = Listings ,
4071     case = D ,
4072         Name-sg = Listing ,
4073         Name-pl = Listings ,
4074     case = G ,
4075         Name-sg = Listings ,
4076         Name-pl = Listings ,
4077
4078 type = exercise ,
4079     gender = f ,
4080     case = N ,
4081         Name-sg = Übungsaufgabe ,
4082         Name-pl = Übungsaufgaben ,
4083     case = A ,
4084         Name-sg = Übungsaufgabe ,
4085         Name-pl = Übungsaufgaben ,
4086     case = D ,

```



```

4087     Name-sg = Übungsaufgabe ,
4088     Name-pl = Übungsaufgaben ,
4089     case = G ,
4090     Name-sg = Übungsaufgabe ,
4091     Name-pl = Übungsaufgaben ,
4092
4093     type = solution ,
4094     gender = f ,
4095     case = N ,
4096     Name-sg = Lösung ,
4097     Name-pl = Lösungen ,
4098     case = A ,
4099     Name-sg = Lösung ,
4100     Name-pl = Lösungen ,
4101     case = D ,
4102     Name-sg = Lösung ,
4103     Name-pl = Lösungen ,
4104     case = G ,
4105     Name-sg = Lösung ,
4106     Name-pl = Lösungen ,
4107 </dict-german>

```

10.3 French

French dictionary has been initially provided by the author, and has been improved thanks to Denis Bitouzé (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.

```

4108 <*package>
4109 \zcDeclareLanguage [ gender = { f , m } ] { french }
4110 \zcDeclareLanguageAlias { acadian } { french }
4111 \zcDeclareLanguageAlias { canadien } { french }
4112 \zcDeclareLanguageAlias { francais } { french }
4113 \zcDeclareLanguageAlias { frenchb } { french }
4114 </package>
4115 <*dict-french>
4116 namesep = {\nobreakspace} ,
4117 pairsep = {\~et\nobreakspace} ,
4118 listsep = { , ~ } ,
4119 lastsep = {\~et\nobreakspace} ,
4120 tpairsep = {\~et\nobreakspace} ,
4121 tlistsep = { , ~ } ,
4122 tlastsep = {\~et\nobreakspace} ,
4123 notesep = { ~ } ,
4124 rangesep = {\~à\nobreakspace} ,
4125
4126 type = book ,
4127 gender = m ,
4128 Name-sg = Livre ,
4129 name-sg = livre ,
4130 Name-pl = Livres ,

```

```

4131     name-pl = livres ,
4132
4133     type = part ,
4134     gender = f ,
4135     Name-sg = Partie ,
4136     name-sg = partie ,
4137     Name-pl = Parties ,
4138     name-pl = parties ,
4139
4140     type = chapter ,
4141     gender = m ,
4142     Name-sg = Chapitre ,
4143     name-sg = chapitre ,
4144     Name-pl = Chapitres ,
4145     name-pl = chapitres ,
4146
4147     type = section ,
4148     gender = f ,
4149     Name-sg = Section ,
4150     name-sg = section ,
4151     Name-pl = Sections ,
4152     name-pl = sections ,
4153
4154     type = paragraph ,
4155     gender = m ,
4156     Name-sg = Paragraphe ,
4157     name-sg = paragraphe ,
4158     Name-pl = Paragraphes ,
4159     name-pl = paragraphes ,
4160
4161     type = appendix ,
4162     gender = f ,
4163     Name-sg = Annexe ,
4164     name-sg = annexe ,
4165     Name-pl = Annexes ,
4166     name-pl = annexes ,
4167
4168     type = page ,
4169     gender = f ,
4170     Name-sg = Page ,
4171     name-sg = page ,
4172     Name-pl = Pages ,
4173     name-pl = pages ,
4174     rangesep = {-} ,
4175
4176     type = line ,
4177     gender = f ,
4178     Name-sg = Ligne ,
4179     name-sg = ligne ,
4180     Name-pl = Lignes ,
4181     name-pl = lignes ,
4182
4183     type = figure ,
4184     gender = f ,

```

```

4185 Name-sg = Figure ,
4186 name-sg = figure ,
4187 Name-pl = Figures ,
4188 name-pl = figures ,
4189
4190 type = table ,
4191 gender = f ,
4192 Name-sg = Table ,
4193 name-sg = table ,
4194 Name-pl = Tables ,
4195 name-pl = tables ,
4196
4197 type = item ,
4198 gender = m ,
4199 Name-sg = Point ,
4200 name-sg = point ,
4201 Name-pl = Points ,
4202 name-pl = points ,
4203
4204 type = footnote ,
4205 gender = f ,
4206 Name-sg = Note ,
4207 name-sg = note ,
4208 Name-pl = Notes ,
4209 name-pl = notes ,
4210
4211 type = endnote ,
4212 gender = f ,
4213 Name-sg = Note ,
4214 name-sg = note ,
4215 Name-pl = Notes ,
4216 name-pl = notes ,
4217
4218 type = note ,
4219 gender = f ,
4220 Name-sg = Note ,
4221 name-sg = note ,
4222 Name-pl = Notes ,
4223 name-pl = notes ,
4224
4225 type = equation ,
4226 gender = f ,
4227 Name-sg = Équation ,
4228 name-sg = équation ,
4229 Name-pl = Équations ,
4230 name-pl = équations ,
4231 refpre = {} ,
4232 refpos = {} ,
4233
4234 type = theorem ,
4235 gender = m ,
4236 Name-sg = Théorème ,
4237 name-sg = théorème ,
4238 Name-pl = Théorèmes ,

```

```

4239     name-pl = théorèmes ,
4240
4241 type = lemma ,
4242     gender = m ,
4243     Name-sg = Lemme ,
4244     name-sg = lemme ,
4245     Name-pl = Lemmes ,
4246     name-pl = lemmes ,
4247
4248 type = corollary ,
4249     gender = m ,
4250     Name-sg = Corollaire ,
4251     name-sg = corollaire ,
4252     Name-pl = Corollaires ,
4253     name-pl = corollaires ,
4254
4255 type = proposition ,
4256     gender = f ,
4257     Name-sg = Proposition ,
4258     name-sg = proposition ,
4259     Name-pl = Propositions ,
4260     name-pl = propositions ,
4261
4262 type = definition ,
4263     gender = f ,
4264     Name-sg = Définition ,
4265     name-sg = définition ,
4266     Name-pl = Définitions ,
4267     name-pl = définitions ,
4268
4269 type = proof ,
4270     gender = f ,
4271     Name-sg = Démonstration ,
4272     name-sg = démonstration ,
4273     Name-pl = Démonstrations ,
4274     name-pl = démonstrations ,
4275
4276 type = result ,
4277     gender = m ,
4278     Name-sg = Résultat ,
4279     name-sg = résultat ,
4280     Name-pl = Résultats ,
4281     name-pl = résultats ,
4282
4283 type = remark ,
4284     gender = f ,
4285     Name-sg = Remarque ,
4286     name-sg = remarque ,
4287     Name-pl = Remarques ,
4288     name-pl = remarques ,
4289
4290 type = example ,
4291     gender = m ,
4292     Name-sg = Exemple ,

```

```

4293   name-sg = exemple ,
4294   Name-pl = Exemples ,
4295   name-pl = exemples ,
4296
4297   type = algorithm ,
4298   gender = m ,
4299   Name-sg = Algorithme ,
4300   name-sg = algorithme ,
4301   Name-pl = Algorithmes ,
4302   name-pl = algorithmes ,
4303
4304   type = listing ,
4305   gender = m ,
4306   Name-sg = Listing ,
4307   name-sg = listing ,
4308   Name-pl = Listings ,
4309   name-pl = listings ,
4310
4311   type = exercise ,
4312   gender = m ,
4313   Name-sg = Exercice ,
4314   name-sg = exercice ,
4315   Name-pl = Exercices ,
4316   name-pl = exercices ,
4317
4318   type = solution ,
4319   gender = f ,
4320   Name-sg = Solution ,
4321   name-sg = solution ,
4322   Name-pl = Solutions ,
4323   name-pl = solutions ,
4324 </dict-french>

```

10.4 Portuguese

Portuguese dictionary 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.

