

The `zref-clever` package^{*}

Code documentation

Gustavo Barros[†]

2022-01-10

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	11
4.4	Reference format	11
4.5	Languages	13
4.6	Language files	17
4.7	Options	26
5	Configuration	42
5.1	<code>\zcsetup</code>	42
5.2	<code>\zcRefTypeSetup</code>	42
5.3	<code>\zcLanguageSetup</code>	45
6	User interface	50
6.1	<code>\zceref</code>	50
6.2	<code>\zcpageref</code>	52
7	Sorting	52
8	Typesetting	59

^{*}This file describes v0.1.2-alpha, released 2022-01-10.

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

9	Compatibility	86
9.1	appendix	86
9.2	appendices	87
9.3	memoir	88
9.4	KOMA	91
9.5	amsmath	92
9.6	mathtools	94
9.7	breqn	95
9.8	listings	96
9.9	enumitem	97
9.10	subcaption	97
9.11	subfig	98
10	Language files	98
10.1	English	99
10.2	German	102
10.3	French	111
10.4	Portuguese	115
10.5	Spanish	120
10.6	Dutch	124
	Index	128

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-11-15}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
```

```

9      'zref-clever' requires a LaTeX kernel 2021-11-15 or newer.%
10      \MessageBreak Loading will abort!%
11      }%
12      \endinput
13  }%

Identify the package.

14 \ProvidesExplPackage {zref-clever} {2022-01-10} {0.1.2-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 `\cl@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see `ltcounts.dtx` in `texdoc source2e`). Besides, there may be a chain of resetting counters, which must be taken into account: if `counterC` gets reset by `counterB`, and `counterB` gets reset by `counterA`, stepping the latter affects all three of them.

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

`__zrefclever_get_enclosing_counters_value:n`

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

```

\__zrefclever_get_enclosing_counters_value:n {<counter>}

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 } { key-boolean-or-empty }
127 {
128   The-key~'#1'~only-accepts-the-values~'true',~'false'~
129   or-an-empty-value~\msg_line_context:~
130 }
131 \msg_new:nnn { zref-clever } { language-declared }
132 { Language~'#1'~is-already-declared~\msg_line_context:~Nothing-to-do. }
133 \msg_new:nnn { zref-clever } { unknown-language-alias }
134 {
135   Language~'#1'~is-unknown~\msg_line_context:~Can't-alias-to-it.~
136   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
137   '\iow_char:N\zcDeclareLanguageAlias'.
138 }
139 \msg_new:nnn { zref-clever } { unknown-language-setup }
140 {
141   Language~'#1'~is-unknown~\msg_line_context:~Can't-set-it-up.~
142   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
143   '\iow_char:N\zcDeclareLanguageAlias'.
144 }
145 \msg_new:nnn { zref-clever } { unknown-language-opt }
146 {
147   Language~'#1'~is-unknown~\msg_line_context:~Using-default.~
148   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
149   '\iow_char:N\zcDeclareLanguageAlias'.
150 }
151 \msg_new:nnn { zref-clever } { unknown-language-decl }
152 {
153   Can't-set-declension~'#1'~for-unknown-language~'#2'~\msg_line_context:~
154   See-documentation-for~'\iow_char:N\zcDeclareLanguage'~and~
155   '\iow_char:N\zcDeclareLanguageAlias'.
156 }
157 \msg_new:nnn { zref-clever } { language-no-decl-ref }
158 {
159   Language~'#1'~has-no-declared-declension-cases~\msg_line_context:~
160   Nothing-to-do-with-option~'d=#2'.
161 }
162 \msg_new:nnn { zref-clever } { language-no-gender }

```



```

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

```

```

217     '\iow_char:N\zcDeclareLanguageAlias'.
218   }
219   \msg_new:nnn { zref-clever } { zref-property-undefined }
220   {
221     Option~'ref=#1'~requested~\msg_line_context:..~
222     But~the~property~'#1'~is~not~declared,~falling-back-to~'default'.
223   }
224   \msg_new:nnn { zref-clever } { hyperref-preamble-only }
225   {
226     Option~'hyperref'~only~available~in~the~preamble~\msg_line_context:..~
227     To~inhibit~hyperlinking~locally,~you~can~use~the~starred~version~of~
228     '\iow_char:N\zceref'.
229   }
230   \msg_new:nnn { zref-clever } { missing-hyperref }
231   { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
232   \msg_new:nnn { zref-clever } { titleref-preamble-only }
233   {
234     Option~'titleref'~only~available~in~the~preamble~\msg_line_context:..~
235     Did~you~mean~'ref=title'?.
236   }
237   \msg_new:nnn { zref-clever } { option-preamble-only }
238   { Option~'#1'~only~available~in~the~preamble~\msg_line_context:.. }
239   \msg_new:nnn { zref-clever } { unknown-compat-module }
240   {
241     Unknown~compatibility~module~'#1'~given~to~option~'nocompat'..~
242     Nothing~to~do.
243   }
244   \msg_new:nnn { zref-clever } { missing-zref-check }
245   {
246     Option~'check'~requested~\msg_line_context:..~
247     But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
248   }
249   \msg_new:nnn { zref-clever } { missing-type }
250   { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
251   \msg_new:nnn { zref-clever } { missing-property }
252   { Reference~property~'#1'~undefined~for~label~'#2'~\msg_line_context:.. }
253   \msg_new:nnn { zref-clever } { missing-name }
254   { Reference~format~option~'#1'~undefined~for~type~'#2'~\msg_line_context:.. }
255   \msg_new:nnn { zref-clever } { missing-string }
256   {
257     We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:..~
258     But~we~should~have:~throw~a~rock~at~the~maintainer.
259   }
260   \msg_new:nnn { zref-clever } { single-element-range }
261   { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }
262   \msg_new:nnn { zref-clever } { compat-package }
263   { Loaded~support~for~'#1'~package. }
264   \msg_new:nnn { zref-clever } { compat-class }
265   { Loaded~support~for~'#1'~documentclass. }
266   \msg_new:nnn { zref-clever } { option-deprecated }
267   {
268     Option~'#1'~has~been~deprecated~\msg_line_context:.\iow_newline:
269     Use~'#2'~instead.
270   }

```

4.3 Data extraction

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

```

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

271 \cs_new_protected:Npn \__zrefclever_extract_default:Nnnn #1#2#3#4
272   {
273     \exp_args:NNNo \exp_args:NNo \tl_set:Nn #1
274     { \zref@extractdefault {#2} {#3} {#4} }
275   }
276 \cs_generate_variant:Nn \__zrefclever_extract_default:Nnnn { NVnn }

```

(End definition for `_zrefclever_extract_default:Nnnn`.)

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

```

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

277 \cs_new:Npn \__zrefclever_extract_unexp:nnn #1#2#3
278   {
279     \exp_args:NNo \exp_args:No
280     \exp_not:n { \zref@extractdefault {#1} {#2} {#3} }
281   }
282 \cs_generate_variant:Nn \__zrefclever_extract_unexp:nnn { Vnn , nvn , Vvn }

```

(End definition for `_zrefclever_extract_unexp:nnn`.)

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

```

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

283 \cs_new:Npn \__zrefclever_extract:nnn #1#2#3
284   { \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_opt_typeset:nN`, `_zrefclever_get_ref_opt_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_unknown_lang_prop`.

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

```

285 \tl_new:N \l__zrefclever_setup_type_tl
286 \tl_new:N \l__zrefclever_base_language_tl
287 \tl_new:N \l__zrefclever_lang_decl_case_tl
288 \seq_new:N \l__zrefclever_lang_declension_seq
289 \seq_new:N \l__zrefclever_lang_gender_seq

```

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

`\c__zrefclever_ref_options_type_names_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.

```

290 \seq_const_from_clist:Nn
291 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
292 {
293   tpairsep ,
294   tlistsep ,
295   tlastsep ,
296   notesep ,
297 }
298 \seq_const_from_clist:Nn
299 \c__zrefclever_ref_options_possibly_type_specific_seq
300 {
301   namesep ,
302   pairsep ,
303   listsep ,
304   lastsep ,
305   rangesep ,
306   preref ,
307   postref ,
308 }

```

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

```

309 \seq_const_from_clist:Nn
310 \c__zrefclever_ref_options_type_names_seq
311 {
312   Name-sg ,
313   name-sg ,
314   Name-pl ,
315   name-pl ,
316   Name-sg-ab ,
317   name-sg-ab ,
318   Name-pl-ab ,
319   name-pl-ab ,
320 }

```

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

```

321 \seq_const_from_clist:Nn
322 \c__zrefclever_ref_options_font_seq
323 {
324     namefont ,
325     reffont ,
326 }

```

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

```

327 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
328 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
329 \c__zrefclever_ref_options_possibly_type_specific_seq
330 \c__zrefclever_ref_options_type_names_seq
331 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
332 \c__zrefclever_ref_options_typesetup_seq
333 \c__zrefclever_ref_options_font_seq
334 \seq_new:N \c__zrefclever_ref_options_reference_seq
335 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
336 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
337 \c__zrefclever_ref_options_possibly_type_specific_seq
338 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
339 \c__zrefclever_ref_options_reference_seq
340 \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 “base language 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 language file in that particular language”) is just like any other one, the only difference is that the “language name” happens to be the same as the “base language name”, in other words, it is an “alias to itself”.

```

341 \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 “base language 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 language files would become much too sensitive to this particular user input, and unnecessarily so. `\zcDeclareLanguage` is preamble only.

```

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

```

```

342 \NewDocumentCommand \zcDeclareLanguage { 0 { } m }
343 {
344   \group_begin:
345   \tl_if_empty:nF {#2}
346   {
347     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
348     { \msg_warning:nnn { zref-clever } { language-declared } {#2} }
349     {
350       \prop_gput:Nnn \g__zrefclever_languages_prop {#2} {#2}
351       \prop_new:c { g__zrefclever_lang_ #2 _prop }
352       \tl_set:Nn \l__zrefclever_base_language_tl {#2}
353       \keys_set:nn { zref-clever / declarelang } {#1}
354     }
355   }
356   \group_end:
357 }
358 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

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

```

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

359 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
360 {
361   \tl_if_empty:nF {#1}
362   {
363     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
364     {
365       \exp_args:NNnx
366       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
367       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
368     }
369     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
370   }
371 }
372 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

```

373 \keys_define:nn { zref-clever / declarelang }
374 {
375   declension .code:n =
376   {
377     \prop_gput:cnn
378     { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
379     { declension } {#1}
380   } ,
381   declension .value_required:n = true ,
382   gender .code:n =
383   {
384     \prop_gput:cnn
385     { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }

```

```

386         { gender } {#1}
387     } ,
388     gender .value_required:n = true ,
389     allcaps .code:n =
390     {
391         \prop_gput:cnn
392         { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
393         { allcaps } { true }
394     } ,
395     allcaps .value_forbidden:n = true ,
396 }

```

`_zrefclever_process_language_options:` Auxiliary function for `_zrefclever_zcref: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.

```

397 \cs_new_protected:Npn \_zrefclever_process_language_options:
398 {
399     \exp_args:NNx \prop_get:NnNTF \g__zrefclever_languages_prop
400     { \l__zrefclever_ref_language_tl }
401     \l__zrefclever_base_language_tl
402     {

```

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.

```

403         \exp_args:NNx \seq_set_from_clist:Nn
404         \l__zrefclever_lang_declension_seq
405         {
406             \prop_item:cn
407             {
408                 g__zrefclever_lang_
409                 \l__zrefclever_base_language_tl _prop
410             }
411             { declension }
412         }
413     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
414     {
415         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
416         {
417             \msg_warning:nxxx { zref-clever }
418             { language-no-decl-ref }
419             { \l__zrefclever_ref_language_tl }
420             { \l__zrefclever_ref_decl_case_tl }
421             \tl_clear:N \l__zrefclever_ref_decl_case_tl
422         }
423     }

```

```

424 {
425   \tl_if_empty:NTF \l__zrefclever_ref_decl_case_tl
426   {
427     \seq_get_left:NN \l__zrefclever_lang_declension_seq
428     \l__zrefclever_ref_decl_case_tl
429   }
430   {
431     \seq_if_in:NVF \l__zrefclever_lang_declension_seq
432     \l__zrefclever_ref_decl_case_tl
433     {
434       \msg_warning:nxxx { zref-clever }
435       { unknown-decl-case }
436       { \l__zrefclever_ref_decl_case_tl }
437       { \l__zrefclever_ref_language_tl }
438       \seq_get_left:NN \l__zrefclever_lang_declension_seq
439       \l__zrefclever_ref_decl_case_tl
440     }
441   }
442 }

```

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.

```

443 \exp_args:NNx \seq_set_from_clist:Nn
444   \l__zrefclever_lang_gender_seq
445 {
446   \prop_item:cn
447   {
448     g__zrefclever_lang_
449     \l__zrefclever_base_language_tl _prop
450   }
451   { gender }
452 }
453 \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
454 {
455   \tl_if_empty:NF \l__zrefclever_ref_gender_tl
456   {
457     \msg_warning:nxxxx { zref-clever }
458     { language-no-gender }
459     { \l__zrefclever_ref_language_tl }
460     { g }
461     { \l__zrefclever_ref_gender_tl }
462     \tl_clear:N \l__zrefclever_ref_gender_tl
463   }
464 }
465 {
466   \tl_if_empty:NF \l__zrefclever_ref_gender_tl
467   {
468     \seq_if_in:NVF \l__zrefclever_lang_gender_seq
469     \l__zrefclever_ref_gender_tl
470     {
471       \msg_warning:nxxx { zref-clever }
472       { gender-not-declared }
473       { \l__zrefclever_ref_language_tl }

```



```

474         { \l__zrefclever_ref_gender_tl }
475         \tl_clear:N \l__zrefclever_ref_gender_tl
476     }
477 }
478 }

```

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

```

479     \str_if_eq:eeT
480     {
481         \prop_item:cn
482         {
483             g__zrefclever_lang_
484             \l__zrefclever_base_language_tl _prop
485         }
486         { allcaps }
487     }
488     { true }
489     { \prop_put:Nnn \l__zrefclever_ref_options_prop { cap } { true } }
490 }
491 {

```

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

```

492     \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
493     {
494         \msg_warning:nxxx { zref-clever } { unknown-language-decl }
495         { \l__zrefclever_ref_decl_case_tl }
496         { \l__zrefclever_ref_language_tl }
497         \tl_clear:N \l__zrefclever_ref_decl_case_tl
498     }
499     \tl_if_empty:NF \l__zrefclever_ref_gender_tl
500     {
501         \msg_warning:nxxx { zref-clever }
502         { language-no-gender }
503         { \l__zrefclever_ref_language_tl }
504         { g }
505         { \l__zrefclever_ref_gender_tl }
506         \tl_clear:N \l__zrefclever_ref_gender_tl
507     }
508 }
509 }

```

(End definition for `__zrefclever_process_language_options:.`)

4.6 Language files

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

The built-in language files and their related infrastructure are designed to perform “on the fly” loading of the language files, “lazily” as needed. Much like `babel` does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of

parsimony, of “loading the least possible”. Therefore, we load at `begindocument` one single language (see **lang option**), as specified by the user in the preamble with the `lang` option or, failing any specification, the current language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

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

`__zrefclever_provide_langfile:n` is only meant to load the built-in language files. For languages declared by the user, or for any settings to a known language made with `\zcLanguageSetup`, values are populated directly to a variable `\g__zrefclever_lang_⟨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_langfiles_seq` Used to keep track of whether a language file has already been loaded or not.

```
510 \seq_new:N \g__zrefclever_loaded_langfiles_seq
```

(End definition for `\g__zrefclever_loaded_langfiles_seq`.)

`\l__zrefclever_load_langfile_verbose_bool` Controls whether `__zrefclever_provide_langfile:n` fails silently or verbosely in case of unknown languages or not found language files.