```

4325 <*package>
4326 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
4327 \zcDeclareLanguageAlias { brazilian } { portuguese }
4328 \zcDeclareLanguageAlias { brazil } { portuguese }
4329 \zcDeclareLanguageAlias { portuges } { portuguese }
4330 </package>
4331 <*dict-portuguese>
4332 namesep = {\nobreakspace} ,
4333 pairsep = {\sim\nobreakspace} ,
4334 listsep = {,~} ,
4335 lastsep = {\sim\nobreakspace} ,
4336 tpairsep = {\sim\nobreakspace} ,
4337 tlistsep = {,~} ,
4338 tlastsep = {\sim\nobreakspace} ,

```

```

4339 notesep = {~} ,
4340 rangeseq = {~a\nobreakspace} ,
4341
4342 type = book ,
4343     gender = m ,
4344     Name-sg = Livro ,
4345     name-sg = livro ,
4346     Name-pl = Livros ,
4347     name-pl = livros ,
4348
4349 type = part ,
4350     gender = f ,
4351     Name-sg = Parte ,
4352     name-sg = parte ,
4353     Name-pl = Partes ,
4354     name-pl = partes ,
4355
4356 type = chapter ,
4357     gender = m ,
4358     Name-sg = Capítulo ,
4359     name-sg = capítulo ,
4360     Name-pl = Capítulos ,
4361     name-pl = capítulos ,
4362
4363 type = section ,
4364     gender = f ,
4365     Name-sg = Seção ,
4366     name-sg = seção ,
4367     Name-pl = Seções ,
4368     name-pl = seções ,
4369
4370 type = paragraph ,
4371     gender = m ,
4372     Name-sg = Parágrafo ,
4373     name-sg = parágrafo ,
4374     Name-pl = Parágrafos ,
4375     name-pl = parágrafos ,
4376     Name-sg-ab = Par. ,
4377     name-sg-ab = par. ,
4378     Name-pl-ab = Par. ,
4379     name-pl-ab = par. ,
4380
4381 type = appendix ,
4382     gender = m ,
4383     Name-sg = Apêndice ,
4384     name-sg = apêndice ,
4385     Name-pl = Apêndices ,
4386     name-pl = apêndices ,
4387
4388 type = page ,
4389     gender = f ,
4390     Name-sg = Página ,
4391     name-sg = página ,
4392     Name-pl = Páginas ,

```

```

4393 name-pl = páginas ,
4394 name-sg-ab = p. ,
4395 name-pl-ab = pp. ,
4396 rangesep = {\textendash} ,
4397
4398 type = line ,
4399 gender = f ,
4400 Name-sg = Linha ,
4401 name-sg = linha ,
4402 Name-pl = Linhas ,
4403 name-pl = linhas ,
4404
4405 type = figure ,
4406 gender = f ,
4407 Name-sg = Figura ,
4408 name-sg = figura ,
4409 Name-pl = Figuras ,
4410 name-pl = figuras ,
4411 Name-sg-ab = Fig. ,
4412 name-sg-ab = fig. ,
4413 Name-pl-ab = Figs. ,
4414 name-pl-ab = figs. ,
4415
4416 type = table ,
4417 gender = f ,
4418 Name-sg = Tabela ,
4419 name-sg = tabela ,
4420 Name-pl = Tabelas ,
4421 name-pl = tabelas ,
4422
4423 type = item ,
4424 gender = m ,
4425 Name-sg = Item ,
4426 name-sg = item ,
4427 Name-pl = Itens ,
4428 name-pl = itens ,
4429
4430 type = footnote ,
4431 gender = f ,
4432 Name-sg = Nota ,
4433 name-sg = nota ,
4434 Name-pl = Notas ,
4435 name-pl = notas ,
4436
4437 type = endnote ,
4438 gender = f ,
4439 Name-sg = Nota ,
4440 name-sg = nota ,
4441 Name-pl = Notas ,
4442 name-pl = notas ,
4443
4444 type = note ,
4445 gender = f ,
4446 Name-sg = Nota ,

```

```

4447     name-sg = nota ,
4448     Name-pl = Notas ,
4449     name-pl = notas ,
4450
4451 type = equation ,
4452     gender = f ,
4453     Name-sg = Equação ,
4454     name-sg = equação ,
4455     Name-pl = Equações ,
4456     name-pl = equações ,
4457     Name-sg-ab = Eq. ,
4458     name-sg-ab = eq. ,
4459     Name-pl-ab = Eqs. ,
4460     name-pl-ab = eqs. ,
4461     refpre = {(} ,
4462     refpos = {)} ,
4463
4464 type = theorem ,
4465     gender = m ,
4466     Name-sg = Teorema ,
4467     name-sg = teorema ,
4468     Name-pl = Teoremas ,
4469     name-pl = teoremas ,
4470
4471 type = lemma ,
4472     gender = m ,
4473     Name-sg = Lema ,
4474     name-sg = lema ,
4475     Name-pl = Lemas ,
4476     name-pl = lemas ,
4477
4478 type = corollary ,
4479     gender = m ,
4480     Name-sg = Corolário ,
4481     name-sg = corolário ,
4482     Name-pl = Corolários ,
4483     name-pl = corolários ,
4484
4485 type = proposition ,
4486     gender = f ,
4487     Name-sg = Proposição ,
4488     name-sg = proposição ,
4489     Name-pl = Proposições ,
4490     name-pl = proposições ,
4491
4492 type = definition ,
4493     gender = f ,
4494     Name-sg = Definição ,
4495     name-sg = definição ,
4496     Name-pl = Definições ,
4497     name-pl = definições ,
4498
4499 type = proof ,
4500     gender = f ,

```



```

4501 Name-sg = Demonstração ,
4502 name-sg = demonstração ,
4503 Name-pl = Demonstrações ,
4504 name-pl = demonstrações ,
4505
4506 type = result ,
4507 gender = m ,
4508 Name-sg = Resultado ,
4509 name-sg = resultado ,
4510 Name-pl = Resultados ,
4511 name-pl = resultados ,
4512
4513 type = remark ,
4514 gender = f ,
4515 Name-sg = Observação ,
4516 name-sg = observação ,
4517 Name-pl = Observações ,
4518 name-pl = observações ,
4519
4520 type = example ,
4521 gender = m ,
4522 Name-sg = Exemplo ,
4523 name-sg = exemplo ,
4524 Name-pl = Exemplos ,
4525 name-pl = exemplos ,
4526
4527 type = algorithm ,
4528 gender = m ,
4529 Name-sg = Algoritmo ,
4530 name-sg = algoritmo ,
4531 Name-pl = Algoritmos ,
4532 name-pl = algoritmos ,
4533
4534 type = listing ,
4535 gender = f ,
4536 Name-sg = Listagem ,
4537 name-sg = listagem ,
4538 Name-pl = Listagens ,
4539 name-pl = listagens ,
4540
4541 type = exercise ,
4542 gender = m ,
4543 Name-sg = Exercício ,
4544 name-sg = exercício ,
4545 Name-pl = Exercícios ,
4546 name-pl = exercícios ,
4547
4548 type = solution ,
4549 gender = f ,
4550 Name-sg = Solução ,
4551 name-sg = solução ,
4552 Name-pl = Soluções ,
4553 name-pl = soluções ,
4554 </dict-portuguese>

```

10.5 Spanish

Spanish dictionary has been initially provided by the author.

```
4555 <*package>
4556 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
4557 </package>

4558 <*dict-spanish>

4559 namesep = {\nobreakspace} ,
4560 pairsep = {\~y\nobreakspace} ,
4561 listsep = { ,~ } ,
4562 lastsep = {\~y\nobreakspace} ,
4563 tpairsep = {\~y\nobreakspace} ,
4564 tlistsep = { ,~ } ,
4565 tlastsep = {\~y\nobreakspace} ,
4566 notesep = {~} ,
4567 rangesep = {\~a\nobreakspace} ,
4568
4569 type = book ,
4570   gender = m ,
4571   Name-sg = Libro ,
4572   name-sg = libro ,
4573   Name-pl = Libros ,
4574   name-pl = libros ,
4575
4576 type = part ,
4577   gender = f ,
4578   Name-sg = Parte ,
4579   name-sg = parte ,
4580   Name-pl = Partes ,
4581   name-pl = partes ,
4582
4583 type = chapter ,
4584   gender = m ,
4585   Name-sg = Capítulo ,
4586   name-sg = capítulo ,
4587   Name-pl = Capítulos ,
4588   name-pl = capítulos ,
4589
4590 type = section ,
4591   gender = f ,
4592   Name-sg = Sección ,
4593   name-sg = sección ,
4594   Name-pl = Secciones ,
4595   name-pl = secciones ,
4596
4597 type = paragraph ,
4598   gender = m ,
4599   Name-sg = Párrafo ,
4600   name-sg = párrafo ,
4601   Name-pl = Párrafos ,
4602   name-pl = párrafos ,
4603
4604 type = appendix ,
```

```

4605     gender = m ,
4606     Name-sg = Apéndice ,
4607     name-sg = apéndice ,
4608     Name-pl = Apéndices ,
4609     name-pl = apéndices ,
4610
4611     type = page ,
4612     gender = f ,
4613     Name-sg = Página ,
4614     name-sg = página ,
4615     Name-pl = Páginas ,
4616     name-pl = páginas ,
4617     rangesep = {\textendash} ,
4618
4619     type = line ,
4620     gender = f ,
4621     Name-sg = Línea ,
4622     name-sg = línea ,
4623     Name-pl = Líneas ,
4624     name-pl = líneas ,
4625
4626     type = figure ,
4627     gender = f ,
4628     Name-sg = Figura ,
4629     name-sg = figura ,
4630     Name-pl = Figuras ,
4631     name-pl = figuras ,
4632
4633     type = table ,
4634     gender = m ,
4635     Name-sg = Cuadro ,
4636     name-sg = cuadro ,
4637     Name-pl = Cuadros ,
4638     name-pl = cuadros ,
4639
4640     type = item ,
4641     gender = m ,
4642     Name-sg = Punto ,
4643     name-sg = punto ,
4644     Name-pl = Puntos ,
4645     name-pl = puntos ,
4646
4647     type = footnote ,
4648     gender = f ,
4649     Name-sg = Nota ,
4650     name-sg = nota ,
4651     Name-pl = Notas ,
4652     name-pl = notas ,
4653
4654     type = endnote ,
4655     gender = f ,
4656     Name-sg = Nota ,
4657     name-sg = nota ,
4658     Name-pl = Notas ,

```

```

4659     name-pl = notas ,
4660
4661 type = note ,
4662     gender = f ,
4663     Name-sg = Nota ,
4664     name-sg = nota ,
4665     Name-pl = Notas ,
4666     name-pl = notas ,
4667
4668 type = equation ,
4669     gender = f ,
4670     Name-sg = Ecuación ,
4671     name-sg = ecuación ,
4672     Name-pl = Ecuaciones ,
4673     name-pl = ecuaciones ,
4674     refpre = {()} ,
4675     refpos = {} ,
4676
4677 type = theorem ,
4678     gender = m ,
4679     Name-sg = Teorema ,
4680     name-sg = teorema ,
4681     Name-pl = Teoremas ,
4682     name-pl = teoremas ,
4683
4684 type = lemma ,
4685     gender = m ,
4686     Name-sg = Lema ,
4687     name-sg = lema ,
4688     Name-pl = Lemas ,
4689     name-pl = lemas ,
4690
4691 type = corollary ,
4692     gender = m ,
4693     Name-sg = Corolario ,
4694     name-sg = corolario ,
4695     Name-pl = Corolarios ,
4696     name-pl = corolarios ,
4697
4698 type = proposition ,
4699     gender = f ,
4700     Name-sg = Proposición ,
4701     name-sg = proposición ,
4702     Name-pl = Proposiciones ,
4703     name-pl = proposiciones ,
4704
4705 type = definition ,
4706     gender = f ,
4707     Name-sg = Definición ,
4708     name-sg = definición ,
4709     Name-pl = Definiciones ,
4710     name-pl = definiciones ,
4711
4712 type = proof ,

```