```
511 \bool_new:N \l__zrefclever_load_langfile_verbose_bool
```

(End definition for `\l__zrefclever_load_langfile_verbose_bool`.)

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

```
\__zrefclever_provide_langfile:n {⟨language⟩}
```

```
512 \cs_new_protected:Npn \__zrefclever_provide_langfile:n #1
```

```
513 {
514   \group_begin:
515   \@bsphack
```

```

516 \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
517 \l__zrefclever_base_language_tl
518 {
519   \seq_if_in:NVF
520     \g__zrefclever_loaded_langfiles_seq
521     \l__zrefclever_base_language_tl
522     {
523       \exp_args:Nx \file_get:nnNTF
524         { zref-clever- \l__zrefclever_base_language_tl .lang }
525         { \ExplSyntaxOn }
526         \l_tmpa_tl
527         {
528           \tl_clear:N \l__zrefclever_setup_type_tl
529           \exp_args:NNx \seq_set_from_clist:Nn
530             \l__zrefclever_lang_declension_seq
531             {
532               \prop_item:cn
533                 {
534                   g__zrefclever_lang_
535                   \l__zrefclever_base_language_tl _prop
536                 }
537               { declension }
538             }
539           \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
540             { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
541             {
542               \seq_get_left:NN \l__zrefclever_lang_declension_seq
543               \l__zrefclever_lang_decl_case_tl
544             }
545           \exp_args:NNx \seq_set_from_clist:Nn
546             \l__zrefclever_lang_gender_seq
547             {
548               \prop_item:cn
549               {
550                 g__zrefclever_lang_
551                 \l__zrefclever_base_language_tl _prop
552               }
553               { gender }
554             }
555           \keys_set:nV { zref-clever / langfile } \l_tmpa_tl
556           \seq_gput_right:NV \g__zrefclever_loaded_langfiles_seq
557             \l__zrefclever_base_language_tl
558           \msg_note:nxx { zref-clever } { langfile-loaded }
559             { \l__zrefclever_base_language_tl }
560         }
561       {
562         \bool_if:NT \l__zrefclever_load_langfile_verbose_bool
563         {
564           \msg_warning:nxx { zref-clever } { langfile-not-available }
565             { \l__zrefclever_base_language_tl }
566         }
567       }
568     }
569   }
570 }

```

Even if we don't have the actual language file, we register it as "loaded". At this point, it is a known language, properly declared. There is no point in trying to load it multiple

times, because users cannot really provide the language 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 language-specific options themselves, by means of `\zcLanguageSetup`, everything would be in place, and they could use the `lang` option multiple times, and the `langfile-not-available` warning would never go away.

```

567             \seq_gput_right:NV \g__zrefclever_loaded_langfiles_seq
568             \l__zrefclever_base_language_tl
569         }
570     }
571 }
572 {
573     \bool_if:NT \l__zrefclever_load_langfile_verbose_bool
574     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
575 }
576 \@esphack
577 \group_end:
578 }
579 \cs_generate_variant:Nn \__zrefclever_provide_langfile:n { x }

```

(End definition for `__zrefclever_provide_langfile:n`.)

`__zrefclever_provide_langfile_verbose:n` Does the same as `__zrefclever_provide_langfile:n`, but warns if the loading of the language file has failed.

```

        \__zrefclever_provide_langfile_verbose:n {<language>}

580 \cs_new_protected:Npn \__zrefclever_provide_langfile_verbose:n #1
581 {
582     \group_begin:
583     \bool_set_true:N \l__zrefclever_load_langfile_verbose_bool
584     \__zrefclever_provide_langfile:n {#1}
585     \group_end:
586 }
587 \cs_generate_variant:Nn \__zrefclever_provide_langfile_verbose:n { x }

```

(End definition for `__zrefclever_provide_langfile_verbose:n`.)

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

```

        \__zrefclever_provide_lang_opt_type:nn {<key>} {<value>}
        \__zrefclever_provide_lang_opt_default:nn {<key>} {<value>}

588 \cs_new_protected:Npn \__zrefclever_provide_lang_opt_type:nn #1#2
589 {
590     \exp_args:Nnx \prop_gput_if_new:cnn
591     { \g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
592     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
593 }
594 \cs_generate_variant:Nn \__zrefclever_provide_lang_opt_type:nn { nV }
595 \cs_new_protected:Npn \__zrefclever_provide_lang_opt_default:nn #1#2
596 {

```

```

597 \prop_gput_if_new:cnn
598 { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
599 { default- #1 } {#2}
600 }
601 \cs_generate_variant:Nn \__zrefclever_provide_lang_opt_default:nn { nV }

```

(End definition for __zrefclever_provide_lang_opt_type:nn and __zrefclever_provide_lang_opt_default:nn.)

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

```

602 \keys_define:nn { zref-clever / langfile }
603 {
604   type .code:n =
605   {
606     \tl_if_empty:NTF {#1}
607     { \tl_clear:N \l__zrefclever_setup_type_tl }
608     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
609   } ,
610
611   case .code:n =
612   {
613     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
614     {
615       \msg_info:nxxx { zref-clever } { language-no-decl-setup }
616       { \l__zrefclever_base_language_tl } {#1}
617     }
618     {
619       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
620       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
621       {
622         \msg_info:nxxx { zref-clever } { unknown-decl-case }
623         {#1} { \l__zrefclever_base_language_tl }
624         \seq_get_left:NN \l__zrefclever_lang_declension_seq
625         \l__zrefclever_lang_decl_case_tl
626       }
627     }
628   } ,
629   case .value_required:n = true ,
630
631   gender .code:n =
632   {
633     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
634     {
635       \msg_info:nxxxx { zref-clever } { language-no-gender }
636       { \l__zrefclever_base_language_tl } { gender } {#1}
637     }
638     {
639       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
640       {
641         \msg_info:nnn { zref-clever }
642         { option-only-type-specific } { gender }

```

```

643     }
644     {
645         \clist_clear:N \l_tmpa_clist
646         \clist_map_inline:nn {#1}
647         {
648             \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
649             { \clist_put_right:Nn \l_tmpa_clist {##1} }
650             {
651                 \msg_info:nxxx { zref-clever }
652                 { gender-not-declared }
653                 { \l__zrefclever_base_language_tl } {##1}
654             }
655         }
656         \clist_if_empty:NF \l_tmpa_clist
657         {
658             \exp_args:Nnx \l__zrefclever_provide_lang_opt_type:nn
659             { gender } { \clist_use:Nn \l_tmpa_clist { , } }
660         }
661     }
662 } ,
663 gender .value_required:n = true ,
664
665 cap .choices:nn =
666 { true , false }
667 {
668     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
669     {
670         {
671             \l__zrefclever_provide_lang_opt_default:nV
672             { cap } \l_keys_choice_tl
673         }
674         {
675             \l__zrefclever_provide_lang_opt_type:nV
676             { cap } \l_keys_choice_tl
677         }
678     } ,
679 cap .default:n = true ,
680 nocap .meta:n = { cap = false } ,
681 nocap .value_forbidden:n = true ,
682
683 abbrev .choices:nn =
684 { true , false }
685 {
686     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
687     {
688         {
689             \l__zrefclever_provide_lang_opt_default:nV
690             { abbrev } \l_keys_choice_tl
691         }
692         {
693             \l__zrefclever_provide_lang_opt_type:nV
694             { abbrev } \l_keys_choice_tl
695         }
696     } ,
697 abbrev .default:n = true ,

```

```

697     noabbrev .meta:n = { abbrev = false },
698     noabbrev .value_forbidden:n = true ,
699   }
700 \seq_map_inline:Nn
701   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
702   {
703     \keys_define:nn { zref-clever / langfile }
704     {
705       #1 .value_required:n = true ,
706       #1 .code:n =
707       {
708         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
709         { \__zrefclever_provide_lang_opt_default:nn {#1} {##1} }
710         {
711           \msg_info:nnn { zref-clever }
712             { option-not-type-specific } {#1}
713         }
714       } ,
715     }
716   }
717 \seq_map_inline:Nn
718   \c__zrefclever_ref_options_possibly_type_specific_seq
719   {
720     \keys_define:nn { zref-clever / langfile }
721     {
722       #1 .value_required:n = true ,
723       #1 .code:n =
724       {
725         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
726         { \__zrefclever_provide_lang_opt_default:nn {#1} {##1} }
727         { \__zrefclever_provide_lang_opt_type:nn {#1} {##1} }
728       } ,
729     }
730   }
731 \seq_map_inline:Nn
732   \c__zrefclever_ref_options_type_names_seq
733   {
734     \keys_define:nn { zref-clever / langfile }
735     {
736       #1 .value_required:n = true ,
737       #1 .code:n =
738       {
739         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
740         {
741           \msg_info:nnn { zref-clever }
742             { option-only-type-specific } {#1}
743         }
744         {
745           \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
746           { \__zrefclever_provide_lang_opt_type:nn {#1} {##1} }
747           {
748             \__zrefclever_provide_lang_opt_type:nn
749               { \l__zrefclever_lang_decl_case_tl - #1 } {##1}
750           }
751         }
752       }
753     }
754   }

```

```

751         }
752     } ,
753 }
754 }

```

Fallback

All “strings” queried with `__zrefclever_get_ref_opt_typeset: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 language-specific option, 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_opt_font:nN` – do not need to be provided here, since the later function sets an empty value if the option is not found.

```

755 \prop_new:N \g__zrefclever_fallback_unknown_lang_prop
756 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_unknown_lang_prop
757 {
758     tpairsep = {,~} ,
759     tlistsep = {,~} ,
760     tlastsep = {,~} ,
761     notesep = {~} ,
762     namesep = {\nobreakspace} ,
763     pairsep = {,~} ,
764     listsep = {,~} ,
765     lastsep = {,~} ,
766     rangesep = {\textendash} ,
767     preref = {} ,
768     postref = {} ,
769 }

```

Get language options

`__zrefclever_get_lang_opt_type:nnnNF` Get type-specific language option 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_lang_opt_type:nnnNF {\language} {\type} {\key}
\l variable {\false code}

770 \prg_new_protected_conditional:Npnn
771 \__zrefclever_get_lang_opt_type:nnnN #1#2#3#4 { F }
772 {
773     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
774     \l__zrefclever_base_language_tl
775     {
776         \prop_get:cnNTF
777         { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
778         { type- #2 - #3 } #4
779         { \prg_return_true: }

```



```

780         { \prg_return_false: }
781     }
782     { \prg_return_false: }
783 }
784 \prg_generate_conditional_variant:Nnn
785   \__zrefclever_get_lang_opt_type:nnnN { xxxN , xxnN } { F }

```

(End definition for __zrefclever_get_lang_opt_type:nnnNF.)

__zrefclever_get_lang_opt_default:nnNF Get default language option 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_lang_opt_default:nnNF { \langle language \rangle } { \langle key \rangle }
  \langle tl variable \rangle { \langle false code \rangle }

786 \prg_new_protected_conditional:Npnn
787   \__zrefclever_get_lang_opt_default:nnN #1#2#3 { F }
788 {
789   \prop_get:NnNTF \g__zrefclever_languages_prop { #1 }
790   \l__zrefclever_base_language_tl
791   {
792     \prop_get:cnNTF
793       { g__zrefclever_lang_ \l__zrefclever_base_language_tl _prop }
794       { default- #2 } #3
795       { \prg_return_true: }
796       { \prg_return_false: }
797   }
798   { \prg_return_false: }
799 }
800 \prg_generate_conditional_variant:Nnn
801   \__zrefclever_get_lang_opt_default:nnN { xnN } { F }

```

(End definition for __zrefclever_get_lang_opt_default:nnNF.)

__zrefclever_get_fallback_unknown_lang_opt:nnNF Get fallback language option 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_unknown_lang_opt:nnNF { \langle key \rangle }
  \langle tl variable \rangle { \langle false code \rangle }

802 % { \langle key \rangle } <tl var to set>
803 \prg_new_protected_conditional:Npnn
804   \__zrefclever_get_fallback_unknown_lang_opt:nnN #1#2 { F }
805 {
806   \prop_get:NnNTF \g__zrefclever_fallback_unknown_lang_prop
807     { #1 } #2
808     { \prg_return_true: }
809     { \prg_return_false: }
810 }

```

(End definition for __zrefclever_get_fallback_unknown_lang_opt: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 <property list> {\<key>} {\<value>}
811 \cs_new_protected:Npn \_zrefclever_prop_put_non_empty:Nnn #1#2#3
812 {
813   \tl_if_empty:nTF {#3}
814     { \prop_remove:Nn #1 {#2} }
815     { \prop_put:Nnn #1 {#2} {#3} }
816 }
```

(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.

```

817 \tl_new:N \l_zrefclever_ref_property_tl
818 \keys_define:nn { zref-clever / reference }
819 {
820   ref .code:n =
821   {
822     \zref@ifpropundefined {#1}
823     {
824       \msg_warning:nnn { zref-clever } { zref-property-undefined } {#1}
825       \tl_set:Nn \l_zrefclever_ref_property_tl { default }
826     }
827     { \tl_set:Nn \l_zrefclever_ref_property_tl {#1} }
828   } ,
829   ref .initial:n = default ,
830   ref .value_required:n = true ,
831   page .meta:n = { ref = page },
832   page .value_forbidden:n = true ,
833 }
```

typeset option

```

834 \bool_new:N \l_zrefclever_typeset_ref_bool
835 \bool_new:N \l_zrefclever_typeset_name_bool
836 \keys_define:nn { zref-clever / reference }
837 {
838   typeset .choice: ,
839   typeset / both .code:n =
```

```

840     {
841       \bool_set_true:N \l__zrefclever_typeset_ref_bool
842       \bool_set_true:N \l__zrefclever_typeset_name_bool
843     } ,
844     typeset / ref .code:n =
845     {
846       \bool_set_true:N \l__zrefclever_typeset_ref_bool
847       \bool_set_false:N \l__zrefclever_typeset_name_bool
848     } ,
849     typeset / name .code:n =
850     {
851       \bool_set_false:N \l__zrefclever_typeset_ref_bool
852       \bool_set_true:N \l__zrefclever_typeset_name_bool
853     } ,
854     typeset .initial:n = both ,
855     typeset .value_required:n = true ,
856
857     noname .meta:n = { typeset = ref } ,
858     noname .value_forbidden:n = true ,
859     noref .meta:n = { typeset = name } ,
860     noref .value_forbidden:n = true ,
861   }

```

sort option

```

862 \bool_new:N \l__zrefclever_typeset_sort_bool
863 \keys_define:nn { zref-clever / reference }
864 {
865   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
866   sort .initial:n = true ,
867   sort .default:n = true ,
868   nosort .meta:n = { sort = false } ,
869   nosort .value_forbidden:n = true ,
870 }

```

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.

```

871 \seq_new:N \l__zrefclever_typesort_seq
872 \keys_define:nn { zref-clever / reference }
873 {
874   typesort .code:n =
875   {
876     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
877     \seq_reverse:N \l__zrefclever_typesort_seq
878   } ,
879   typesort .initial:n =
880   { part , chapter , section , paragraph } ,
881   typesort .value_required:n = true ,
882   notypesort .code:n =
883   { \seq_clear:N \l__zrefclever_typesort_seq } ,
884   notypesort .value_forbidden:n = true ,

```

```
885 }
```

comp option

```
886 \bool_new:N \l__zrefclever_typeset_compress_bool
887 \keys_define:nn { zref-clever / reference }
888 {
889   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
890   comp .initial:n = true ,
891   comp .default:n = true ,
892   nocomp .meta:n = { comp = false },
893   nocomp .value_forbidden:n = true ,
894 }
```

range option

```
895 \bool_new:N \l__zrefclever_typeset_range_bool
896 \keys_define:nn { zref-clever / reference }
897 {
898   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
899   range .initial:n = false ,
900   range .default:n = true ,
901 }
```

cap and capfirst options

```
902 \bool_new:N \l__zrefclever_capitalize_first_bool
903 \keys_define:nn { zref-clever / reference }
904 {
905   cap .code:n =
906   {
907     \tl_if_empty:nTF {#1}
908     { \prop_remove:Nn \l__zrefclever_ref_options_prop { cap } }
909     {
910       \bool_lazy_or:nnTF
911       { \str_if_eq_p:nn {#1} { true } }
912       { \str_if_eq_p:nn {#1} { false } }
913       { \prop_put:Nnn \l__zrefclever_ref_options_prop { cap } {#1} }
914       {
915         \msg_warning:nnn { zref-clever }
916         { key-boolean-or-empty } {#1}
917       }
918     }
919   } ,
920   cap .default:n = true ,
921   nocap .meta:n = { cap = false } ,
922   nocap .value_forbidden:n = true ,
923
924   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
925   capfirst .initial:n = false ,
926   capfirst .default:n = true ,
927 }
```

abbrev and noabbrevfirst options

```
928 \bool_new:N \l__zrefclever_noabbrev_first_bool
929 \keys_define:nn { zref-clever / reference }
```

```

930 {
931   abbrev .code:n =
932     {
933       \tl_if_empty:nTF {#1}
934       { \prop_remove:Nn \l__zrefclever_ref_options_prop { abbrev } }
935       {
936         \bool_lazy_or:nnTF
937         { \str_if_eq_p:nn {#1} { true } }
938         { \str_if_eq_p:nn {#1} { false } }
939         { \prop_put:Nnn \l__zrefclever_ref_options_prop { abbrev } {#1} }
940         {
941           \msg_warning:nnn { zref-clever }
942           { key-boolean-or-empty } {#1}
943         }
944       }
945     } ,
946     abbrev .default:n = true ,
947     noabbrev .meta:n = { abbrev = false },
948     noabbrev .value_forbidden:n = true ,
949
950     noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
951     noabbrevfirst .initial:n = false ,
952     noabbrevfirst .default:n = true ,
953 }

```

S option

```

954 \keys_define:nn { zref-clever / reference }
955 {
956   S .meta:n =
957   { capfirst = true , noabbrevfirst = true },
958   S .value_forbidden:n = true ,
959 }

```

hyperref option

```

960 \bool_new:N \l__zrefclever_use_hyperref_bool
961 \bool_new:N \l__zrefclever_warn_hyperref_bool
962 \keys_define:nn { zref-clever / reference }
963 {
964   hyperref .choice: ,
965   hyperref / auto .code:n =
966   {
967     \bool_set_true:N \l__zrefclever_use_hyperref_bool
968     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
969   } ,
970   hyperref / true .code:n =
971   {
972     \bool_set_true:N \l__zrefclever_use_hyperref_bool
973     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
974   } ,
975   hyperref / false .code:n =
976   {
977     \bool_set_false:N \l__zrefclever_use_hyperref_bool
978     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
979   } ,

```

```

980     hyperref .initial:n = auto ,
981     hyperref .default:n = auto
982   }
983   \AddToHook { begindocument }
984   {
985     \__zrefclever_if_package_loaded:NTF { hyperref }
986     {
987       \bool_if:NT \l__zrefclever_use_hyperref_bool
988       { \RequirePackage { zref-hyperref } }
989     }
990     {
991       \bool_if:NT \l__zrefclever_warn_hyperref_bool
992       { \msg_warning:nn { zref-clever } { missing-hyperref } }
993       \bool_set_false:N \l__zrefclever_use_hyperref_bool
994     }
995     \keys_define:nn { zref-clever / reference }
996     {
997       hyperref .code:n =
998       { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
999     }
1000   }

```

nameinlink option

```

1001 \str_new:N \l__zrefclever_nameinlink_str
1002 \keys_define:nn { zref-clever / reference }
1003 {
1004   nameinlink .choice: ,
1005   nameinlink / true .code:n =
1006   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
1007   nameinlink / false .code:n =
1008   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
1009   nameinlink / single .code:n =
1010   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
1011   nameinlink / tsingle .code:n =
1012   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
1013   nameinlink .initial:n = tsingle ,
1014   nameinlink .default:n = true ,
1015 }

```

preposinlink option

```

1016 \bool_new:N \l__zrefclever_preposinlink_bool
1017 \keys_define:nn { zref-clever / reference }
1018 {
1019   preposinlink .bool_set:N = \l__zrefclever_preposinlink_bool ,
1020   preposinlink .initial:n = false ,
1021   preposinlink .default:n = true ,
1022 }

```

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 language file gets loaded, if it hadn't been already.

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

```

1023 \tl_new:N \l__zrefclever_ref_language_tl
1024 \tl_new:N \l__zrefclever_current_language_tl
1025 \tl_new:N \l__zrefclever_main_language_tl
1026 \AddToHook { begindocument }
1027 {
1028   \__zrefclever_if_package_loaded:nTF { babel }
1029   {
1030     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
1031     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
1032   }
1033   {
1034     \__zrefclever_if_package_loaded:nTF { polyglossia }
1035     {
1036       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
1037       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
1038     }
1039     {
1040       \tl_set:Nn \l__zrefclever_current_language_tl { english }
1041       \tl_set:Nn \l__zrefclever_main_language_tl { english }
1042     }
1043   }

```

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 language file loading is done verbosely, from when we are setting the default value (here), in which case the language file loading is done silently.

```

1044   \tl_set:Nn \l__zrefclever_ref_language_tl
1045   { \l__zrefclever_current_language_tl }
1046 }

```

```

1047 \keys_define:nn { zref-clever / reference }
1048 {
1049   lang .code:n =
1050   {
1051     \AddToHook { begindocument }
1052     {
1053       \str_case:nnF {#1}
1054       {
1055         { current }
1056         {
1057           \tl_set:Nn \l__zrefclever_ref_language_tl
1058             { \l__zrefclever_current_language_tl }
1059           \__zrefclever_provide_langfile_verbose:x
1060             { \l__zrefclever_ref_language_tl }
1061         }
1062       }
1063       { main }
1064       {
1065         \tl_set:Nn \l__zrefclever_ref_language_tl
1066           { \l__zrefclever_main_language_tl }
1067         \__zrefclever_provide_langfile_verbose:x
1068           { \l__zrefclever_ref_language_tl }
1069       }
1070     }
1071     {
1072       \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1073       { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
1074       {
1075         \msg_warning:nnn { zref-clever }
1076           { unknown-language-opt } {#1}
1077         \tl_set:Nn \l__zrefclever_ref_language_tl
1078           { \l__zrefclever_current_language_tl }
1079       }
1080       \__zrefclever_provide_langfile_verbose:x
1081         { \l__zrefclever_ref_language_tl }
1082     }
1083   }
1084 },
1085 lang .value_required:n = true ,
1086 }

1087 \AddToHook { begindocument / before }
1088 {
1089   \AddToHook { begindocument }
1090   {

```

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.

```

1091   \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body. Also, drop the verbose language file 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.

```

1092   \keys_define:nn { zref-clever / reference }

```



```

1093 {
1094   lang .code:n =
1095   {
1096     \str_case:nnF {#1}
1097     {
1098       { current }
1099       {
1100         \tl_set:Nn \l__zrefclever_ref_language_tl
1101           { \l__zrefclever_current_language_tl }
1102         \__zrefclever_provide_langfile:x
1103           { \l__zrefclever_ref_language_tl }
1104       }
1105
1106       { main }
1107       {
1108         \tl_set:Nn \l__zrefclever_ref_language_tl
1109           { \l__zrefclever_main_language_tl }
1110         \__zrefclever_provide_langfile:x
1111           { \l__zrefclever_ref_language_tl }
1112       }
1113     }
1114   {
1115     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
1116     { \tl_set:Nn \l__zrefclever_ref_language_tl {#1} }
1117     {
1118       \msg_warning:nnn { zref-clever }
1119         { unknown-language-opt } {#1}
1120       \tl_set:Nn \l__zrefclever_ref_language_tl
1121         { \l__zrefclever_current_language_tl }
1122     }
1123     \__zrefclever_provide_langfile:x
1124       { \l__zrefclever_ref_language_tl }
1125   }
1126   } ,
1127   lang .value_required:n = true ,
1128 }
1129 }
1130 }

```

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 xref package (<https://github.com/frougon/xref>), have been an insightful source to frame the problem in general terms.

```

1131 \tl_new:N \l__zrefclever_ref_decl_case_tl
1132 \keys_define:nn { zref-clever / reference }
1133 {
1134   d .code:n =
1135   { \msg_warning:nnn { zref-clever } { option-document-only } { d } } ,
1136 }

```

```

1137 \AddToHook { begindocument }
1138 {
1139   \keys_define:nn { zref-clever / reference }
1140   {

```

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

```

1141     d .tl_set:N = \l__zrefclever_ref_decl_case_tl ,
1142     d .value_required:n = true ,
1143   }
1144 }

```

nudge & co. options

```

1145 \bool_new:N \l__zrefclever_nudge_enabled_bool
1146 \bool_new:N \l__zrefclever_nudge_multitype_bool
1147 \bool_new:N \l__zrefclever_nudge_comptosing_bool
1148 \bool_new:N \l__zrefclever_nudge_singular_bool
1149 \bool_new:N \l__zrefclever_nudge_gender_bool
1150 \tl_new:N \l__zrefclever_ref_gender_tl
1151 \keys_define:nn { zref-clever / reference }
1152 {
1153   nudge .choice: ,
1154   nudge / true .code:n =
1155     { \bool_set_true:N \l__zrefclever_nudge_enabled_bool } ,
1156   nudge / false .code:n =
1157     { \bool_set_false:N \l__zrefclever_nudge_enabled_bool } ,
1158   nudge / ifdraft .code:n =
1159     {
1160       \ifdraft
1161         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1162         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1163       } ,
1164   nudge / iffinaal .code:n =
1165     {
1166       \ifoptionfinal
1167         { \bool_set_true:N \l__zrefclever_nudge_enabled_bool }
1168         { \bool_set_false:N \l__zrefclever_nudge_enabled_bool }
1169       } ,
1170   nudge .initial:n = false ,
1171   nudge .default:n = true ,
1172   nonnudge .meta:n = { nudge = false } ,
1173   nonnudge .value_forbidden:n = true ,
1174   nudgeif .code:n =
1175     {
1176       \bool_set_false:N \l__zrefclever_nudge_multitype_bool
1177       \bool_set_false:N \l__zrefclever_nudge_comptosing_bool
1178       \bool_set_false:N \l__zrefclever_nudge_gender_bool
1179       \clist_map_inline:nn {#1}
1180       {
1181         \str_case:nnF {##1}
1182         {
1183           { multitype }
1184           { \bool_set_true:N \l__zrefclever_nudge_multitype_bool }

```

```

1185         { comptosing }
1186         { \bool_set_true:N \l__zrefclever_nudge_comptosing_bool }
1187         { gender }
1188         { \bool_set_true:N \l__zrefclever_nudge_gender_bool }
1189         { all }
1190         {
1191             \bool_set_true:N \l__zrefclever_nudge_multitype_bool
1192             \bool_set_true:N \l__zrefclever_nudge_comptosing_bool
1193             \bool_set_true:N \l__zrefclever_nudge_gender_bool
1194         }
1195     }
1196     {
1197         \msg_warning:nnn { zref-clever }
1198         { nudgeif-unknown-value } {##1}
1199     }
1200 }
1201 },
1202 nudgeif .value_required:n = true ,
1203 nudgeif .initial:n = all ,
1204 sg .bool_set:N = \l__zrefclever_nudge_singular_bool ,
1205 sg .initial:n = false ,
1206 sg .default:n = true ,
1207 g .code:n =
1208     { \msg_warning:nnn { zref-clever } { option-document-only } { g } } ,
1209 }
1210 \AddToHook { begindocument }
1211 {
1212     \keys_define:nn { zref-clever / reference }
1213     {

```

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

```

1214         g .tl_set:N = \l__zrefclever_ref_gender_tl ,
1215         g .value_required:n = true ,
1216     }
1217 }

```

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>).

```

1218 \tl_new:N \l__zrefclever_ref_typeset_font_tl
1219 \keys_define:nn { zref-clever / reference }
1220 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

titleref option

```

1221 \keys_define:nn { zref-clever / reference }
1222 {
1223     titleref .code:n = { \RequirePackage { zref-titleref } } ,
1224     titleref .value_forbidden:n = true ,
1225 }

```

```

1226 \AddToHook { begindocument }
1227 {
1228   \keys_define:nn { zref-clever / reference }
1229   {
1230     titleref .code:n =
1231       { \msg_warning:nn { zref-clever } { titleref-preamble-only } }
1232   }
1233 }

```

note option

```

1234 \tl_new:N \l__zrefclever_zcref_note_tl
1235 \keys_define:nn { zref-clever / reference }
1236 {
1237   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
1238   note .value_required:n = true ,
1239 }

```

check option

Integration with zref-check.

```

1240 \bool_new:N \l__zrefclever_zrefcheck_available_bool
1241 \bool_new:N \l__zrefclever_zcref_with_check_bool
1242 \keys_define:nn { zref-clever / reference }
1243 {
1244   check .code:n = { \RequirePackage { zref-check } } ,
1245   check .value_forbidden:n = true ,
1246 }
1247 \AddToHook { begindocument }
1248 {
1249   \__zrefclever_if_package_loaded:nTF { zref-check }
1250   {
1251     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
1252     \keys_define:nn { zref-clever / reference }
1253     {
1254       check .code:n =
1255       {
1256         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
1257         \keys_set:nn { zref-check / zcheck } {#1}
1258       } ,
1259       check .value_required:n = true ,
1260     }
1261   }
1262   {
1263     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
1264     \keys_define:nn { zref-clever / reference }
1265     {
1266       check .value_forbidden:n = false ,
1267       check .code:n =
1268       { \msg_warning:nn { zref-clever } { missing-zref-check } } ,
1269     }
1270   }
1271 }

```

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`.

```
1272 \prop_new:N \l__zrefclever_counter_type_prop
1273 \keys_define:nn { zref-clever / label }
1274 {
1275     countertype .code:n =
1276     {
1277         \keyval_parse:nnn
1278         {
1279             \msg_warning:nnnn { zref-clever }
1280             { key-requires-value } { countertype }
1281         }
1282         {
1283             \__zrefclever_prop_put_non_empty:Nnn
1284             \l__zrefclever_counter_type_prop
1285         }
1286         {#1}
1287     } ,
1288     countertype .value_required:n = true ,
1289     countertype .initial:n =
1290     {
1291         subsection      = section ,
1292         subsubsection   = section ,
1293         subparagraph    = paragraph ,
1294         enumi           = item ,
1295         enumii          = item ,
1296         enumiii         = item ,
1297         enumiv          = item ,
1298         mpfootnote     = footnote ,
1299     } ,
1300 }
```

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.

```
1301 \seq_new:N \l__zrefclever_counter_resetters_seq
1302 \keys_define:nn { zref-clever / label }
1303 {
1304   counterresetters .code:n =
1305   {
1306     \clist_map_inline:nn {#1}
1307     {
1308       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
1309       {
1310         \seq_put_right:Nn
1311         \l__zrefclever_counter_resetters_seq {##1}
1312       }
1313     }
1314   } ,
1315   counterresetters .initial:n =
1316   {
1317     part ,
1318     chapter ,
1319     section ,
1320     subsection ,
1321     subsubsection ,
1322     paragraph ,
1323     subparagraph ,
1324   },
1325   counterresetters .value_required:n = true ,
1326 }
```

counterresetby option

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

```
1327 \prop_new:N \l__zrefclever_counter_resetby_prop
1328 \keys_define:nn { zref-clever / label }
1329 {
1330   counterresetby .code:n =
1331   {
1332     \keyval_parse:nnn
1333     {
1334       \msg_warning:nnn { zref-clever }
1335       { key-requires-value } { counterresetby }
1336     }
1337   }
```

```

1337         {
1338             \__zrefclever_prop_put_non_empty:Nnn
1339             \l__zrefclever_counter_resetby_prop
1340         }
1341         {#1}
1342     } ,
1343     counterresetby .value_required:n = true ,
1344     counterresetby .initial:n =
1345     {

```

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.

```

1346         enumii = enumi ,
1347         enumiii = enumii ,
1348         enumiv = enumiii ,
1349     } ,
1350 }

```

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.

```

1351 \tl_new:N \l__zrefclever_current_counter_tl
1352 \keys_define:nn { zref-clever / label }
1353 {
1354     currentcounter .tl_set:N = \l__zrefclever_current_counter_tl ,
1355     currentcounter .value_required:n = true ,
1356     currentcounter .initial:n = \@currentcounter ,
1357 }

```

nocompat option

```

1358 \bool_new:N \g__zrefclever_nocompat_bool
1359 \seq_new:N \g__zrefclever_nocompat_modules_seq
1360 \keys_define:nn { zref-clever / reference }
1361 {
1362     nocompat .code:n =
1363     {
1364         \tl_if_empty:nTF {#1}
1365         { \bool_gset_true:N \g__zrefclever_nocompat_bool }
1366         {
1367             \clist_map_inline:nn {#1}
1368             {
1369                 \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {##1}
1370                 {
1371                     \seq_gput_right:Nn
1372                     \g__zrefclever_nocompat_modules_seq {##1}
1373                 }
1374             }
1375         }

```

```

1376     } ,
1377   }
1378   \AddToHook { begindocument }
1379   {
1380     \keys_define:nn { zref-clever / reference }
1381     {
1382       nocompat .code:n =
1383       {
1384         \msg_warning:nnn { zref-clever }
1385         { option-preamble-only } { nocompat }
1386       }
1387     }
1388   }
1389   \AtEndOfPackage
1390   {
1391     \AddToHook { begindocument }
1392     {
1393       \seq_map_inline:Nn \g__zrefclever_nocompat_modules_seq
1394       { \msg_warning:nnn { zref-clever } { unknown-compat-module } {#1} }
1395     }
1396   }

```

`_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>}

1397 \cs_new_protected:Npn \__zrefclever_compat_module:nn #1#2
1398 {
1399   \AddToHook { begindocument }
1400   {
1401     \bool_if:NF \g__zrefclever_nocompat_bool
1402     { \seq_if_in:NnF \g__zrefclever_nocompat_modules_seq {#1} {#2} }
1403     \seq_gremove_all:Nn \g__zrefclever_nocompat_modules_seq {#1}
1404   }
1405 }

```

(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_opt_typeset:nN` and `__zrefclever_get_ref_opt_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.

```

1406 \prop_new:N \l__zrefclever_ref_options_prop
1407 \seq_map_inline:Nn
1408   \c__zrefclever_ref_options_reference_seq
1409   {
1410     \keys_define:nn { zref-clever / reference }
1411     {
1412       #1 .default:V = \c_novalue_tl ,
1413       #1 .code:n =
1414       {
1415         \tl_if_novalue:nTF {##1}
1416         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
1417         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
1418       } ,
1419     }
1420   }
1421 \keys_define:nn { zref-clever / reference }
1422 {
1423   refpre .code:n =
1424   {
1425     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1426     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1427     { refpre } { preref }
1428   } ,
1429   refpos .code:n =
1430   {
1431     % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1432     \msg_warning:nnnn { zref-clever }{ option-deprecated }
1433     { refpos } { postref }
1434   } ,
1435 }

```

Package options

The options have been separated in two different groups, so that we can potentially apply them selectively to different contexts: **label** and **reference**. Currently, the only use of this selection is the ability to exclude label related options from `\zceref`'s options. Anyway, for 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.

```

1436 \keys_define:nn { }
1437 {
1438   zref-clever / zcsetup .inherit:n =
1439   {
1440     zref-clever / label ,
1441     zref-clever / reference ,
1442   }
1443 }

```

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

```
1444 \ProcessKeysOptions { zref-clever / zcsetup }
```

5 Configuration

5.1 \zcsetup

\zcsetup Provide \zcsetup.