```

4713     gender = f ,
4714     Name-sg = Demostración ,
4715     name-sg = demostración ,
4716     Name-pl = Demostraciones ,
4717     name-pl = demostraciones ,
4718
4719     type = result ,
4720     gender = m ,
4721     Name-sg = Resultado ,
4722     name-sg = resultado ,
4723     Name-pl = Resultados ,
4724     name-pl = resultados ,
4725
4726     type = remark ,
4727     gender = f ,
4728     Name-sg = Observación ,
4729     name-sg = observación ,
4730     Name-pl = Observaciones ,
4731     name-pl = observaciones ,
4732
4733     type = example ,
4734     gender = m ,
4735     Name-sg = Ejemplo ,
4736     name-sg = ejemplo ,
4737     Name-pl = Ejemplos ,
4738     name-pl = ejemplos ,
4739
4740     type = algorithm ,
4741     gender = m ,
4742     Name-sg = Algoritmo ,
4743     name-sg = algoritmo ,
4744     Name-pl = Algoritmos ,
4745     name-pl = algoritmos ,
4746
4747     type = listing ,
4748     gender = m ,
4749     Name-sg = Listado ,
4750     name-sg = listado ,
4751     Name-pl = Listados ,
4752     name-pl = listados ,
4753
4754     type = exercise ,
4755     gender = m ,
4756     Name-sg = Ejercicio ,
4757     name-sg = ejercicio ,
4758     Name-pl = Ejercicios ,
4759     name-pl = ejercicios ,
4760
4761     type = solution ,
4762     gender = f ,
4763     Name-sg = Solución ,
4764     name-sg = solución ,
4765     Name-pl = Soluciones ,
4766     name-pl = soluciones ,

```

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>\AddToHook</code>	100, 900, 943, 968, 1004, 1006, 1054, 1127, 1143, 1164, 1295, 1308, 1316, 3051, 3087, 3104, 3106, 3111, 3156, 3158, 3160, 3162, 3164, 3166, 3168, 3170, 3174, 3178, 3180, 3185, 3195, 3204, 3210, 3212, 3238, 3248, 3280, 3332, 3353
<code>\appendix</code>	2, 80, 81, 83
<code>\appendixname</code>	80
<code>\AtEndOfPackage</code>	1306
B	
<code>\babelname</code>	953
<code>\babelprovide</code>	17, 29
<code>\bicaption</code>	83
<code>\bionenumcaption</code>	83
<code>\bitwonumcaption</code>	83
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_gset_true:N</code>	1282
<code>\bool_if:NTF</code>	551, 562, 904, 908, 1318, 1618, 1988, 2059, 2196, 2220, 2239, 2246, 2251, 2298, 2302, 2354, 2378, 2382, 2388, 2398, 2404, 2525, 2532, 2539, 2542, 2563, 2592, 2595, 2630, 2643, 2651, 2654, 2710, 2712
<code>\bool_if:nTF</code>	68, 1730, 1740, 1764, 1781, 1796, 1861, 1869, 2232, 2519, 2624, 3029
<code>\bool_lazy_all:nTF</code>	2326, 2722
<code>\bool_lazy_and:nnTF</code>	1592, 1610, 2695, 2737, 2975
<code>\bool_lazy_any:nTF</code>	2855, 2864
<code>\bool_lazy_or:nnTF</code>	1596, 2683
<code>\bool_new:N</code>	500, 775, 776, 803, 827, 836, 843, 844, 857, 858, 877, 878, 933, 1062, 1063, 1064, 1065, 1066, 1157, 1158, 1275, 1626, 1639, 1901, 1902, 1912, 1919, 1920, 1933, 3292
<code>\bool_set:Nn</code>	1589
<code>\bool_set_false:N</code>	788, 792, 885, 894, 895, 910, 1074, 1078, 1085, 1093, 1094, 1095, 1180, 1722, 1957, 1994, 2005, 2208, 2352, 2353, 2862, 2879
<code>\bool_set_true:N</code>	478, 572, 782, 783, 787, 793, 884, 889, 890, 1072, 1079, 1084, 1101, 1103, 1105, 1108, 1109, 1110, 1168, 1173, 1736, 1746, 1750, 1772, 1787, 1802, 1826, 1965, 1989, 1995, 1999, 2006, 2878, 2914, 2921, 2922, 2940, 2947, 2959, 3299
<code>\bool_until_do:Nn</code>	1762, 1958
<code>\bool_while_do:nn</code>	3384
C	
<code>\caption</code>	85
clist commands:	
<code>\clist_map_inline:nn</code>	1096, 1223, 1284, 3262, 3346
<code>\contsubbottom</code>	83
<code>\contsubcaption</code>	83
<code>\counterwithin</code>	5
cs commands:	
<code>\cs_generate_variant:Nn</code>	65, 265, 271, 568, 576, 1351, 1443, 1449, 2557, 2892
<code>\cs_if_exist:NTF</code>	25, 28, 46, 49, 58, 78, 3075, 3081, 3202
<code>\cs_if_exist_p:N</code>	3386
<code>\cs_if_exist_use:N</code>	3173, 3424
<code>\cs_new:Npn</code>	56, 66, 76, 87, 266, 272, 2515, 2558, 2882
<code>\cs_new_eq:NN</code>	3240, 3245
<code>\cs_new_protected:Npn</code>	260, 386, 501, 569, 577, 583, 752, 1314, 1349, 1438, 1444, 1584, 1643, 1685, 1696, 1709, 1839, 1891, 1934, 2066, 2348, 2511, 2513, 2672, 2893, 2969, 3023, 3146, 3300
<code>\cs_new_protected:Npx</code>	99
<code>\cs_set:Npn</code>	3149
<code>\cs_set_eq:NN</code>	103, 3148, 3154, 3241, 3246
<code>\cs_set_nopar:Npn</code>	3231
E	
<code>\endinput</code>	12
<code>\endsidecaption</code>	83
exp commands:	
<code>\exp_args:NNe</code>	35, 38


```

\msg_info:nnn ..... 626, 652, 682, 3125, 3197,
3224, 3289, 3324, 3356, 3376, 3403
\msg_info:nnnn ..... 601, 608, 633
\msg_info:nnnnn ..... 620
\msg_line_context: ..... 115, 121,
125, 127, 130, 136, 142, 148, 154,
159, 164, 169, 174, 180, 185, 188,
191, 196, 200, 206, 209, 215, 220,
228, 232, 240, 244, 246, 248, 251, 255
\msg_new:nnn ..... 113,
119, 124, 126, 128, 134, 140, 146,
152, 157, 162, 167, 172, 177, 182,
187, 189, 194, 199, 201, 203, 205,
207, 213, 218, 224, 226, 231, 233,
238, 243, 245, 247, 249, 254, 256, 258
\msg_note:nnn ..... 547
\msg_warning:nn .....
..... 909, 915, 1148, 1185, 2332
\msg_warning:nnn .....
..... 337, 358, 553, 563, 765,
992, 1035, 1052, 1114, 1125, 1251,
1301, 1311, 1366, 1434, 1487, 1522,
1561, 2020, 2203, 2716, 2732, 3015
\msg_warning:nnnn .....
..... 406, 423, 460, 483, 1196,
1462, 1469, 1499, 2028, 2794, 2842
\msg_warning:nnnnn 446, 490, 1481, 2753
\msg_warning:nnnnnn ..... 2760

```

```

\newcounter ..... 5, 3071, 3072
\NewDocumentCommand .....
331, 348, 1347, 1352, 1399, 1582, 1627
\newfloat ..... 83
\newsfloat ..... 83
\nobreakspace ..... 703,
3460, 3461, 3463, 3464, 3466, 3468,
3667, 3668, 3670, 3671, 3673, 3675,
4116, 4117, 4119, 4120, 4122, 4124,
4332, 4333, 4335, 4336, 4338, 4340,
4559, 4560, 4562, 4563, 4565, 4567

```

\PackageError	7
\pagenote	83, 84
\pagenumbering	7
\pageref	47
\paragraph	36
prg commands:	
\prg_generate_conditional_ variant:Nnn	725, 741
\prg new conditional:Nppnn ..	109, 111

```

\prg_new_protected_conditional:Npnm
    ..... 711, 727, 744
\prg_return_false: .....
    .... 110, 112, 721, 723, 737, 739, 750
\prg_return_true: .....
    ..... 110, 112, 720, 736, 749
\ProcessKeysOptions ..... 1346
prop commands:
    \prop_get:NnN ..... 2992
    \prop_get:NnNTF ..... 388,
        505, 714, 717, 730, 733, 747, 1402,
        2773, 2801, 2809, 2972, 3026, 3038
    \prop_gput:Nnn .....
        .. 339, 355, 366, 373, 380, 1440, 1446
    \prop_gput_if_new:Nnn ..... 579, 585
    \prop_gset_from_keyval:Nn ..... 697
    \prop_if_exist:NTF ..... 1354
    \prop_if_exist_p:N ..... 2977, 3031
    \prop_if_in:NnTF 35, 336, 352, 989, 1032
    \prop_if_in_p:Nn ..... 69, 2984
    \prop_item:Nn ..... 38, 70,
        356, 395, 435, 470, 521, 537, 1409, 1425
    \prop_new:N .....
        330, 340, 696, 1189, 1244, 1323, 1355
    \prop_put:Nnn ..... 756, 1334, 1389
    \prop_remove:Nn ..... 755, 1333, 1381
\providecommand ..... 3
\ProvidesExplPackage ..... 14
\ProvidesFile ..... 17

```

```
\refstepcounter .... 3, 83, 85, 87, 89, 90
\renewlist ..... 91
\RequirePackage .....
... 16, 17, 18, 19, 20, 905, 1140, 1161
```