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

```
1445 \NewDocumentCommand \zcsetup { m }
1446 { \__zrefclever_zcsetup:n {#1} }
```

(End definition for \zcsetup.)

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

```
\__zrefclever_zcsetup:n{<options>}
```

```
1447 \cs_new_protected:Npn \__zrefclever_zcsetup:n #1
1448 { \keys_set:nn { zref-clever / zcsetup } {#1} }
1449 \cs_generate_variant:Nn \__zrefclever_zcsetup:n { x }
```

(End definition for __zrefclever_zcsetup:n.)

5.2 \zcRefTypeSetup

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

```
\zcRefTypeSetup \zcRefTypeSetup {<type>} {<options>}
1450 \NewDocumentCommand \zcRefTypeSetup { m m }
1451 {
1452   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
1453   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
1454   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
1455   \keys_set:nn { zref-clever / typesetup } {#2}
1456 }
```

(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 language-specific options, 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 language-specific options 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>.

```

1457 \keys_define:nn { zref-clever / typesetup }
1458 {
1459   cap .code:n =
1460   {
1461     \tl_if_empty:nTF {#1}
1462     {
1463       \prop_remove:cn
1464       {
1465         l__zrefclever_type_
1466         \l__zrefclever_setup_type_tl _options_prop
1467       }
1468       { cap }
1469     }
1470     {
1471       \bool_lazy_or:nnTF
1472       { \str_if_eq_p:nn {#1} { true } }
1473       { \str_if_eq_p:nn {#1} { false } }
1474       {
1475         \prop_put:cnn
1476         {
1477           l__zrefclever_type_
1478           \l__zrefclever_setup_type_tl _options_prop
1479         }
1480         { cap } {#1}
1481       }
1482       {
1483         \msg_warning:nnn { zref-clever }
1484         { key-boolean-or-empty } {#1}
1485       }
1486     }
1487   } ,
1488   cap .default:n = true ,
1489   nocap .meta:n = { cap = false } ,
1490   nocap .value_forbidden:n = true ,
1491
1492   abbrev .code:n =
1493   {
1494     \tl_if_empty:nTF {#1}
1495     {
1496       \prop_remove:cn
1497       {
1498         l__zrefclever_type_
1499         \l__zrefclever_setup_type_tl _options_prop
1500       }
1501       { abbrev }
1502     }

```

```

1503     {
1504         \bool_lazy_or:nnTF
1505         { \str_if_eq_p:nn {#1} { true } }
1506         { \str_if_eq_p:nn {#1} { false } }
1507     {
1508         \prop_put:cnn
1509         {
1510             l__zrefclever_type_
1511             \l__zrefclever_setup_type_tl _options_prop
1512         }
1513         { abbrev } {#1}
1514     }
1515     {
1516         \msg_warning:nnn { zref-clever }
1517         { key-boolean-or-empty } {#1}
1518     }
1519 }
1520 },
1521 abbrev .default:n = true ,
1522 noabbrev .meta:n = { abbrev = false },
1523 noabbrev .value_forbidden:n = true ,
1524 }
1525 \seq_map_inline:Nn
1526 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1527 {
1528     \keys_define:nn { zref-clever / typesetup }
1529     {
1530         #1 .code:n =
1531         {
1532             \msg_warning:nnn { zref-clever }
1533             { option-not-type-specific } {#1}
1534         } ,
1535     }
1536 }
1537 \seq_map_inline:Nn
1538 \c__zrefclever_ref_options_typesetup_seq
1539 {
1540     \keys_define:nn { zref-clever / typesetup }
1541     {
1542         #1 .default:V = \c_novaluel_tl ,
1543         #1 .code:n =
1544         {
1545             \tl_if_novalue:nTF {##1}
1546             {
1547                 \prop_remove:cn
1548                 {
1549                     l__zrefclever_type_
1550                     \l__zrefclever_setup_type_tl _options_prop
1551                 }
1552                 {#1}
1553             }
1554             {
1555                 \prop_put:cnn

```

```

1556         {
1557             l__zrefclever_type_
1558             \l__zrefclever_setup_type_tl _options_prop
1559         }
1560         {#1} {##1}
1561     }
1562 } ,
1563 }
1564 }
1565 \keys_define:nn { zref-clever / typesetup }
1566 {
1567     refpre .code:n =
1568     {
1569         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1570         \msg_warning:nnnn { zref-clever }{ option-deprecated }
1571         { refpre } { preref }
1572     } ,
1573     refpos .code:n =
1574     {
1575         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1576         \msg_warning:nnnn { zref-clever }{ option-deprecated }
1577         { refpos } { postref }
1578     } ,
1579 }

```

5.3 \zcLanguageSetup

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

```

\zcLanguageSetup      \zcLanguageSetup{<language>}{<options>}
1580 \NewDocumentCommand \zcLanguageSetup { m m }
1581 {
1582     \group_begin:
1583     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
1584     \l__zrefclever_base_language_tl
1585     {
1586         \tl_clear:N \l__zrefclever_setup_type_tl
1587         \exp_args:NNx \seq_set_from_clist:Nn
1588         \l__zrefclever_lang_declension_seq
1589         {
1590             \prop_item:cn
1591             {
1592                 g__zrefclever_lang_
1593                 \l__zrefclever_base_language_tl _prop
1594             }
1595             { declension }
1596         }

```

```

1597 \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1598 { \tl_clear:N \l__zrefclever_lang_decl_case_tl }
1599 {
1600   \seq_get_left:NN \l__zrefclever_lang_declension_seq
1601   \l__zrefclever_lang_decl_case_tl
1602 }
1603 \exp_args:NNx \seq_set_from_clist:Nn
1604 \l__zrefclever_lang_gender_seq
1605 {
1606   \prop_item:cn
1607   {
1608     g__zrefclever_lang_
1609     \l__zrefclever_base_language_tl _prop
1610   }
1611   { gender }
1612 }
1613 \keys_set:nn { zref-clever / langsetup } {#2}
1614 }
1615 { \msg_warning:nnn { zref-clever } { unknown-language-setup } {#1} }
1616 \group_end:
1617 }
1618 \@onlypreamble \zcLanguageSetup

```

(End definition for \zcLanguageSetup.)

_zrefclever_declare_lang_opt_type:nnnn A couple of auxiliary functions for the of zref-clever/langsetup keys set in \zcLanguageSetup.
_zrefclever_declare_lang_opt_default:nnn They respectively declare (unconditionally set) “type-specific” and “default” language-specific options.

```

\__zrefclever_declare_lang_opt_type:nnnn {<language>} {<type>}
{<key>} {<value>}
\__zrefclever_declare_lang_opt_default:nnn {<language>}
{<key>} {<value>}

1619 \cs_new_protected:Npn \__zrefclever_declare_lang_opt_type:nnnn #1#2#3#4
1620 {
1621   \prop_gput:cnn { g__zrefclever_lang_ #1 _prop }
1622   { type- #2 - #3 } {#4}
1623 }
1624 \cs_generate_variant:Nn
1625 \__zrefclever_declare_lang_opt_type:nnnn { VVnn , VVxn , VVnx , VVnV }
1626 \cs_new_protected:Npn \__zrefclever_declare_lang_opt_default:nnn #1#2#3
1627 {
1628   \prop_gput:cnn { g__zrefclever_lang_ #1 _prop }
1629   { default- #2 } {#3}
1630 }
1631 \cs_generate_variant:Nn \__zrefclever_declare_lang_opt_default:nnn { Vnn , VnV }

```

(End definition for __zrefclever_declare_lang_opt_type:nnnn and __zrefclever_declare_lang_opt_default:nnn.)

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

```

1632 \keys_define:nn { zref-clever / langsetup }
1633 {
1634   type .code:n =

```

```

1635     {
1636       \tl_if_empty:NTF {#1}
1637       { \tl_clear:N \l__zrefclever_setup_type_tl }
1638       { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
1639     } ,
1640
1641   case .code:n =
1642   {
1643     \seq_if_empty:NTF \l__zrefclever_lang_declension_seq
1644     {
1645       \msg_warning:nxxx { zref-clever } { language-no-decl-setup }
1646       { \l__zrefclever_base_language_tl } {#1}
1647     }
1648     {
1649       \seq_if_in:NnTF \l__zrefclever_lang_declension_seq {#1}
1650       { \tl_set:Nn \l__zrefclever_lang_decl_case_tl {#1} }
1651       {
1652         \msg_warning:nxxx { zref-clever } { unknown-decl-case }
1653         {#1} { \l__zrefclever_base_language_tl }
1654         \seq_get_left:NN \l__zrefclever_lang_declension_seq
1655         \l__zrefclever_lang_decl_case_tl
1656       }
1657     }
1658   } ,
1659   case .value_required:n = true ,
1660
1661   gender .code:n =
1662   {
1663     \seq_if_empty:NTF \l__zrefclever_lang_gender_seq
1664     {
1665       \msg_warning:nxxxx { zref-clever } { language-no-gender }
1666       { \l__zrefclever_base_language_tl } { gender } {#1}
1667     }
1668     {
1669       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1670       {
1671         \msg_warning:nnn { zref-clever }
1672         { option-only-type-specific } { gender }
1673       }
1674       {
1675         \clist_clear:N \l_tmpa_clist
1676         \clist_map_inline:nn {#1}
1677         {
1678           \seq_if_in:NnTF \l__zrefclever_lang_gender_seq {##1}
1679           { \clist_put_right:Nn \l_tmpa_clist {##1} }
1680           {
1681             \msg_warning:nxxx { zref-clever }
1682             { gender-not-declared }
1683             { \l__zrefclever_base_language_tl } {##1}
1684           }
1685         }
1686         \clist_if_empty:NF \l_tmpa_clist
1687         {
1688           \__zrefclever_declare_lang_opt_type:VVnx

```

```

1689         \l__zrefclever_base_language_tl
1690         \l__zrefclever_setup_type_tl
1691         { gender } { \clist_use:Nn \l_tmpa_clist { , } }
1692     }
1693 }
1694 }
1695 },
1696 gender .value_required:n = true ,
1697
1698 cap .choices:nn =
1699 { true , false }
1700 {
1701     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1702     {
1703         \__zrefclever_declare_lang_opt_default:VnV
1704         \l__zrefclever_base_language_tl
1705         { cap } \l_keys_choice_tl
1706     }
1707     {
1708         \__zrefclever_declare_lang_opt_type:VVnV
1709         \l__zrefclever_base_language_tl
1710         \l__zrefclever_setup_type_tl
1711         { cap } \l_keys_choice_tl
1712     }
1713 },
1714 cap .default:n = true ,
1715 nocap .meta:n = { cap = false } ,
1716 nocap .value_forbidden:n = true ,
1717
1718 abbrev .choices:nn =
1719 { true , false }
1720 {
1721     \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1722     {
1723         \__zrefclever_declare_lang_opt_default:VnV
1724         \l__zrefclever_base_language_tl
1725         { abbrev } \l_keys_choice_tl
1726     }
1727     {
1728         \__zrefclever_declare_lang_opt_type:VVnV
1729         \l__zrefclever_base_language_tl
1730         \l__zrefclever_setup_type_tl
1731         { abbrev } \l_keys_choice_tl
1732     }
1733 },
1734 abbrev .default:n = true ,
1735 noabbrev .meta:n = { abbrev = false } ,
1736 noabbrev .value_forbidden:n = true ,
1737 }
1738 \seq_map_inline:Nn
1739 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
1740 {
1741     \keys_define:nn { zref-clever / langsetup }
1742     {

```



```

1743     #1 .value_required:n = true ,
1744     #1 .code:n =
1745     {
1746         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1747         {
1748             \__zrefclever_declare_lang_opt_default:Vnn
1749             \l__zrefclever_base_language_tl
1750             {#1} {##1}
1751         }
1752         {
1753             \msg_warning:nnn { zref-clever }
1754             { option-not-type-specific } {#1}
1755         }
1756     } ,
1757 }
1758 }
1759 \seq_map_inline:Nn
1760 \c__zrefclever_ref_options_possibly_type_specific_seq
1761 {
1762     \keys_define:nn { zref-clever / langsetup }
1763     {
1764         #1 .value_required:n = true ,
1765         #1 .code:n =
1766         {
1767             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1768             {
1769                 \__zrefclever_declare_lang_opt_default:Vnn
1770                 \l__zrefclever_base_language_tl
1771                 {#1} {##1}
1772             }
1773             {
1774                 \__zrefclever_declare_lang_opt_type:VVnn
1775                 \l__zrefclever_base_language_tl
1776                 \l__zrefclever_setup_type_tl
1777                 {#1} {##1}
1778             }
1779         } ,
1780     }
1781 }
1782 \keys_define:nn { zref-clever / langsetup }
1783 {
1784     refpre .code:n =
1785     {
1786         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1787         \msg_warning:nnnn { zref-clever }{ option-deprecated }
1788         { refpre } { preref }
1789     } ,
1790     refpos .code:n =
1791     {
1792         % NOTE Option deprecated in 2022-01-10 for v0.1.2-alpha.
1793         \msg_warning:nnnn { zref-clever }{ option-deprecated }
1794         { refpos } { postref }
1795     } ,
1796 }

```

```

1797 \seq_map_inline:Nn
1798   \c__zrefclever_ref_options_type_names_seq
1799   {
1800     \keys_define:nn { zref-clever / langsetup }
1801     {
1802       #1 .value_required:n = true ,
1803       #1 .code:n =
1804       {
1805         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1806         {
1807           \msg_warning:nnn { zref-clever }
1808             { option-only-type-specific } {#1}
1809         }
1810         {
1811           \tl_if_empty:NTF \l__zrefclever_lang_decl_case_tl
1812           {
1813             \__zrefclever_declare_lang_opt_type:VVnn
1814               \l__zrefclever_base_language_tl
1815               \l__zrefclever_setup_type_tl
1816               {#1} {##1}
1817           }
1818           {
1819             \__zrefclever_declare_lang_opt_type:VVxn
1820               \l__zrefclever_base_language_tl
1821               \l__zrefclever_setup_type_tl
1822               { \l__zrefclever_lang_decl_case_tl - #1 } {##1}
1823           }
1824         }
1825       } ,
1826     }
1827 }

```

6 User interface

6.1 \zcref

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

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

```

1828 \NewDocumentCommand \zcref { s O { } m }
1829 { \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>}
```

```

1830 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1831 {
1832   \group_begin:

```

Set options.

```
1833 \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```
1834 \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1835 \bool_set:Nn \l__zrefclever_link_star_bool {#2}
```

Ensure language file for reference language is loaded, if available. We cannot rely on `\keys_set:nn` for the task, since if the `lang` option is set for `current`, the actual language may have changed outside our control. `__zrefclever_provide_langfile:x` does nothing if the language file is already loaded.

```
1836 \__zrefclever_provide_langfile:x { \l__zrefclever_ref_language_tl }
```

Process `\zcDeclareLanguage` options.

```
1837 \__zrefclever_process_language_options:
```

Integration with `zref-check`.

```
1838 \bool_lazy_and:nnT
1839 { \l__zrefclever_zrefcheck_available_bool }
1840 { \l__zrefclever_zcref_with_check_bool }
1841 { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1842 \bool_lazy_or:nnT
1843 { \l__zrefclever_typeset_sort_bool }
1844 { \l__zrefclever_typeset_range_bool }
1845 { \__zrefclever_sort_labels: }
```

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

```
1846 \group_begin:
1847 \l__zrefclever_ref_typeset_font_tl
1848 \__zrefclever_typeset_refs:
1849 \group_end:
```

Typeset note.

```
1850 \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1851 {
1852   \__zrefclever_get_ref_opt_typeset:nN { notesep } \l_tmpa_tl
1853   \l_tmpa_tl
1854   \l__zrefclever_zcref_note_tl
1855 }
```

Integration with `zref-check`.

```
1856 \bool_lazy_and:nnT
1857 { \l__zrefclever_zrefcheck_available_bool }
1858 { \l__zrefclever_zcref_with_check_bool }
1859 {
1860   \zrefcheck_zcref_end_label_maybe:
1861   \zrefcheck_zcref_run_checks_on_labels:n
1862     { \l__zrefclever_zcref_labels_seq }
1863 }
```

Integration with mathtools.

```

1864 \bool_if:NT \l__zrefclever_mathtools_showonlyrefs_bool
1865 {
1866   \__zrefclever_mathtools_showonlyrefs:n
1867   { \l__zrefclever_zcref_labels_seq }
1868 }
1869 \group_end:
1870 }

```

(End definition for __zrefclever_zcref:nnnn.)

\l__zrefclever_zcref_labels_seq

\l__zrefclever_link_star_bool

```

1871 \seq_new:N \l__zrefclever_zcref_labels_seq
1872 \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{*}[\langle options \rangle]{\langle labels \rangle}

```

1873 \NewDocumentCommand \zcpageref { s O { } m }
1874 {
1875   \IfBooleanTF {#1}
1876   { \zcref*{#2, ref = page} {#3} }
1877   { \zcref [ #2, ref = page] {#3} }
1878 }

```

(End definition for \zcpageref.)

7 Sorting

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

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_a_tl

\l__zrefclever_label_type_b_tl

\l__zrefclever_label_enclval_a_tl

\l__zrefclever_label_enclval_b_tl

\l__zrefclever_label_extdoc_a_tl

\l__zrefclever_label_extdoc_b_tl

```

1879 \tl_new:N \l__zrefclever_label_type_a_tl
1880 \tl_new:N \l__zrefclever_label_type_b_tl
1881 \tl_new:N \l__zrefclever_label_enclval_a_tl
1882 \tl_new:N \l__zrefclever_label_enclval_b_tl
1883 \tl_new:N \l__zrefclever_label_extdoc_a_tl
1884 \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.

```
1885 \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`

```
1886 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
1887 \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 `\zcref`. This variable is populated by `__zrefclever_label_type_put_new_right:n` at the start of `__zrefclever_sort_labels:`. This order is required as a “last resort” sort criterion between the reference types, for use in `__zrefclever_sort_default_different_types:nn`.

```
1888 \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_zcref:nnnn` where a number of environment variables are to be set appropriately. In particular, `\l__zrefclever_zcref_labels_seq` should contain the labels received as argument to `\zcref`, and the function performs its task by sorting this variable.

```
1889 \cs_new_protected:Npn \__zrefclever_sort_labels:
1890 {
```

Store label types sequence.

```
1891   \seq_clear:N \l__zrefclever_label_types_seq
1892   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1893   {
1894     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1895     \__zrefclever_label_type_put_new_right:n
1896   }
```

Sort.

```
1897   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1898   {
1899     \zref@ifrefundefined {##1}
1900     {
1901       \zref@ifrefundefined {##2}
1902       {
1903         % Neither label is defined.
1904         \sort_return_same:
1905       }
1906     {
1907       % The second label is defined, but the first isn't, leave the
1908       % undefined first (to be more visible).
1909       \sort_return_same:
```

```

1910     }
1911   }
1912   {
1913     \zref@ifrefundefined {##2}
1914     {
1915       % The first label is defined, but the second isn't, bring the
1916       % second forward.
1917       \sort_return_swapped:
1918     }
1919     {
1920       % The interesting case: both labels are defined. References
1921       % to the "default" property or to the "page" are quite
1922       % different with regard to sorting, so we branch them here to
1923       % specialized functions.
1924       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1925       { \__zrefclever_sort_page:nn {##1} {##2} }
1926       { \__zrefclever_sort_default:nn {##1} {##2} }
1927     }
1928   }
1929 }
1930 }

```

(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>}

1931 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1932 {
1933   \__zrefclever_extract_default:Nnnn
1934   \l__zrefclever_label_type_a_tl {#1} {zc@type} { \c_empty_tl }
1935   \seq_if_in:NVF \l__zrefclever_label_types_seq
1936   \l__zrefclever_label_type_a_tl
1937   {
1938     \seq_put_right:NV \l__zrefclever_label_types_seq
1939     \l__zrefclever_label_type_a_tl
1940   }
1941 }

```

(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:.

```

1942 \cs_new_protected:Npn \__zrefclever_sort_default:nn {<label a>} {<label b>}
1943 {
1944   \__zrefclever_extract_default:Nnnn
1945   \l__zrefclever_label_type_a_tl {#1} {zc@type} {zc@missingtype}
1946   \__zrefclever_extract_default:Nnnn
1947   \l__zrefclever_label_type_b_tl {#2} {zc@type} {zc@missingtype}
1948
1949   \tl_if_eq:NNTF
1950   \l__zrefclever_label_type_a_tl
1951   \l__zrefclever_label_type_b_tl
1952   { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1953   { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1954 }

```

(End definition for __zrefclever_sort_default:nn.)

```

\__zrefclever_sort_default_same_type:nn \__zrefclever_sort_default_same_type:nn {<label a>} {<label b>}
1955 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1956 {
1957   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_a_tl
1958   {#1} {zc@enclval} {\c_empty_tl}
1959   \tl_reverse:N \l__zrefclever_label_enclval_a_tl
1960   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_enclval_b_tl
1961   {#2} {zc@enclval} {\c_empty_tl}
1962   \tl_reverse:N \l__zrefclever_label_enclval_b_tl
1963   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
1964   {#1} {externaldocument} {\c_empty_tl}
1965   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
1966   {#2} {externaldocument} {\c_empty_tl}
1967
1968   \bool_set_false:N \l__zrefclever_sort_decided_bool
1969
1970   % First we check if there's any "external document" difference (coming
1971   % from 'zref-xr') and, if so, sort based on that.
1972   \tl_if_eq:NNTF
1973   \l__zrefclever_label_extdoc_a_tl
1974   \l__zrefclever_label_extdoc_b_tl
1975   {
1976     \bool_if:nTF
1977     {
1978       \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1979       ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl
1980     }
1981     {
1982       \bool_set_true:N \l__zrefclever_sort_decided_bool
1983       \sort_return_same:
1984     }
1985     {
1986       \bool_if:nTF
1987       {
1988         ! \tl_if_empty_p:V \l__zrefclever_label_extdoc_a_tl &&
1989         \tl_if_empty_p:V \l__zrefclever_label_extdoc_b_tl

```

```

1990     }
1991     {
1992         \bool_set_true:N \l__zrefclever_sort_decided_bool
1993         \sort_return_swapped:
1994     }
1995     {
1996         \bool_set_true:N \l__zrefclever_sort_decided_bool
1997         % Two different "external documents": last resort, sort by the
1998         % document name itself.
1999         \str_compare:eNeTF
2000         { \l__zrefclever_label_extdoc_b_tl } <
2001         { \l__zrefclever_label_extdoc_a_tl }
2002         { \sort_return_swapped: }
2003         { \sort_return_same:    }
2004     }
2005 }
2006 }
2007
2008 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
2009 {
2010     \bool_if:nTF
2011     {
2012         % Both are empty: neither label has any (further) "enclosing
2013         % counters" (left).
2014         \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl &&
2015         \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
2016     }
2017     {
2018         \bool_set_true:N \l__zrefclever_sort_decided_bool
2019         \int_compare:nNnTF
2020         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
2021         >
2022         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
2023         { \sort_return_swapped: }
2024         { \sort_return_same:    }
2025     }
2026     {
2027         \bool_if:nTF
2028         {
2029             % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
2030             \tl_if_empty_p:V \l__zrefclever_label_enclval_a_tl
2031         }
2032         {
2033             \bool_set_true:N \l__zrefclever_sort_decided_bool
2034             \int_compare:nNnTF
2035             { \__zrefclever_extract:nnn {#1} { zc@cntval } { } }
2036             >
2037             { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2038             { \sort_return_swapped: }
2039             { \sort_return_same:    }
2040         }
2041     }
2042     \bool_if:nTF
2043     {

```



```

2044 % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
2045 \tl_if_empty_p:V \l__zrefclever_label_enclval_b_tl
2046 }
2047 {
2048   \bool_set_true:N \l__zrefclever_sort_decided_bool
2049   \int_compare:nNnTF
2050     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2051     <
2052     { \__zrefclever_extract:nnn {#2} { zc@cntval } { } }
2053     { \sort_return_same: }
2054     { \sort_return_swapped: }
2055   }
2056 {
2057   % Neither is empty: we can compare the values of the
2058   % current enclosing counter in the loop, if they are
2059   % equal, we are still in the loop, if they are not, a
2060   % sorting decision can be made directly.
2061   \int_compare:nNnTF
2062     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2063     =
2064     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2065     {
2066       \tl_set:Nx \l__zrefclever_label_enclval_a_tl
2067         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
2068       \tl_set:Nx \l__zrefclever_label_enclval_b_tl
2069         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
2070     }
2071     {
2072       \bool_set_true:N \l__zrefclever_sort_decided_bool
2073       \int_compare:nNnTF
2074         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
2075         >
2076         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
2077         { \sort_return_swapped: }
2078         { \sort_return_same: }
2079     }
2080   }
2081 }
2082 }
2083 }
2084 }

```

(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}}
2085 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
2086 {

```

Retrieve sort priorities for $\langle label\ a \rangle$ and $\langle label\ b \rangle$. $\backslash l_zrefclever_typesort_seq$ 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”.

```

2087 \int_zero:N \l__zrefclever_sort_prior_a_int
2088 \int_zero:N \l__zrefclever_sort_prior_b_int
2089 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq

```

```

2090 {
2091   \tl_if_eq:nnTF {##2} {{othertypes}}
2092   {
2093     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
2094     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
2095     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
2096     { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
2097   }
2098   {
2099     \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
2100     { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
2101     {
2102       \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
2103       { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
2104     }
2105   }
2106 }

```

Then do the actual sorting.

```

2107 \bool_if:nTF
2108 {
2109   \int_compare_p:nNn
2110   { \l__zrefclever_sort_prior_a_int } <
2111   { \l__zrefclever_sort_prior_b_int }
2112 }
2113 { \sort_return_same: }
2114 {
2115   \bool_if:nTF
2116   {
2117     \int_compare_p:nNn
2118     { \l__zrefclever_sort_prior_a_int } >
2119     { \l__zrefclever_sort_prior_b_int }
2120   }
2121   { \sort_return_swapped: }
2122   {
2123     % Sort priorities are equal: the type that occurs first in
2124     % 'labels', as given by the user, is kept (or brought) forward.
2125     \seq_map_inline:Nn \l__zrefclever_label_types_seq
2126     {
2127       \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
2128       { \seq_map_break:n { \sort_return_same: } }
2129       {
2130         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
2131         { \seq_map_break:n { \sort_return_swapped: } }
2132       }
2133     }
2134   }
2135 }
2136 }

```

(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}\}

2137 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
2138 {
2139   \int_compare:nNnTF
2140     { \__zrefclever_extract:nnn {#1} { abspage } { -1 } }
2141     >
2142     { \__zrefclever_extract:nnn {#2} { abspage } { -1 } }
2143     { \sort_return_swapped: }
2144     { \sort_return_same: }
2145 }

```

(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 an 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>	2146 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l__zrefclever_last_of_type_bool</code>	2147 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	2148 <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>	2149 <code>\int_new:N \l__zrefclever_type_count_int</code>
	2150 <code>\int_new:N \l__zrefclever_label_count_int</code>

(End definition for `\l__zrefclever_type_count_int` and `\l__zrefclever_label_count_int`.)

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

`\l__zrefclever_typeset_queue_prev_tl` 2151 `\tl_new:N \l__zrefclever_label_a_tl`
`\l__zrefclever_typeset_queue_curr_tl` 2152 `\tl_new:N \l__zrefclever_label_b_tl`
`\l__zrefclever_type_first_label_tl` 2153 `\tl_new:N \l__zrefclever_typeset_queue_prev_tl`
`\l__zrefclever_type_first_label_type_tl` 2154 `\tl_new:N \l__zrefclever_typeset_queue_curr_tl`
2155 `\tl_new:N \l__zrefclever_type_first_label_tl`
2156 `\tl_new:N \l__zrefclever_type_first_label_type_tl`

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

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

`\l__zrefclever_name_in_link_bool` 2157 `\tl_new:N \l__zrefclever_type_name_tl`
`\l__zrefclever_name_format_tl` 2158 `\bool_new:N \l__zrefclever_name_in_link_bool`
`\l__zrefclever_name_format_fallback_tl` 2159 `\tl_new:N \l__zrefclever_name_format_tl`
`\l__zrefclever_type_name_gender_tl` 2160 `\tl_new:N \l__zrefclever_name_format_fallback_tl`
2161 `\tl_new:N \l__zrefclever_type_name_gender_tl`

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

`\l__zrefclever_range_count_int` Auxiliary variables for `__zrefclever_typeset_refs`: range handling.

`\l__zrefclever_range_same_count_int` 2162 `\int_new:N \l__zrefclever_range_count_int`
`\l__zrefclever_range_beg_label_tl` 2163 `\int_new:N \l__zrefclever_range_same_count_int`
`\l__zrefclever_next_maybe_range_bool` 2164 `\tl_new:N \l__zrefclever_range_beg_label_tl`
`\l__zrefclever_next_is_same_bool` 2165 `\bool_new:N \l__zrefclever_next_maybe_range_bool`
2166 `\bool_new:N \l__zrefclever_next_is_same_bool`

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

`\l__zrefclever_tpairsep_tl` Auxiliary variables for `__zrefclever_typeset_refs`: separators, pre-/postref and font and other options.
`\l__zrefclever_tlistsep_tl`

`\l__zrefclever_tlastsep_tl` 2167 `\tl_new:N \l__zrefclever_tpairsep_tl`
`\l__zrefclever_namesep_tl` 2168 `\tl_new:N \l__zrefclever_tlistsep_tl`
`\l__zrefclever_pairsep_tl` 2169 `\tl_new:N \l__zrefclever_tlastsep_tl`
`\l__zrefclever_listsep_tl` 2170 `\tl_new:N \l__zrefclever_namesep_tl`
`\l__zrefclever_lastsep_tl` 2171 `\tl_new:N \l__zrefclever_pairsep_tl`
`\l__zrefclever_rangesep_tl` 2172 `\tl_new:N \l__zrefclever_listsep_tl`
`\l__zrefclever_preref_tl` 2173 `\tl_new:N \l__zrefclever_lastsep_tl`
`\l__zrefclever_postref_tl` 2174 `\tl_new:N \l__zrefclever_rangesep_tl`
`\l__zrefclever_namefont_tl` 2175 `\tl_new:N \l__zrefclever_preref_tl`
`\l__zrefclever_reffont_tl` 2176 `\tl_new:N \l__zrefclever_postref_tl`
`\l__zrefclever_capitalize_bool` 2177 `\tl_new:N \l__zrefclever_namefont_tl`
`\l__zrefclever_abbrev_bool` 2178 `\tl_new:N \l__zrefclever_reffont_tl`
2179 `\bool_new:N \l__zrefclever_capitalize_bool`
2180 `\bool_new:N \l__zrefclever_abbrev_bool`

(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`.

2181 `\bool_new:N \l__zrefclever_verbose_testing_bool`

(End definition for `\l__zrefclever_verbose_testing_bool`.)