```

\scantokens ..... 81
seq commands:
  \seq_clear:N ..... 824, 1645
  \seq_const_from_clist:Nn .....
    ..... 279, 287, 298, 310
  \seq_gconcat:NNN ... 317, 320, 324, 327
  \seq_get_left:NN .....
    . 416, 427, 531, 610, 1419, 1471, 1968
  \seq_gput_right:Nn ... 545, 556, 1288
  \seq_gremove_all:Nn ..... 1320
  \seq_if_empty:NTF ..... 402, 442,
    528, 599, 618, 1416, 1460, 1479, 1962
  \seq_if_in:NnTF 420, 457, 508, 605,
    630, 1225, 1286, 1319, 1466, 1491, 1689
  \seq_map_break:n ..... 90, 1882, 1885
  \seq_map_function:NN ..... 1648
  \seq_map_indexed_inline:Nn . 26, 1843

```


<code>\seq_map_inline:Nn</code>	5
.. 641, 658, 672, 1310, 1324, 1359,	
1371, 1507, 1528, 1551, 1879, 3303	
<code>\seq_map_tokens:Nn</code>	72
<code>\seq_new:N</code>	277, 278, 316, 323,
499, 812, 1218, 1276, 1625, 1642, 1900	
<code>\seq_pop_left:NN</code>	1960
<code>\seq_put_right:Nn</code>	1227, 1692
<code>\seq_reverse:N</code>	818
<code>\seq_set_eq:NN</code>	1936
<code>\seq_set_from_clist:Nn</code>	392,
432, 518, 534, 817, 1406, 1422, 1588	
<code>\seq_sort:Nn</code>	49, 1651
<code>\setcounter</code> ..	3073, 3074, 3090, 3105, 3109
<code>\sidefootnote</code>	83, 84
sort commands:	
<code>\sort_return_same:</code>	50,
54, 1658, 1663, 1737, 1757, 1778,	
1793, 1807, 1832, 1867, 1882, 1898	
<code>\sort_return_swapped:</code>	
.. 50, 54, 1671, 1747, 1756, 1777,	
1792, 1808, 1831, 1875, 1885, 1897	
<code>\stepcounter</code>	89, 3089, 3108
str commands:	
<code>\str_case:nnTF</code>	970, 1013, 1098
<code>\str_compare:nNnTF</code>	1753
<code>\str_if_eq:nnTF</code>	89, 468
<code>\str_if_eq_p:nn</code>	2860, 2866, 2868, 2872
<code>\str_new:N</code>	918
<code>\str_set:Nn</code>	923, 925, 927, 929
<code>\string</code>	3310, 3318
<code>\subbottom</code>	83
<code>\subcaption</code>	83
<code>\subcaptionref</code>	83
<code>\subref</code>	92
<code>\subsubsection</code>	36
<code>\subsubsection</code>	36
<code>\subtop</code>	83
T	
<code>\tag</code>	87, 89
TeX and L ^A T _E X commands:	
<code>\@@sidecaption</code>	82
<code>\@Alph</code>	80
<code>\@addtoreset</code>	5
<code>\@auxout</code>	3309, 3317
<code>\@bsphack</code>	504, 3302
<code>\@capytype</code>	
.. 85, 3173, 3206, 3207, 3211, 3424	
<code>\@chapapp</code>	80
<code>\@currentcounter</code> ..	2, 3, 5, 37, 83, 85,
87, 90, 28, 29, 49, 50, 1273, 3207, 3219	
<code>\@currentlabel</code>	3, 83, 85, 90
<code>\@currenvir</code>	3214
<code>\@elt</code>	5
<code>\@esphack</code>	565, 3322
<code>\@ifl@t@r</code>	3
<code>\@mem@scap@afterhook</code>	83
<code>\@memsubcaption</code>	83
<code>\@onlypreamble</code>	347, 361, 1437
<code>\@raw@opt@<package>.sty</code>	34
<code>\bbl@loaded</code>	29
<code>\bbl@main@language</code>	29, 948
<code>\c@lstnumber</code>	90
<code>\c@page</code>	7, 103
<code>\caption@subtypehook</code>	3425
<code>\hyper@link</code> ..	68, 2256, 2528, 2570, 2633
<code>\lst@AddToHook</code>	3372, 3374
<code>\lst@Init</code>	90
<code>\lst@label</code>	3373
<code>\lst@MakeCaption</code>	90
<code>\ltx@gobble</code>	86
<code>\ltx@label</code> ..	86, 3240, 3241, 3245, 3246
<code>\m@mscaplabel</code>	82, 83
<code>\MT@newlabel</code>	3310, 3318
<code>\protected@write</code>	3309, 3317
<code>\zref@addprop</code> ..	22, 32, 43, 53, 55, 97, 108
<code>\zref@default</code>	68, 69, 2512, 2514
<code>\zref@extractdefault</code>	
.. 10, 11, 76, 263, 269, 273	
<code>\zref@ifpropundefined</code> ..	25, 763, 2884
<code>\zref@ifrefcontainsprop</code>	
.. 25, 2023, 2517, 2565, 2620, 2887	
<code>\zref@ifrefundefined</code>	
1653, 1655, 1667, 1991, 1993, 1998,	
2015, 2200, 2209, 2356, 2560, 2674	
<code>\zref@label</code>	86, 3234
<code>\zref@localaddprop</code>	3175, 3426
<code>\ZREF@mainlist</code>	22,
32, 43, 53, 55, 97, 108, 3175, 3426	
<code>\zref@newprop</code>	5, 7,
21, 23, 33, 44, 54, 92, 107, 3172, 3423	
<code>\zref@refused</code>	2013
<code>\zref@wrapper@babel</code> ..	46, 86, 1583, 3234
<code>\textendash</code> ...	707, 3517, 3781, 4396, 4617
<code>\textup</code>	88, 3286, 3287
<code>\thechapter</code>	80
<code>\thelstnumber</code>	90
<code>\thepage</code>	7, 104
<code>\thesection</code>	80
tl commands:	
<code>\c_empty_tl</code> ..	1688, 1712, 1715, 1718,
1720, 2885, 2888, 2889, 2896, 2898	
<code>\c_novalue_tl</code>	1329, 1376
<code>\tl_clear:N</code>	
.. 410, 451, 464, 486, 495, 517, 529,	
594, 1405, 1417, 1455, 1938, 1939,	
1940, 1941, 1942, 1964, 2338, 2339,	

2340, 2341, 2380, 2675, 2679, 2707, 2746, 2793, 2841, 3014, 3044, 3046	\l_tmpa_tl 515, 544, 1606, 1607
\tl_gclear:N 3222	U
\tl_gset:Nn 104	\upshape 3285
\tl_gset_eq:NN 3211	use commands:
\tl_head:N	\use:N 26, 29
.. 1791, 1804, 1816, 1818, 1828, 1830	
\tl_if_empty:NTF 80, 404, 414, 444, 455, 481, 488, 624, 649, 666, 680, 686, 1485, 1515, 1536, 1559, 1565, 1604, 2198, 2608, 2692, 2714, 2751, 2771, 2781, 2817, 3216, 3373	V
\tl_if_empty:nTF 334, 350, 593, 754, 1281, 1454, 2145, 2162, 2179, 2432, 2464, 2477, 2492	\value 3090, 3109
\tl_if_empty_p:N	\verbfootnote 83, 84
..... 2725, 2739, 2859, 2869, 2873	
\tl_if_empty_p:n 1732, 1733, 1742, 1743, 1768, 1769, 1784, 1799	Z
\tl_if_eq:NNTF	\zcDeclareLanguage 13, 14, 46, 331, 3450, 3656, 4109, 4326, 4556