Main functions

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

```

2182 \cs_new_protected:Npn \_zrefclever_typeset_refs:
2183 {
2184   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
2185     \l__zrefclever_zceref_labels_seq
2186   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
2187   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2188   \tl_clear:N \l__zrefclever_type_first_label_tl
2189   \tl_clear:N \l__zrefclever_type_first_label_type_tl
2190   \tl_clear:N \l__zrefclever_range_beg_label_tl
2191   \int_zero:N \l__zrefclever_label_count_int
2192   \int_zero:N \l__zrefclever_type_count_int
2193   \int_zero:N \l__zrefclever_range_count_int
2194   \int_zero:N \l__zrefclever_range_same_count_int
2195
2196   % Get type block options (not type-specific).
2197   \_zrefclever_get_ref_opt_typeset:nN { tpairsep }
2198     \l__zrefclever_tpairsep_tl
2199   \_zrefclever_get_ref_opt_typeset:nN { tlistsep }
2200     \l__zrefclever_tlistsep_tl
2201   \_zrefclever_get_ref_opt_typeset:nN { tlastsep }
2202     \l__zrefclever_tlastsep_tl
2203
2204   % Process label stack.
2205   \bool_set_false:N \l__zrefclever_typeset_last_bool
2206   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
2207     {
2208       \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
2209         \l__zrefclever_label_a_tl
2210       \seq_if_empty:NNTF \l__zrefclever_typeset_labels_seq
2211         {
2212           \tl_clear:N \l__zrefclever_label_b_tl
2213           \bool_set_true:N \l__zrefclever_typeset_last_bool
2214         }
2215         {
2216           \seq_get_left:NN \l__zrefclever_typeset_labels_seq
2217             \l__zrefclever_label_b_tl
2218         }
2219
2220       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2221         {
2222           \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
2223           \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
2224         }
2225         {
2226           \_zrefclever_extract_default:NVnn
2227             \l__zrefclever_label_type_a_tl
2228             \l__zrefclever_label_a_tl { zc@type } { zc@missingtype }
2229           \_zrefclever_extract_default:NVnn
2230             \l__zrefclever_label_type_b_tl
2231             \l__zrefclever_label_b_tl { zc@type } { zc@missingtype }
2232         }

```

```

2233
2234 % First, we establish whether the "current label" (i.e. 'a') is the
2235 % last one of its type. This can happen because the "next label"
2236 % (i.e. 'b') is of a different type (or different definition status),
2237 % or because we are at the end of the list.
2238 \bool_if:NTF \l__zrefclever_typeset_last_bool
2239 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2240 {
2241   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2242   {
2243     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2244     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2245     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2246   }
2247   {
2248     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
2249     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2250     {
2251       % Neither is undefined, we must check the types.
2252       \tl_if_eq:NNTF
2253         \l__zrefclever_label_type_a_tl
2254         \l__zrefclever_label_type_b_tl
2255         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
2256         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
2257     }
2258   }
2259 }
2260
2261 % Handle warnings in case of reference or type undefined.
2262 % Test: 'zc-typeset01.lvt': "Typeset refs: warn ref undefined"
2263 \zref@refused { \l__zrefclever_label_a_tl }
2264 % Test: 'zc-typeset01.lvt': "Typeset refs: warn missing type"
2265 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2266 {}
2267 {
2268   \tl_if_eq:NnT \l__zrefclever_label_type_a_tl { zc@missingtype }
2269   {
2270     \msg_warning:nxx { zref-clever } { missing-type }
2271     { \l__zrefclever_label_a_tl }
2272   }
2273   \zref@ifrefcontainsprop
2274     { \l__zrefclever_label_a_tl }
2275     { \l__zrefclever_ref_property_tl }
2276     { }
2277     {
2278       \msg_warning:nxxx { zref-clever } { missing-property }
2279       { \l__zrefclever_ref_property_tl }
2280       { \l__zrefclever_label_a_tl }
2281     }
2282 }
2283
2284 % Get type-specific separators, pre-/postref font and other options,
2285 % once per type.
2286 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }

```

```

2287 {
2288   \__zrefclever_get_ref_opt_typeset:nN { namesep }
2289   \l__zrefclever_namesep_tl
2290   \__zrefclever_get_ref_opt_typeset:nN { pairsep }
2291   \l__zrefclever_pairsep_tl
2292   \__zrefclever_get_ref_opt_typeset:nN { listsep }
2293   \l__zrefclever_listsep_tl
2294   \__zrefclever_get_ref_opt_typeset:nN { lastsep }
2295   \l__zrefclever_lastsep_tl
2296   \__zrefclever_get_ref_opt_typeset:nN { rangesep }
2297   \l__zrefclever_rangesep_tl
2298   \__zrefclever_get_ref_opt_typeset:nN { preref }
2299   \l__zrefclever_preref_tl
2300   \__zrefclever_get_ref_opt_typeset:nN { postref }
2301   \l__zrefclever_postref_tl
2302   \__zrefclever_get_ref_opt_font:nN { namefont }
2303   \l__zrefclever_namefont_tl
2304   \__zrefclever_get_ref_opt_font:nN { reffont }
2305   \l__zrefclever_reffont_tl
2306   \__zrefclever_get_ref_opt_bool:nnN { cap } { false }
2307   \l__zrefclever_capitalize_bool
2308   \__zrefclever_get_ref_opt_bool:nnN { abbrev } { false }
2309   \l__zrefclever_abbrev_bool
2310 }
2311
2312 % Here we send this to a couple of auxiliary functions.
2313 \bool_if:NTF \l__zrefclever_last_of_type_bool
2314 % There exists no next label of the same type as the current.
2315 { \__zrefclever_typeset_refs_last_of_type: }
2316 % There exists a next label of the same type as the current.
2317 { \__zrefclever_typeset_refs_not_last_of_type: }
2318 }
2319 }

```

(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.

```

2320 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
2321 {
2322   % Process the current label to the current queue.
2323   \int_case:nnF { \l__zrefclever_label_count_int }
2324   {
2325     % It is the last label of its type, but also the first one, and that's
2326     % what matters here: just store it.
2327     % Test: 'zc-typeset01.lvt': "Last of type: single"

```



```

2328 { 0 }
2329 {
2330   \tl_set:NV \l__zrefclever_type_first_label_tl
2331   \l__zrefclever_label_a_tl
2332   \tl_set:NV \l__zrefclever_type_first_label_type_tl
2333   \l__zrefclever_label_type_a_tl
2334 }
2335
2336 % The last is the second: we have a pair (if not repeated).
2337 % Test: 'zc-typeset01.lvt': "Last of type: pair"
2338 { 1 }
2339 {
2340   \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
2341   {
2342     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2343     {
2344       \exp_not:V \l__zrefclever_pairsep_tl
2345       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2346     }
2347   }
2348 }
2349 }
2350 % Last is third or more of its type: without repetition, we'd have the
2351 % last element on a list, but control for possible repetition.
2352 {
2353   \int_case:nnF { \l__zrefclever_range_count_int }
2354   {
2355     % There was no range going on.
2356     % Test: 'zc-typeset01.lvt': "Last of type: not range"
2357     { 0 }
2358     {
2359       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2360       {
2361         \exp_not:V \l__zrefclever_lastsep_tl
2362         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2363       }
2364     }
2365     % Last in the range is also the second in it.
2366     % Test: 'zc-typeset01.lvt': "Last of type: pair in sequence"
2367     { 1 }
2368     {
2369       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2370       {
2371         % We know 'range_beg_label' is not empty, since this is the
2372         % second element in the range, but the third or more in the
2373         % type list.
2374         \exp_not:V \l__zrefclever_listsep_tl
2375         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
2376         \int_compare:nNnF
2377         { \l__zrefclever_range_same_count_int } = { 1 }
2378         {
2379           \exp_not:V \l__zrefclever_lastsep_tl
2380           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2381         }
2382       }
2383     }
2384   }

```

```

2382     }
2383   }
2384 }
2385 % Last in the range is third or more in it.
2386 {
2387   \int_case:nnF
2388   {
2389     \l__zrefclever_range_count_int -
2390     \l__zrefclever_range_same_count_int
2391   }
2392   {
2393     % Repetition, not a range.
2394     % Test: 'zc-typeset01.lvt': "Last of type: range to one"
2395     { 0 }
2396     {
2397       % If 'range_beg_label' is empty, it means it was also the
2398       % first of the type, and hence was already handled.
2399       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2400       {
2401         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2402         {
2403           \exp_not:V \l__zrefclever_lastsep_tl
2404           \__zrefclever_get_ref:V
2405             \l__zrefclever_range_beg_label_tl
2406         }
2407       }
2408     }
2409     % A 'range', but with no skipped value, treat as list.
2410     % Test: 'zc-typeset01.lvt': "Last of type: range to pair"
2411     { 1 }
2412     {
2413       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2414       {
2415         % Ditto.
2416         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2417         {
2418           \exp_not:V \l__zrefclever_listsep_tl
2419           \__zrefclever_get_ref:V
2420             \l__zrefclever_range_beg_label_tl
2421         }
2422         \exp_not:V \l__zrefclever_lastsep_tl
2423         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2424       }
2425     }
2426   }
2427   {
2428     % An actual range.
2429     % Test: 'zc-typeset01.lvt': "Last of type: range"
2430     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2431     {
2432       % Ditto.
2433       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2434       {
2435         \exp_not:V \l__zrefclever_lastsep_tl

```

```

2436         \_zrefclever_get_ref:V
2437         \l__zrefclever_range_beg_label_tl
2438     }
2439     \exp_not:V \l__zrefclever_rangesep_tl
2440     \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
2441 }
2442 }
2443 }
2444 }
2445
2446 % Handle "range" option. The idea is simple: if the queue is not empty,
2447 % we replace it with the end of the range (or pair). We can still
2448 % retrieve the end of the range from 'label_a' since we know to be
2449 % processing the last label of its type at this point.
2450 \bool_if:NT \l__zrefclever_typeset_range_bool
2451 {
2452     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2453     {
2454         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2455         { }
2456         {
2457             \msg_warning:nxx { zref-clever } { single-element-range }
2458             { \l__zrefclever_type_first_label_type_tl }
2459         }
2460     }
2461     {
2462         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2463         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2464         { }
2465         {
2466             \_zrefclever_labels_in_sequence:nn
2467             { \l__zrefclever_type_first_label_tl }
2468             { \l__zrefclever_label_a_tl }
2469         }
2470         % Test: 'zc-typeset01.lvt': "Last of type: option range"
2471         % Test: 'zc-typeset01.lvt': "Last of type: option range to pair"
2472         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2473         {
2474             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
2475             { \exp_not:V \l__zrefclever_pairsep_tl }
2476             { \exp_not:V \l__zrefclever_rangesep_tl }
2477             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
2478         }
2479     }
2480 }
2481
2482 % Now that the type block is finished, we can add the name and the first
2483 % ref to the queue. Also, if "typeset" option is not "both", handle it
2484 % here as well.
2485 \_zrefclever_type_name_setup:
2486 \bool_if:nTF
2487 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
2488 {
2489     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl

```

```

2490     { \_zrefclever_get_ref_first: }
2491   }
2492   {
2493     \bool_if:NTF \l__zrefclever_typeset_ref_bool
2494     {
2495       % Test: 'zc-typeset01.lvt': "Last of type: option typeset ref"
2496       \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2497       { \_zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
2498     }
2499     {
2500       \bool_if:NTF \l__zrefclever_typeset_name_bool
2501       {
2502         % Test: 'zc-typeset01.lvt': "Last of type: option typeset name"
2503         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
2504         {
2505           \bool_if:NTF \l__zrefclever_name_in_link_bool
2506           {
2507             \exp_not:N \group_begin:
2508             \exp_not:V \l__zrefclever_namefont_tl
2509             % It's two '@s', but escaped for DocStrip.
2510             \exp_not:N \hyper@@link
2511             {
2512               \_zrefclever_extract_url_unexp:V
2513               \l__zrefclever_type_first_label_tl
2514             }
2515             {
2516               \_zrefclever_extract_unexp:Vnn
2517               \l__zrefclever_type_first_label_tl
2518               { anchor } { }
2519             }
2520             { \exp_not:V \l__zrefclever_type_name_tl }
2521             \exp_not:N \group_end:
2522           }
2523           {
2524             \exp_not:N \group_begin:
2525             \exp_not:V \l__zrefclever_namefont_tl
2526             \exp_not:V \l__zrefclever_type_name_tl
2527             \exp_not:N \group_end:
2528           }
2529         }
2530       }
2531     }
2532     % Logically, this case would correspond to "typeset=none", but
2533     % it should not occur, given that the options are set up to
2534     % typeset either "ref" or "name". Still, leave here a
2535     % sensible fallback, equal to the behavior of "both".
2536     % Test: 'zc-typeset01.lvt': "Last of type: option typeset none"
2537     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
2538     { \_zrefclever_get_ref_first: }
2539   }
2540 }
2541 }
2542
2543 % Typeset the previous type block, if there is one.

```

```

2544 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
2545 {
2546     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
2547     { \l__zrefclever_tlistsep_tl }
2548     \l__zrefclever_typeset_queue_prev_tl
2549 }
2550
2551 % Extra log for testing.
2552 \bool_if:NT \l__zrefclever_verbose_testing_bool
2553 { \tl_show:N \l__zrefclever_typeset_queue_curr_tl }
2554
2555 % Wrap up loop, or prepare for next iteration.
2556 \bool_if:NTF \l__zrefclever_typeset_last_bool
2557 {
2558     % We are finishing, typeset the current queue.
2559     \int_case:nnF { \l__zrefclever_type_count_int }
2560     {
2561         % Single type.
2562         % Test: 'zc-typeset01.lvt': "Last of type: single type"
2563         { 0 }
2564         { \l__zrefclever_typeset_queue_curr_tl }
2565         % Pair of types.
2566         % Test: 'zc-typeset01.lvt': "Last of type: pair of types"
2567         { 1 }
2568         {
2569             \l__zrefclever_tpairsep_tl
2570             \l__zrefclever_typeset_queue_curr_tl
2571         }
2572     }
2573     {
2574         % Last in list of types.
2575         % Test: 'zc-typeset01.lvt': "Last of type: list of types"
2576         \l__zrefclever_tlastsep_tl
2577         \l__zrefclever_typeset_queue_curr_tl
2578     }
2579     % And nudge in case of multitype reference.
2580     \bool_lazy_all:nT
2581     {
2582         { \l__zrefclever_nudge_enabled_bool }
2583         { \l__zrefclever_nudge_multitype_bool }
2584         { \int_compare_p:nNn { \l__zrefclever_type_count_int } > { 0 } }
2585     }
2586     { \msg_warning:nn { zref-clever } { nudge-multitype } }
2587 }
2588 {
2589     % There are further labels, set variables for next iteration.
2590     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
2591     \l__zrefclever_typeset_queue_curr_tl
2592     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
2593     \tl_clear:N \l__zrefclever_type_first_label_tl
2594     \tl_clear:N \l__zrefclever_type_first_label_type_tl
2595     \tl_clear:N \l__zrefclever_range_beg_label_tl
2596     \int_zero:N \l__zrefclever_label_count_int
2597     \int_incr:N \l__zrefclever_type_count_int

```

```

2598         \int_zero:N \l__zrefclever_range_count_int
2599         \int_zero:N \l__zrefclever_range_same_count_int
2600     }
2601 }

```

(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.

```

2602 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
2603 {
2604     % Signal if next label may form a range with the current one (only
2605     % considered if compression is enabled in the first place).
2606     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
2607     \bool_set_false:N \l__zrefclever_next_is_same_bool
2608     \bool_if:NT \l__zrefclever_typeset_compress_bool
2609     {
2610         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
2611         { }
2612         {
2613             \__zrefclever_labels_in_sequence:nn
2614             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
2615         }
2616     }
2617
2618     % Process the current label to the current queue.
2619     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
2620     {
2621         % Current label is the first of its type (also not the last, but it
2622         % doesn't matter here): just store the label.
2623         \tl_set:NV \l__zrefclever_type_first_label_tl
2624             \l__zrefclever_label_a_tl
2625         \tl_set:NV \l__zrefclever_type_first_label_type_tl
2626             \l__zrefclever_label_type_a_tl
2627
2628         % If the next label may be part of a range, we set 'range_beg_label'
2629         % to "empty" (we deal with it as the "first", and must do it there, to
2630         % handle hyperlinking), but also step the range counters.
2631         % Test: 'zc-typeset01.lvt': "Not last of type: first is range"
2632         \bool_if:NT \l__zrefclever_next_maybe_range_bool
2633         {
2634             \tl_clear:N \l__zrefclever_range_beg_label_tl
2635             \int_incr:N \l__zrefclever_range_count_int
2636             \bool_if:NT \l__zrefclever_next_is_same_bool
2637             { \int_incr:N \l__zrefclever_range_same_count_int }
2638         }
2639     }
2640     {
2641         % Current label is neither the first (nor the last) of its type.
2642         \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
2643         {
2644             % Starting, or continuing a range.
2645             \int_compare:nNnTF
2646             { \l__zrefclever_range_count_int } = { 0 }
2647             {

```

```

2648 % There was no range going, we are starting one.
2649 \tl_set:NV \l__zrefclever_range_beg_label_tl
2650 \l__zrefclever_label_a_tl
2651 \int_incr:N \l__zrefclever_range_count_int
2652 \bool_if:NT \l__zrefclever_next_is_same_bool
2653 { \int_incr:N \l__zrefclever_range_same_count_int }
2654 }
2655 {
2656 % Second or more in the range, but not the last.
2657 \int_incr:N \l__zrefclever_range_count_int
2658 \bool_if:NT \l__zrefclever_next_is_same_bool
2659 { \int_incr:N \l__zrefclever_range_same_count_int }
2660 }
2661 }
2662 {
2663 % Next element is not in sequence: there was no range, or we are
2664 % closing one.
2665 \int_case:nnF { \l__zrefclever_range_count_int }
2666 {
2667 % There was no range going on.
2668 % Test: 'zc-typeset01.lvt': "Not last of type: no range"
2669 { 0 }
2670 {
2671 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2672 {
2673 \exp_not:V \l__zrefclever_listsep_tl
2674 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2675 }
2676 }
2677 % Last is second in the range: if 'range_same_count' is also
2678 % '1', it's a repetition (drop it), otherwise, it's a "pair
2679 % within a list", treat as list.
2680 % Test: 'zc-typeset01.lvt': "Not last of type: range pair to one"
2681 % Test: 'zc-typeset01.lvt': "Not last of type: range pair"
2682 { 1 }
2683 {
2684 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2685 {
2686 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2687 {
2688 \exp_not:V \l__zrefclever_listsep_tl
2689 \__zrefclever_get_ref:V
2690 \l__zrefclever_range_beg_label_tl
2691 }
2692 \int_compare:nNnF
2693 { \l__zrefclever_range_same_count_int } = { 1 }
2694 {
2695 \exp_not:V \l__zrefclever_listsep_tl
2696 \__zrefclever_get_ref:V
2697 \l__zrefclever_label_a_tl
2698 }
2699 }
2700 }
2701 }

```

```

2702 {
2703   % Last is third or more in the range: if 'range_count' and
2704   % 'range_same_count' are the same, its a repetition (drop it),
2705   % if they differ by '1', its a list, if they differ by more,
2706   % it is a real range.
2707   \int_case:nnF
2708   {
2709     \l__zrefclever_range_count_int -
2710     \l__zrefclever_range_same_count_int
2711   }
2712   {
2713     % Test: 'zc-typeset01.lvt': "Not last of type: range to one"
2714     { 0 }
2715     {
2716       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2717       {
2718         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2719         {
2720           \exp_not:V \l__zrefclever_listsep_tl
2721           \__zrefclever_get_ref:V
2722             \l__zrefclever_range_beg_label_tl
2723         }
2724       }
2725     }
2726     % Test: 'zc-typeset01.lvt': "Not last of type: range to pair"
2727     { 1 }
2728     {
2729       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2730       {
2731         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2732         {
2733           \exp_not:V \l__zrefclever_listsep_tl
2734           \__zrefclever_get_ref:V
2735             \l__zrefclever_range_beg_label_tl
2736         }
2737         \exp_not:V \l__zrefclever_listsep_tl
2738         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2739       }
2740     }
2741   }
2742   {
2743     % Test: 'zc-typeset01.lvt': "Not last of type: range"
2744     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2745     {
2746       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2747       {
2748         \exp_not:V \l__zrefclever_listsep_tl
2749         \__zrefclever_get_ref:V
2750           \l__zrefclever_range_beg_label_tl
2751       }
2752       \exp_not:V \l__zrefclever_rangeseq_tl
2753       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2754     }
2755   }

```



```

2756         }
2757         % Reset counters.
2758         \int_zero:N \l__zrefclever_range_count_int
2759         \int_zero:N \l__zrefclever_range_same_count_int
2760     }
2761 }
2762 % Step label counter for next iteration.
2763 \int_incr:N \l__zrefclever_label_count_int
2764 }

```

(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 nicl requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, `__zrefclever_get_ref: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.

```

2765 \cs_new_protected:Npn \__zrefclever_ref_default:
2766 { \zref@default }
2767 \cs_new_protected:Npn \__zrefclever_name_default:
2768 { \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}

```

```

2769 \cs_new:Npn \__zrefclever_get_ref:n #1
2770 {
2771   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2772   {
2773     \bool_if:nTF
2774     {
2775       \l__zrefclever_use_hyperref_bool &&
2776       ! \l__zrefclever_link_star_bool
2777     }
2778     {
2779       \bool_if:NF \l__zrefclever_preposinlink_bool
2780       { \exp_not:V \l__zrefclever_preref_tl }
2781       % It's two '@s', but escaped for DocStrip.
2782       \exp_not:N \hyper@@link
2783       { \__zrefclever_extract_url_unexp:n {#1} }
2784       { \__zrefclever_extract_unexp:nnn {#1} { anchor } { } }
2785       {
2786         \bool_if:NT \l__zrefclever_preposinlink_bool
2787         { \exp_not:V \l__zrefclever_preref_tl }
2788         \exp_not:N \group_begin:
2789         \exp_not:V \l__zrefclever_reffont_tl
2790         \__zrefclever_extract_unexp:nvn {#1}
2791         { \l__zrefclever_ref_property_tl } { }
2792         \exp_not:N \group_end:
2793         \bool_if:NT \l__zrefclever_preposinlink_bool
2794         { \exp_not:V \l__zrefclever_postref_tl }
2795       }
2796       \bool_if:NF \l__zrefclever_preposinlink_bool
2797       { \exp_not:V \l__zrefclever_postref_tl }
2798     }
2799     {
2800       \exp_not:V \l__zrefclever_preref_tl
2801       \exp_not:N \group_begin:
2802       \exp_not:V \l__zrefclever_reffont_tl
2803       \__zrefclever_extract_unexp:nvn {#1}
2804       { \l__zrefclever_ref_property_tl } { }
2805       \exp_not:N \group_end:
2806       \exp_not:V \l__zrefclever_postref_tl
2807     }
2808   }
2809   { \__zrefclever_ref_default: }
2810 }
2811 \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.

```

2812 \cs_new:Npn \__zrefclever_get_ref_first:
2813 {
2814   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2815   { \__zrefclever_ref_default: }
2816   {
2817     \bool_if:NTF \l__zrefclever_name_in_link_bool
2818     {
2819       \zref@ifrefcontainsprop
2820       { \l__zrefclever_type_first_label_tl }
2821       { \l__zrefclever_ref_property_tl }
2822       {
2823         % It's two '@s', but escaped for DocStrip.
2824         \exp_not:N \hyper@@link
2825         {
2826           \__zrefclever_extract_url_unexp:V
2827           \l__zrefclever_type_first_label_tl
2828         }
2829         {
2830           \__zrefclever_extract_unexp:Vnn
2831           \l__zrefclever_type_first_label_tl { anchor } { }
2832         }
2833         {
2834           \exp_not:N \group_begin:
2835           \exp_not:V \l__zrefclever_namefont_tl
2836           \exp_not:V \l__zrefclever_type_name_tl
2837           \exp_not:N \group_end:
2838           \exp_not:V \l__zrefclever_namesep_tl
2839           \exp_not:V \l__zrefclever_preref_tl
2840           \exp_not:N \group_begin:
2841           \exp_not:V \l__zrefclever_reffont_tl
2842           \__zrefclever_extract_unexp:Vnn
2843           \l__zrefclever_type_first_label_tl
2844           { \l__zrefclever_ref_property_tl } { }
2845           \exp_not:N \group_end:
2846           \bool_if:NT \l__zrefclever_preposinlink_bool
2847           { \exp_not:V \l__zrefclever_postref_tl }
2848         }
2849         \bool_if:NF \l__zrefclever_preposinlink_bool
2850         { \exp_not:V \l__zrefclever_postref_tl }
2851       }
2852     }
2853     {
2854       \exp_not:N \group_begin:
2855       \exp_not:V \l__zrefclever_namefont_tl
2856       \exp_not:V \l__zrefclever_type_name_tl
2857       \exp_not:N \group_end:
2858       \exp_not:V \l__zrefclever_namesep_tl
2859       \__zrefclever_ref_default:
2860     }
2861   }
2862   {
2863     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2864     {
2865       \__zrefclever_name_default:
2866       \exp_not:V \l__zrefclever_namesep_tl

```

```

2866     }
2867     {
2868         \exp_not:N \group_begin:
2869         \exp_not:V \l__zrefclever_namefont_tl
2870         \exp_not:V \l__zrefclever_type_name_tl
2871         \exp_not:N \group_end:
2872         \exp_not:V \l__zrefclever_namesep_tl
2873     }
2874 \zref@ifrefcontainsprop
2875 { \l__zrefclever_type_first_label_tl }
2876 { \l__zrefclever_ref_property_tl }
2877 {
2878     \bool_if:nTF
2879     {
2880         \l__zrefclever_use_hyperref_bool &&
2881         ! \l__zrefclever_link_star_bool
2882     }
2883     {
2884         \bool_if:NF \l__zrefclever_preposinlink_bool
2885         { \exp_not:V \l__zrefclever_preref_tl }
2886         % It's two '@s', but escaped for DocStrip.
2887         \exp_not:N \hyper@@link
2888         {
2889             \__zrefclever_extract_url_unexp:V
2890             \l__zrefclever_type_first_label_tl
2891         }
2892         {
2893             \__zrefclever_extract_unexp:Vnn
2894             \l__zrefclever_type_first_label_tl { anchor } { }
2895         }
2896         {
2897             \bool_if:NT \l__zrefclever_preposinlink_bool
2898             { \exp_not:V \l__zrefclever_preref_tl }
2899             \exp_not:N \group_begin:
2900             \exp_not:V \l__zrefclever_reffont_tl
2901             \__zrefclever_extract_unexp:Vnn
2902             \l__zrefclever_type_first_label_tl
2903             { \l__zrefclever_ref_property_tl } { }
2904             \exp_not:N \group_end:
2905             \bool_if:NT \l__zrefclever_preposinlink_bool
2906             { \exp_not:V \l__zrefclever_postref_tl }
2907         }
2908         \bool_if:NF \l__zrefclever_preposinlink_bool
2909         { \exp_not:V \l__zrefclever_postref_tl }
2910     }
2911     {
2912         \exp_not:V \l__zrefclever_preref_tl
2913         \exp_not:N \group_begin:
2914         \exp_not:V \l__zrefclever_reffont_tl
2915         \__zrefclever_extract_unexp:Vnn
2916         \l__zrefclever_type_first_label_tl
2917         { \l__zrefclever_ref_property_tl } { }
2918         \exp_not:N \group_end:
2919         \exp_not:V \l__zrefclever_postref_tl

```

```

2920     }
2921   }
2922   { \_zrefclever_ref_default: }
2923 }
2924 }
2925 }

```

(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.

```

2926 \cs_new_protected:Npn \_zrefclever_type_name_setup:
2927 {
2928   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
2929   { \tl_clear:N \l\_zrefclever_type_name_tl }
2930   {
2931     \tl_if_eq:NnTF
2932     \l\_zrefclever_type_first_label_type_tl { zc@missingtype }
2933     { \tl_clear:N \l\_zrefclever_type_name_tl }
2934     {
2935       % Determine whether we should use capitalization, abbreviation,
2936       % and plural.
2937       \bool_lazy_or:nnTF
2938       { \l\_zrefclever_capitalize_bool }
2939       {
2940         \l\_zrefclever_capitalize_first_bool &&
2941         \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 }
2942       }
2943       { \tl_set:Nn \l\_zrefclever_name_format_tl {Name} }
2944       { \tl_set:Nn \l\_zrefclever_name_format_tl {name} }
2945       % If the queue is empty, we have a singular, otherwise, plural.
2946       \tl_if_empty:NTF \l\_zrefclever_typeset_queue_curr_tl
2947       { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -sg } }
2948       { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -pl } }
2949       \bool_lazy_and:nnTF
2950       { \l\_zrefclever_abbrev_bool }
2951       {
2952         ! \int_compare_p:nNn
2953         { \l\_zrefclever_type_count_int } = { 0 } ||
2954         ! \l\_zrefclever_noabbrev_first_bool
2955       }
2956       {
2957         \tl_set:NV \l\_zrefclever_name_format_fallback_tl
2958         \l\_zrefclever_name_format_tl

```

```

2959     \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2960   }
2961   { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2962
2963   % Handle number and gender nudges.
2964   \bool_if:NT \l__zrefclever_nudge_enabled_bool
2965   {
2966     \bool_if:NTF \l__zrefclever_nudge_singular_bool
2967     {
2968       \tl_if_empty:NF \l__zrefclever_typeset_queue_curr_tl
2969       {
2970         \msg_warning:nnx { zref-clever }
2971         { nudge-plural-when-sg }
2972         { \l__zrefclever_type_first_label_type_tl }
2973       }
2974     }
2975     {
2976       \bool_lazy_all:nT
2977       {
2978         { \l__zrefclever_nudge_comptosing_bool }
2979         { \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl }
2980         {
2981           \int_compare_p:nNn
2982             { \l__zrefclever_label_count_int } > { 0 }
2983         }
2984       }
2985       {
2986         \msg_warning:nnx { zref-clever }
2987         { nudge-comptosing }
2988         { \l__zrefclever_type_first_label_type_tl }
2989       }
2990     }
2991   \bool_lazy_and:nnT
2992   { \l__zrefclever_nudge_gender_bool }
2993   { ! \tl_if_empty_p:N \l__zrefclever_ref_gender_tl }
2994   {
2995     \__zrefclever_get_lang_opt_type:xxnNF
2996     { \l__zrefclever_ref_language_tl }
2997     { \l__zrefclever_type_first_label_type_tl }
2998     { gender }
2999     \l__zrefclever_type_name_gender_tl
3000     { \tl_clear:N \l__zrefclever_type_name_gender_tl }
3001   \clist_set:NV \l_tmpa_clist
3002     \l__zrefclever_type_name_gender_tl
3003   \clist_if_in:NVF
3004     \l_tmpa_clist
3005     \l__zrefclever_ref_gender_tl
3006     {
3007       \tl_if_empty:NTF \l__zrefclever_type_name_gender_tl
3008       {
3009         \msg_warning:nxxxx { zref-clever }
3010         { nudge-gender-not-declared-for-type }
3011         { \l__zrefclever_ref_gender_tl }
3012         { \l__zrefclever_type_first_label_type_tl }

```

```

3013         { \l__zrefclever_ref_language_tl }
3014     }
3015     {
3016         \msg_warning:nxxxxx { zref-clever }
3017         { nudge-gender-mismatch }
3018         { \l__zrefclever_type_first_label_type_tl }
3019         { \l__zrefclever_ref_gender_tl }
3020         { \l__zrefclever_type_name_gender_tl }
3021         { \l__zrefclever_ref_language_tl }
3022     }
3023 }
3024 }
3025 }
3026
3027 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
3028 {
3029     \prop_get:cVNF
3030     {
3031         l__zrefclever_type_
3032         \l__zrefclever_type_first_label_type_tl _options_prop
3033     }
3034     \l__zrefclever_name_format_tl
3035     \l__zrefclever_type_name_tl
3036     {
3037         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
3038         {
3039             \tl_put_left:Nn \l__zrefclever_name_format_tl { - }
3040             \tl_put_left:NV \l__zrefclever_name_format_tl
3041             \l__zrefclever_ref_decl_case_tl
3042         }
3043         \__zrefclever_get_lang_opt_type:xxxNF
3044         { \l__zrefclever_ref_language_tl }
3045         { \l__zrefclever_type_first_label_type_tl }
3046         { \l__zrefclever_name_format_tl }
3047         \l__zrefclever_type_name_tl
3048         {
3049             \tl_clear:N \l__zrefclever_type_name_tl
3050             \msg_warning:nxxx { zref-clever } { missing-name }
3051             { \l__zrefclever_name_format_tl }
3052             { \l__zrefclever_type_first_label_type_tl }
3053         }
3054     }
3055 }
3056 {
3057     \prop_get:cVNF
3058     {
3059         l__zrefclever_type_
3060         \l__zrefclever_type_first_label_type_tl _options_prop
3061     }
3062     \l__zrefclever_name_format_tl
3063     \l__zrefclever_type_name_tl
3064     {
3065         \prop_get:cVNF
3066         {

```

```

3067         l__zrefclever_type_
3068         \l__zrefclever_type_first_label_type_tl _options_prop
3069     }
3070     \l__zrefclever_name_format_fallback_tl
3071     \l__zrefclever_type_name_tl
3072     {
3073         \tl_if_empty:NF \l__zrefclever_ref_decl_case_tl
3074         {
3075             \tl_put_left:Nn
3076                 \l__zrefclever_name_format_tl { - }
3077             \tl_put_left:NV \l__zrefclever_name_format_tl
3078                 \l__zrefclever_ref_decl_case_tl
3079             \tl_put_left:Nn
3080                 \l__zrefclever_name_format_fallback_tl { - }
3081             \tl_put_left:NV
3082                 \l__zrefclever_name_format_fallback_tl
3083                 \l__zrefclever_ref_decl_case_tl
3084         }
3085         \__zrefclever_get_lang_opt_type:xxxNF
3086         { \l__zrefclever_ref_language_tl }
3087         { \l__zrefclever_type_first_label_type_tl }
3088         { \l__zrefclever_name_format_tl }
3089         \l__zrefclever_type_name_tl
3090         {
3091             \__zrefclever_get_lang_opt_type:xxxNF
3092             { \l__zrefclever_ref_language_tl }
3093             { \l__zrefclever_type_first_label_type_tl }
3094             { \l__zrefclever_name_format_fallback_tl }
3095             \l__zrefclever_type_name_tl
3096             {
3097                 \tl_clear:N \l__zrefclever_type_name_tl
3098                 \msg_warning:nxxx { zref-clever }
3099                 { missing-name }
3100                 { \l__zrefclever_name_format_tl }
3101                 { \l__zrefclever_type_first_label_type_tl }
3102             }
3103         }
3104     }
3105 }
3106 }
3107 }
3108 }
3109
3110 % Signal whether the type name is to be included in the hyperlink or not.
3111 \bool_lazy_any:nTF
3112 {
3113     { ! \l__zrefclever_use_hyperref_bool }
3114     { \l__zrefclever_link_star_bool }
3115     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
3116     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
3117 }
3118 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
3119 {
3120     \bool_lazy_any:nTF

```



```

3121     {
3122       { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
3123       {
3124         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
3125         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
3126       }
3127       {
3128         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
3129         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
3130         \l__zrefclever_typeset_last_bool &&
3131         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
3132       }
3133     }
3134     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
3135     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
3136   }
3137 }

```

(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`.