..... 1703, 1726, 2002, 2747, 2900	\zcDeclareLanguageAlias
\tl_if_eq:NnTF 14, 348, 3451, 3452, 3453, 3454, 3455, 3456, 3457, 3659, 3660, 3661, 3662, 3663, 3664, 4110, 4111, 4112, 4113, 4327, 4328, 4329
... 1646, 1678, 1853, 1856, 1881, 1884, 1972, 2018, 2677, 2904, 3214	\zcLanguageSetup 11, 17–19, 40, 42, 43, 1399
\tl_if_eq:nnTF 1845, 2906, 2928, 2932, 2949, 3305, 3313	\zcpageref 47, 1627
\tl_if_exist:NTF 3206	\zceref 31, 34, 39, 46–49, 55, 57, 88, 1582, 1630, 1631
\tl_if_novalue:nTF 1332, 1379	\zcRefTypeSetup . . . 11, 40, 41, 1352, 3283
\tl_map_break:n 90	\zcsetup 29, 34, 39, 40, 1347
\tl_map_tokens:Nn 82	\zlabel . . . 82, 83, 86, 87, 89, 90, 3152, 3373
\tl_new:N 98, 274, 275, 276, 758, 940, 941, 942, 1048, 1067, 1135, 1151, 1268, 1633, 1634, 1635, 1636, 1637, 1638, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1913, 1914, 1915, 1918, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 3176, 3209	zrefcheck commands:
\tl_put_left:Nn . 2235, 2242, 2283, 2783, 2784, 2819, 2821, 2823, 2825	\zrefcheck_zceref_beg_label: . . 1595
\tl_put_right:Nn . 2088, 2105, 2115, 2147, 2159, 2176, 2417, 2430, 2462, 2475, 2490, 2693, 2694, 2705, 3425	\zrefcheck_zceref_end_label_- maybe: 1614
\tl_reverse:N 1713, 1716	\zrefcheck_zceref_run_checks_on_- labels:n 1615
\tl_set:Nn 262, 341, 595, 606, 766, 768, 947, 948, 953, 954, 957, 958, 961, 974, 982, 990, 994, 1017, 1025, 1033, 1037, 1356, 1456, 1467, 1820, 1822, 1974, 1975, 2076, 2078, 2218, 2249, 2369, 2371, 2395, 2689, 2690, 2703, 3177, 3179	zrefclever internal commands:
\tl_set_eq:NN 2336, 3207, 3218	\l__zrefclever_abbrev_bool
\tl_show:N 2299 857, 861, 2696
\tl_tail:N 1821, 1823	\l__zrefclever_capitalize_bool . .
 16, 478, 843, 847, 2684
	\l__zrefclever_capitalize_first_- bool 844, 853, 2686
	__zrefclever_compat_module:nn . .
 38, 1314, 3049, 3067, 3128, 3200, 3227, 3293, 3328, 3359, 3379, 3406, 3429
	__zrefclever_counter_reset_by:n . 6, 36, 37, 58, 60, 62, 66, 3255, 3339
	__zrefclever_counter_reset_by_- aux:nn 73, 76
	__zrefclever_counter_reset_by_- auxi:nnn 83, 87
	\l__zrefclever_counter_resetby_- prop 5, 37, 69, 70, 1244, 1256
	\l__zrefclever_counter_resettters_- seq . . 5, 36, 37, 72, 1218, 1225, 1228
	\l__zrefclever_counter_type_prop 4, 35, 35, 38, 1189, 1201

<code>\l_zrefclever_current_counter_-</code>	<code>_zrefclever_get_fallback_-</code>
<code>tl</code> 3 , 5 , 37 , 21 , 25 ,	<code>transl:nN</code> 745
26 , 36 , 39 , 41 , 46 , 47 , 95 , 1268 , 1271	<code>_zrefclever_get_fallback_-</code>
<code>\l_zrefclever_current_language_-</code>	<code>transl:nNTF</code> 24 , 743 , 3012
<code>tl</code> 29 , 941 ,	<code>_zrefclever_get_ref:n</code>
947 , 953 , 957 , 962 , 975 , 995 , 1018 , 1038 68 , 69 , 2091 , 2108 ,
<code>_zrefclever_declare_default_-</code>	2121 , 2126 , 2150 , 2165 , 2169 , 2182 ,
<code>transl:nnn</code> ... 43 , 1438 , 1517 , 1538	2186 , 2223 , 2243 , 2420 , 2435 , 2442 ,
<code>_zrefclever_declare_type_-</code>	2467 , 2480 , 2484 , 2495 , 2499 , 2515
<code>transl:nnnn</code>	<code>_zrefclever_get_ref_first:</code> ...
..... 43 , 1438 , 1493 , 1543 , 1567 , 1573 68 , 69 , 72 , 2236 , 2284 , 2558
<code>_zrefclever_def_extract:Nnnn</code> ..	<code>_zrefclever_get_ref_font:nN</code> ...
..... 10 ,	.. 11 , 23 , 39 , 78 , 79 , 2052 , 2054 , 3023
260 , 1687 , 1698 , 1700 , 1711 , 1714 ,	<code>_zrefclever_get_ref_string:nN</code> .
1717 , 1719 , 1978 , 1980 , 2895 , 2897 11 , 12 , 23 , 39 , 78 ,
<code>\g_zrefclever_dict_(language)_prop</code>	1606 , 1949 , 1951 , 1953 , 2038 , 2040 ,
..... 18	2042 , 2044 , 2046 , 2048 , 2050 , 2969
<code>\l_zrefclever_dict_decl_case_tl</code>	<code>_zrefclever_get_type_transl:nnnN</code>
..... 274 , 529 , 532 , 606 , 611 , 686 , 11 , 712 , 726
690 , 1417 , 1420 , 1467 , 1472 , 1565 , 1576	<code>_zrefclever_get_type_transl:nnnNTF</code>
<code>\l_zrefclever_dict_declension_-</code>	23 , 711 , 2741 , 2787 , 2829 , 2835 , 3001
<code>seq</code> 274 , 393 , 402 , 416 ,	<code>_zrefclever_if_class_loaded:n</code> 109
420 , 427 , 519 , 528 , 531 , 599 , 605 ,	<code>_zrefclever_if_class_loaded:nTF</code>
610 , 1407 , 1416 , 1419 , 1460 , 1466 , 1471 3130
<code>\l_zrefclever_dict_gender_seq</code> ..	<code>_zrefclever_if_package_-</code>
..... 274 , 433 , 442 ,	<code>loaded:n</code> 109
457 , 535 , 618 , 630 , 1423 , 1479 , 1491	<code>_zrefclever_if_package_-</code>
<code>\l_zrefclever_dict_language_tl</code> .	<code>loaded:nTF</code> 902 ,
..... 274 ,	945 , 951 , 1166 , 3069 , 3229 , 3236 ,
341 , 367 , 374 , 381 , 390 , 398 , 438 ,	3295 , 3330 , 3361 , 3381 , 3408 , 3431
473 , 506 , 510 , 513 , 524 , 540 , 546 ,	<code>\g_zrefclever_koma_captionofbeside_-</code>
548 , 554 , 557 , 580 , 586 , 602 , 609 ,	<code>capttype_tl</code> 3209
621 , 634 , 715 , 718 , 731 , 734 , 1403 ,	<code>\g_zrefclever_koma_capttype_tl</code> ..
1412 , 1428 , 1463 , 1470 , 1482 , 1494 , 3211 , 3216 , 3219 , 3222
1500 , 1518 , 1539 , 1544 , 1568 , 1574	<code>\l_zrefclever_label_a_tl</code>
<code>_zrefclever_extract:nnn</code> 54 , 1905 ,
..... 11 , 272 , 1774 , 1776 ,	1961 , 1979 , 1991 , 2013 , 2015 , 2021 ,
1789 , 1806 , 1894 , 1896 , 2911 , 2913 ,	2024 , 2030 , 2077 , 2091 , 2108 , 2126 ,
2917 , 2919 , 2937 , 2939 , 2943 , 2945	2169 , 2186 , 2214 , 2223 , 2356 , 2360 ,
<code>_zrefclever_extract_unexp:nnn</code> .	2370 , 2396 , 2420 , 2443 , 2484 , 2499
11 , 76 , 266 , 2262 , 2530 , 2536 , 2549 ,	<code>\l_zrefclever_label_b_tl</code>
2576 , 2588 , 2639 , 2647 , 2661 , 2885 , 54 , 1905 ,
2888 , 2889 , 2907 , 2908 , 2929 , 2930 ,	1964 , 1969 , 1981 , 1993 , 1998 , 2360
2933 , 2934 , 2951 , 2955 , 3306 , 3314	<code>\l_zrefclever_label_count_int</code> ..
<code>_zrefclever_extract_url_-</code> 55 , 1903 , 1943 ,
<code>unexp:n</code> 2258 , 2529 , 2572 , 2635 , 2882	2036 , 2069 , 2342 , 2365 , 2509 , 2728
<code>\g_zrefclever_fallback_dict_-</code>	<code>\l_zrefclever_label_enclval_a_-</code>
<code>prop</code> 11 , 696 , 697 , 747	<code>tl</code> 1633 , 1711 , 1713 , 1768 ,
<code>_zrefclever_get_default_-</code>	1784 , 1804 , 1816 , 1820 , 1821 , 1828
<code>transl:nnN</code> 11 , 728 , 742	<code>\l_zrefclever_label_enclval_b_-</code>
<code>_zrefclever_get_default_-</code>	<code>tl</code> 1633 , 1714 , 1716 , 1769 ,
<code>transl:nnNTF</code> 24 , 727 , 3007	1791 , 1799 , 1818 , 1822 , 1823 , 1830
<code>_zrefclever_get_enclosing_-</code>	<code>\l_zrefclever_label_extdoc_a_tl</code>
<code>counters_value:n</code> . 5 , 6 , 56 , 61 , 94 1633 , 1717 ,

1727, 1732, 1742, 1755, 2895, 2901
 \l_zrefclever_label_extdoc_b_tl
 1633, 1719,
 1728, 1733, 1743, 1754, 2897, 2902
 \l_zrefclever_label_type_a_tl ..
 78, 1633, 1688,
 1690, 1693, 1699, 1704, 1853, 1881,
 1974, 1978, 2003, 2018, 2079, 2372,
 2980, 2987, 2995, 3003, 3034, 3041
 \l_zrefclever_label_type_b_tl ..
 1633, 1701,
 1705, 1856, 1884, 1975, 1980, 2004
 _zrefclever_label_type_put_-
 new_right:n 48, 50, 1649, 1685
 \l_zrefclever_label_types_seq ..
 49, 1642, 1645, 1689, 1692, 1879
 _zrefclever_labels_in_sequence:nn
 56, 77, 2212, 2359, 2893
 \g_zrefclever_languages_prop ...
 14, 330, 336, 339, 352, 355,
 356, 388, 505, 714, 730, 989, 1032, 1402
 \l_zrefclever_last_of_type_bool
 55, 1900, 1989,
 1994, 1995, 1999, 2005, 2006, 2059
 \l_zrefclever_lastsep_tl . 1921,
 2045, 2107, 2125, 2149, 2168, 2181
 \l_zrefclever_link_star_bool ...
 1589, 1625, 2522, 2627, 2858
 \l_zrefclever_listsep_tl
 ... 1921, 2043, 2120, 2164, 2419,
 2434, 2441, 2466, 2479, 2483, 2494
 \l_zrefclever_load_dict_-
 verbose_bool ... 500, 551, 562, 572
 \g_zrefclever_loaded_dictionaries_-
 seq 499, 509, 545, 556
 _zrefclever_ltxlabel:n
 86, 3231, 3241, 3246
 \l_zrefclever_main_language_tl .
 29, 942, 948, 954, 958, 983, 1026
 _zrefclever_mathtools_showonlyrefs:n
 1620, 3300
 \l_zrefclever_mathtools_-
 showonlyrefs_bool 1618, 3292, 3299
 _zrefclever_memoir_both_-
 labels:
 .. 3146, 3157, 3159, 3161, 3165, 3169
 \l_zrefclever_memoir_footnote_-
 type_tl 3176, 3177, 3179, 3183
 _zrefclever_memoir_label_and_-
 zlabel:n 3149, 3154
 _zrefclever_memoir_orig_-
 label:n 3148, 3151
 _zrefclever_name_default:
 2511, 2610
 \l_zrefclever_name_format_-
 fallback_tl 1911, 2703,
 2707, 2771, 2814, 2824, 2826, 2838
 \l_zrefclever_name_format_tl ...
 ... 1911, 2689, 2690, 2693, 2694,
 2704, 2705, 2778, 2783, 2784, 2790,
 2795, 2806, 2820, 2821, 2832, 2844
 \l_zrefclever_name_in_link_bool
 70,
 72, 1911, 2251, 2563, 2862, 2878, 2879
 \l_zrefclever_namefont_tl 1921,
 2053, 2254, 2271, 2581, 2600, 2615
 \l_zrefclever_nameinlink_str ...
 918, 923,
 925, 927, 929, 2860, 2866, 2868, 2872
 \l_zrefclever_namesep_tl
 .. 1921, 2039, 2584, 2603, 2611, 2618
 \l_zrefclever_next_is_same_bool
 55, 77, 1916,
 2353, 2382, 2398, 2404, 2922, 2960
 \l_zrefclever_next_maybe_range_-
 bool 55,
 76, 77, 1916, 2208, 2220, 2352,
 2378, 2388, 2914, 2921, 2940, 2948
 \l_zrefclever_noabbrev_first_-
 bool 858, 867, 2700
 \g_zrefclever_nocompat_bool ...
 1275, 1282, 1318
 \l_zrefclever_nocompat_bool ... 38
 \g_zrefclever_nocompat_modules_-
 seq 1276, 1286, 1289, 1310, 1319, 1320
 \l_zrefclever_nocompat_modules_-
 seq 38
 \l_zrefclever_nudge_comptosing_-
 bool ... 1064, 1094, 1103, 1109, 2724
 \l_zrefclever_nudge_enabled_-
 bool 1062, 1072, 1074,
 1078, 1079, 1084, 1085, 2328, 2710
 \l_zrefclever_nudge_gender_bool
 1066, 1095, 1105, 1110, 2738
 \l_zrefclever_nudge_multitype_-
 bool ... 1063, 1093, 1101, 1108, 2329
 \l_zrefclever_nudge_singular_-
 bool 1065, 1121, 2712
 _zrefclever_orig_ltxlabel:n ...
 3233, 3240, 3245
 _zrefclever_page_format_aux: ..
 99, 103
 \g_zrefclever_page_format_tl ...
 7, 98, 104, 107
 \l_zrefclever_pairsep_tl
 1921, 2041, 2090, 2221
 \l_zrefclever_preposinlink_bool
 . 933, 936, 2525, 2532, 2539, 2542,

2592, 2595, 2630, 2643, 2651, 2654
 _zrefclever_process_language_-
 options: 32, 33, 386, 1591
 _zrefclever_prop_put_non_-
 empty:Nnn 24, 752, 1200, 1255
 _zrefclever_provide_dict_-
 default_transl:nn 20, 577, 650, 667
 _zrefclever_provide_dict_type_-
 transl:nn 20, 577, 631, 668, 687, 689
 _zrefclever_provide_dictionary:n
 11, 18, 20, 46,
 501, 573, 1008, 1019, 1027, 1040, 1590
 _zrefclever_provide_dictionary_-
 verbose:n ... 20, 569, 976, 984, 997
 \l_zrefclever_range_beg_label_-
 tl 55, 1916, 1942,
 2121, 2145, 2151, 2162, 2166, 2179,
 2183, 2341, 2380, 2395, 2432, 2436,
 2464, 2468, 2477, 2481, 2492, 2496
 \l_zrefclever_range_count_int ..
 55,
 1916, 1945, 2099, 2135, 2344, 2381,
 2392, 2397, 2403, 2411, 2455, 2504
 \l_zrefclever_range_same_count_-
 int 55,
 1916, 1946, 2086, 2123, 2136, 2345,
 2383, 2399, 2405, 2439, 2456, 2505
 \l_zrefclever_rangesep_tl
 1921, 2047, 2185, 2222, 2498
 \l_zrefclever_ref_decl_case_tl ..
 15, 404, 409, 410, 414, 417,
 421, 425, 428, 481, 484, 486, 1048,
 1058, 2781, 2785, 2817, 2822, 2827
 _zrefclever_ref_default:
 2511, 2555, 2561, 2604, 2668
 \l_zrefclever_ref_gender_tl ...
 16, 444, 450,
 451, 455, 458, 463, 464, 488, 494,
 495, 1067, 1131, 2739, 2748, 2755, 2763
 \l_zrefclever_ref_language_tl ..
 15, 29, 30, 389,
 408, 426, 448, 462, 485, 492, 940,
 961, 974, 977, 982, 985, 990, 994,
 998, 1008, 1017, 1020, 1025, 1028,
 1033, 1037, 1041, 1590, 2742, 2757,
 2765, 2788, 2830, 2836, 3002, 3008
 \c_zrefclever_ref_options_font_-
 seq 12, 23, 279
 \c_zrefclever_ref_options_-
 necessarily_not_type_specific_-
 seq 23, 279, 642, 1360, 1508
 \c_zrefclever_ref_options_-
 possibly_type_specific_seq ..
 23, 279, 659, 1529
 \l_zrefclever_ref_options_prop .
 . 39, 41, 1323, 1333, 1334, 2972, 3026
 \c_zrefclever_ref_options_-
 reference_seq 279, 1325
 \c_zrefclever_ref_options_type_-
 names_seq 279, 673, 1552
 \c_zrefclever_ref_options_-
 typesetup_seq 279, 1372
 \l_zrefclever_ref_property_tl ..
 25, 758, 766, 768, 1646, 1678, 1972,
 2025, 2029, 2517, 2567, 2622, 2904
 \l_zrefclever_ref_typeset_font_-
 tl 1135, 1137, 1601
 \l_zrefclever_reffont_tl . 1921,
 2055, 2535, 2548, 2587, 2646, 2660
 \l_zrefclever_refpos_tl
 1921, 2051, 2540, 2543,
 2552, 2593, 2596, 2652, 2655, 2665
 \l_zrefclever_refpre_tl
 1921, 2049, 2526,
 2533, 2546, 2585, 2631, 2644, 2658
 \l_zrefclever_setup_type_tl ...
 20, 274, 517, 581, 594,
 595, 624, 649, 666, 680, 1356, 1384,
 1392, 1405, 1455, 1456, 1485, 1495,
 1515, 1536, 1545, 1559, 1569, 1575
 \l_zrefclever_sort_decided_bool
 1639, 1722, 1736, 1746,
 1750, 1762, 1772, 1787, 1802, 1826
 _zrefclever_sort_default:nn ...
 50, 1680, 1696
 _zrefclever_sort_default_-
 different_types:nn
 26, 48, 53, 1707, 1839
 _zrefclever_sort_default_same_-
 type:nn 48, 50, 1706, 1709
 _zrefclever_sort_labels:
 48-50, 54, 1599, 1643
 _zrefclever_sort_page:nn
 54, 1679, 1891
 \l_zrefclever_sort_prior_a_int .
 1640,
 1841, 1847, 1848, 1854, 1864, 1872
 \l_zrefclever_sort_prior_b_int .
 1640,
 1842, 1849, 1850, 1857, 1865, 1873
 \l_zrefclever_tlastsep_tl
 1921, 1954, 2322
 \l_zrefclever_tlistsep_tl
 1921, 1952, 2293
 \l_zrefclever_tpairsep_tl
 1921, 1950, 2315
 \l_zrefclever_type_<type>-
 options_prop 41

\l_zrefclever_type_count_int ...	\l_zrefclever_typeset_range_-
.. 55, 72, 1903, 1944, 2290, 2292,	bool 836, 839, 1598, 2196
2305, 2330, 2343, 2687, 2699, 2875	\l_zrefclever_typeset_ref_bool .
\l_zrefclever_type_first_label_- 775, 782, 787, 792, 2233, 2239
tl 55, 70, 1905, 1940, 2076, 2200,	_zrefclever_typeset_refs:
2209, 2213, 2243, 2259, 2263, 2339, 54, 56, 57, 1602, 1934
2369, 2560, 2566, 2573, 2577, 2589,	_zrefclever_typeset_refs_last_-
2621, 2636, 2640, 2648, 2662, 2674	of_type: . 60, 68, 70, 72, 2061, 2066
\l_zrefclever_type_first_label_-	_zrefclever_typeset_refs_not_-
type_tl 55, 72, 1905, 1941,	last_of_type:
2078, 2204, 2340, 2371, 2678, 2718, 56, 60, 68, 76, 77, 2063, 2348
2734, 2743, 2756, 2762, 2776, 2789,	\l_zrefclever_typeset_sort_bool
2796, 2804, 2812, 2831, 2837, 2845 803, 806, 1597
\l_zrefclever_type_name_gender_-	\l_zrefclever_typesort_seq
tl 1911, 2745, 2746, 2749, 2751, 2764 26, 53, 812, 817, 818, 824, 1843
_zrefclever_type_name_setup: ..	\l_zrefclever_use_hyperref_bool
..... 11, 12, 70, 2231, 2672 877, 884,
\l_zrefclever_type_name_tl	889, 894, 904, 910, 2521, 2626, 2857
..... 70, 72,	\l_zrefclever_verbose_testing_-
1911, 2266, 2272, 2582, 2601, 2608,	bool 1933, 2298
2616, 2675, 2679, 2779, 2791, 2793,	\l_zrefclever_warn_hyperref_-
2807, 2815, 2833, 2839, 2841, 2859	bool 878, 885, 890, 895, 908
\l_zrefclever_typeset_compress_-	_zrefclever_zcref:nnn
bool 827, 830, 2354 14, 15, 1583, 1584
\l_zrefclever_typeset_labels_-	_zrefclever_zcref:nnnn 46, 48, 1584
seq 54, 1900, 1936, 1960, 1962, 1968	\l_zrefclever_zcref_labels_seq .
\l_zrefclever_typeset_last_bool 48, 49, 1588,
..... 55, 1900,	1616, 1621, 1625, 1648, 1651, 1937
1957, 1958, 1965, 1988, 2302, 2874	\l_zrefclever_zcref_note_tl ...
\l_zrefclever_typeset_name_bool 1151, 1154, 1604, 1608
..... 776, 783, 788, 793, 2233, 2246	\l_zrefclever_zcref_with_check_-
\l_zrefclever_typeset_queue_-	bool 1158, 1173, 1594, 1612
curr_tl 55, 57, 68, 72,	_zrefclever_zcsetup:n
1905, 1939, 2088, 2105, 2115, 2147, 40, 1348, 1349, 3053,
2159, 2176, 2198, 2218, 2235, 2242,	3077, 3083, 3091, 3113, 3132, 3182,
2249, 2283, 2299, 2310, 2316, 2323,	3186, 3187, 3196, 3250, 3281, 3334,
2337, 2338, 2417, 2430, 2462, 2475,	3354, 3363, 3375, 3390, 3410, 3433
2490, 2692, 2714, 2725, 2869, 2873	\l_zrefclever_zrefcheck_-
\l_zrefclever_typeset_queue_-	available_bool
prev_tl . 55, 1905, 1938, 2294, 2336 1157, 1168, 1180, 1593, 1611