```

3138 \cs_new:Npn \__zrefclever_extract_url_unexp:n #1
3139 {
3140   \zref@ifpropundefined { urluse }
3141   { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
3142   {
3143     \zref@ifrefcontainsprop {#1} { urluse }
3144     { \__zrefclever_extract_unexp:nnn {#1} { urluse } { \c_empty_tl } }
3145     { \__zrefclever_extract_unexp:nnn {#1} { url } { \c_empty_tl } }
3146   }
3147 }
3148 \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 {<label a>} {<label b>}

3149 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
3150 {
3151   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_a_tl
3152   {#1} { externaldocument } { \c_empty_tl }
3153   \__zrefclever_extract_default:Nnnn \l__zrefclever_label_extdoc_b_tl
3154   {#2} { externaldocument } { \c_empty_tl }

```

```

3155
3156 \tl_if_eq:NNT
3157   \l__zrefclever_label_extdoc_a_tl
3158   \l__zrefclever_label_extdoc_b_tl
3159   {
3160     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
3161     {
3162       \exp_args:Nxx \tl_if_eq:nnT
3163       { \__zrefclever_extract_unexp:nnn {#1} { zc@pgfmt } { } }
3164       { \__zrefclever_extract_unexp:nnn {#2} { zc@pgfmt } { } }
3165       {
3166         \int_compare:nNnTF
3167         { \__zrefclever_extract:nnn {#1} { zc@pgval } { -2 } + 1 }
3168         =
3169         { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
3170         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
3171         {
3172           \int_compare:nNnT
3173           { \__zrefclever_extract:nnn {#1} { zc@pgval } { -1 } }
3174           =
3175           { \__zrefclever_extract:nnn {#2} { zc@pgval } { -1 } }
3176           {
3177             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
3178             \bool_set_true:N \l__zrefclever_next_is_same_bool
3179           }
3180         }
3181       }
3182     }
3183   {
3184     \exp_args:Nxx \tl_if_eq:nnT
3185     { \__zrefclever_extract_unexp:nnn {#1} { zc@counter } { } }
3186     { \__zrefclever_extract_unexp:nnn {#2} { zc@counter } { } }
3187     {
3188       \exp_args:Nxx \tl_if_eq:nnT
3189       { \__zrefclever_extract_unexp:nnn {#1} { zc@enclval } { } }
3190       { \__zrefclever_extract_unexp:nnn {#2} { zc@enclval } { } }
3191       {
3192         \int_compare:nNnTF
3193         { \__zrefclever_extract:nnn {#1} { zc@cntval } { -2 } + 1 }
3194         =
3195         { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3196         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
3197         {
3198           \int_compare:nNnT
3199           { \__zrefclever_extract:nnn {#1} { zc@cntval } { -1 } }
3200           =
3201           { \__zrefclever_extract:nnn {#2} { zc@cntval } { -1 } }
3202           {
3203             \bool_set_true:N
3204             \l__zrefclever_next_maybe_range_bool
3205             \exp_args:Nxx \tl_if_eq:nnT
3206             {
3207               \__zrefclever_extract_unexp:nvn {#1}
3208               { \l__zrefclever_ref_property_tl } { }

```

```

3209     }
3210     {
3211         \__zrefclever_extract_unexp:nvn {#2}
3212         { l__zrefclever_ref_property_tl } { }
3213     }
3214     {
3215         \bool_set_true:N
3216         \l__zrefclever_next_is_same_bool
3217     }
3218 }
3219 }
3220 }
3221 }
3222 }
3223 }
3224 }

```

(End definition for __zrefclever_labels_in_sequence:nn.)

Finally, some functions for retrieving reference options values, according to the relevant precedence rules. They both receive an *<option>* as argument, and store the retrieved value in an appropriate. 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 each of these functions is the kind of option each should be used for. __zrefclever_get_ref_opt_typeset:nN is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fallback”). __zrefclever_get_ref_opt_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_opt_bool:nN is intended for boolean options.

```

\__zrefclever_get_ref_opt_typeset:nN      \__zrefclever_get_ref_opt_typeset:nN {<option>} {<tl variable>}
3225 \cs_new_protected:Npn \__zrefclever_get_ref_opt_typeset:nN #1#2
3226 {
3227     % First attempt: general options.
3228     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3229     {
3230         % If not found, try type specific options.
3231         \bool_lazy_and:nnTF
3232         {
3233             \prop_if_exist_p:c
3234             {
3235                 l__zrefclever_type_
3236                 \l__zrefclever_label_type_a_tl _options_prop
3237             }
3238         }
3239         {
3240             \prop_if_in_p:cn
3241             {
3242                 l__zrefclever_type_
3243                 \l__zrefclever_label_type_a_tl _options_prop
3244             }

```

```

3245         {#1}
3246     }
3247     {
3248         \prop_get:cnN
3249         {
3250             l__zrefclever_type_
3251             \l__zrefclever_label_type_a_tl _options_prop
3252         }
3253         {#1} #2
3254     }
3255     {
3256         % If not found, try type- and language-specific.
3257         \__zrefclever_get_lang_opt_type:xnNF
3258         { \l__zrefclever_ref_language_tl }
3259         { \l__zrefclever_label_type_a_tl }
3260         {#1} #2
3261         {
3262             % If not found, try language-specific default.
3263             \__zrefclever_get_lang_opt_default:xnNF
3264             { \l__zrefclever_ref_language_tl }
3265             {#1} #2
3266             {
3267                 % If not found, try fallback.
3268                 \__zrefclever_get_fallback_unknown_lang_opt:nNF {#1} #2
3269                 {
3270                     \tl_clear:N #2
3271                     \msg_warning:nnn { zref-clever }
3272                     { missing-string } {#1}
3273                 }
3274             }
3275         }
3276     }
3277 }
3278 }

```

(End definition for __zrefclever_get_ref_opt_typeset:nN.)

```

\__zrefclever_get_ref_opt_font:nN      \__zrefclever_get_ref_opt_font:nN {<option>} {<tl variable>}
3279 \cs_new_protected:Npn \__zrefclever_get_ref_opt_font:nN #1#2
3280 {
3281     % First attempt: general options.
3282     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
3283     {
3284         % If not found, try type specific options.
3285         \bool_if:nTF
3286         {
3287             \prop_if_exist_p:c
3288             {
3289                 l__zrefclever_type_
3290                 \l__zrefclever_label_type_a_tl _options_prop
3291             }
3292         }
3293         {
3294             \prop_get:cnNF

```

```

3295         {
3296             l__zrefclever_type_
3297             \l__zrefclever_label_type_a_tl _options_prop
3298         }
3299         {#1} #2
3300         { \tl_clear:N #2 }
3301     }
3302     { \tl_clear:N #2 }
3303 }
3304 }

```

(End definition for __zrefclever_get_ref_opt_font:nn.)

```

\__zrefclever_get_ref_opt_bool:nnN    \__zrefclever_get_ref_opt_bool:nn {<option>}
                                       {<default>} {<bool variable>}
3305 \cs_new_protected:Npn \__zrefclever_get_ref_opt_bool:nnN #1#2#3
3306 {
3307     % First attempt: general options.
3308     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} \l_tmpa_tl
3309     {
3310         % If not found, try type specific options.
3311         \bool_lazy_and:nnTF
3312         {
3313             \prop_if_exist_p:c
3314             {
3315                 l__zrefclever_type_
3316                 \l__zrefclever_label_type_a_tl _options_prop
3317             }
3318         }
3319         {
3320             \prop_if_in_p:cn
3321             {
3322                 l__zrefclever_type_
3323                 \l__zrefclever_label_type_a_tl _options_prop
3324             }
3325             {#1}
3326         }
3327         {
3328             \prop_get:cnN
3329             {
3330                 l__zrefclever_type_
3331                 \l__zrefclever_label_type_a_tl _options_prop
3332             }
3333             {#1} \l_tmpa_tl
3334         }
3335         {
3336             % If not found, try type- and language-specific.
3337             \__zrefclever_get_lang_opt_type:xxnNF
3338             { \l__zrefclever_ref_language_tl }
3339             { \l__zrefclever_label_type_a_tl }
3340             {#1} \l_tmpa_tl
3341             {
3342                 % If not found, try language-specific default.
3343                 \__zrefclever_get_lang_opt_default:xxnNF

```

```

3344         { \l__zrefclever_ref_language_tl }
3345         {#1} \l_tmpa_tl
3346         {
3347             % If not found, use default argument.
3348             \bool_lazy_or:nnTF
3349             { \str_if_eq_p:nn {#2} { true } }
3350             { \str_if_eq_p:nn {#2} { false } }
3351             { \tl_set:Nn \l_tmpa_tl {#2} }
3352             {
3353                 % And, if even that fails, presume false.
3354                 \tl_set:Nn \l_tmpa_tl { false }
3355             }
3356         }
3357     }
3358 }
3359 }
3360 % Having retrieved the option value, set the boolean. At this point, we
3361 % *know* '\l_tmpa_tl' is either 'true' or 'false'.
3362 \use:c { bool_set_ \l_tmpa_tl :N } #3
3363 }

```

(End definition for `__zrefclever_get_ref_opt_bool:nnN`.)

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.

```

3364 \__zrefclever_compat_module:nn { appendix }
3365 {
3366     \AddToHook { cmd / appendix / before }
3367     {
3368         \__zrefclever_zcsetup:n

```

```

3369     {
3370         countertype =
3371         {
3372             chapter      = appendix ,
3373             section      = appendix ,
3374             subsection   = appendix ,
3375             subsubsection = appendix ,
3376             paragraph    = appendix ,
3377             subparagraph = appendix ,
3378         }
3379     }
3380 }
3381 }

```

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

9.2 appendices

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

```

3382 \__zrefclever_compat_module:nn { appendices }
3383 {
3384     \__zrefclever_if_package_loaded:nT { appendix }
3385     {
3386         \newcounter { zc@appendix }
3387         \newcounter { zc@save@appendix }
3388         \setcounter { zc@appendix } { 0 }
3389         \setcounter { zc@save@appendix } { 0 }
3390         \cs_if_exist:cTF { chapter }
3391         {
3392             \__zrefclever_zcsetup:n
3393             { counterresetby = { chapter = zc@appendix } }
3394         }
3395         {
3396             \cs_if_exist:cT { section }
3397             {
3398                 \__zrefclever_zcsetup:n
3399                 { counterresetby = { section = zc@appendix } }
3400             }
3401         }
3402         \AddToHook { env / appendices / begin }
3403         {
3404             \stepcounter { zc@save@appendix }
3405             \setcounter { zc@appendix } { \value { zc@save@appendix } }
3406             \__zrefclever_zcsetup:n
3407             {

```

```

3408         countertype =
3409         {
3410             chapter      = appendix ,
3411             section      = appendix ,
3412             subsection   = appendix ,
3413             subsubsection = appendix ,
3414             paragraph    = appendix ,
3415             subparagraph = appendix ,
3416         }
3417     }
3418 }
3419 \AddToHook { env / appendices / end }
3420 { \setcounter { zc@appendix } { 0 } }
3421 \AddToHook { cmd / appendix / before }
3422 {
3423     \stepcounter { zc@save@appendix }
3424     \setcounter { zc@appendix } { \value { zc@save@appendix } }
3425 }
3426 \AddToHook { env / subappendices / begin }
3427 {
3428     \__zrefclever_zcsetup:n
3429     {
3430         countertype =
3431         {
3432             section      = appendix ,
3433             subsection   = appendix ,
3434             subsubsection = appendix ,
3435             paragraph    = appendix ,
3436             subparagraph = appendix ,
3437         } ,
3438     }
3439 }
3440 \msg_info:nnn { zref-clever } { compat-package } { appendix }
3441 }
3442 }

```

9.3 memoir

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

1. Caption functionality which receives $\langle label \rangle$ as optional argument, namely:

- (a) The `sidecaption` and `sidecontcaption` environments. These environments *store* the label in an internal macro, `\m@mscaplabel`, at the begin environment code (more precisely in `\@@sidecaption`), but both the call to `\refstepcounter` and the expansion of `\m@mscaplabel` take place at `\endsidecaption`. For this reason, hooks are not particularly helpful, and there is not any easy way to grab the $\langle label \rangle$ argument to start with. I can see

two ways to deal with these environments, none of which I really like. First, map through `\m@mscaplabel` until `\label` is found, then grab the next token which is the `\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.

```

3443 \__zrefclever_compat_module:nn { memoir }
3444 {
3445   \__zrefclever_if_class_loaded:nT { memoir }
3446   {

```

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`.

```

3447   \__zrefclever_zcsetup:n
3448   {
3449     countertype =
3450     {
3451       subfigure = figure ,
3452       subtable  = table  ,
3453       poemline  = line   ,

```

```

3454         } ,
3455         counterresetby =
3456         {
3457             subfigure = figure ,
3458             subtable = table ,
3459         } ,
3460     }

```

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

```

3461 \cs_new_protected:Npn \__zrefclever_memoir_both_labels:
3462 {
3463     \cs_set_eq:NN \__zrefclever_memoir_orig_label:n \label
3464     \cs_set:Npn \__zrefclever_memoir_label_and_zlabel:n ##1
3465     {
3466         \__zrefclever_memoir_orig_label:n {##1}
3467         \zlabel{##1}
3468     }
3469     \cs_set_eq:NN \label \__zrefclever_memoir_label_and_zlabel:n
3470 }
3471 \AddToHook { env / sidecaption / begin }
3472 { \__zrefclever_memoir_both_labels: }
3473 \AddToHook { env / sidecontcaption / begin }
3474 { \__zrefclever_memoir_both_labels: }
3475 \AddToHook { cmd / bitwonumcaption / before }
3476 { \group_begin: \__zrefclever_memoir_both_labels: }
3477 \AddToHook { cmd / bitwonumcaption / after }
3478 { \group_end: }
3479 \AddToHook { cmd / bionenumcaption / before }
3480 { \group_begin: \__zrefclever_memoir_both_labels: }
3481 \AddToHook { cmd / bionenumcaption / after }
3482 { \group_end: }
3483 \AddToHook { cmd / bicaption / before }
3484 { \group_begin: \__zrefclever_memoir_both_labels: }
3485 \AddToHook { cmd / bicaption / after }
3486 { \group_end: }

```

Support for subcaption reference.

```

3487 \zref@newprop { subcaption }
3488 { \cs_if_exist_use:c { @thesub \@captype } }
3489 \AddToHook { cmd / @memsubcaption / before }
3490 { \zref@localaddprop \ZREF@mainlist { subcaption } }

```

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

```

3491 \tl_new:N \l__zrefclever_memoir_footnote_type_tl
3492 \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { footnote }
3493 \AddToHook { env / minipage / begin }
3494 { \tl_set:Nn \l__zrefclever_memoir_footnote_type_tl { mpfootnote } }
3495 \AddToHook { cmd / @makefntext / before }
3496 {
3497     \__zrefclever_zcsetup:x
3498     { currentcounter = \l__zrefclever_memoir_footnote_type_tl }
3499 }
3500 \AddToHook { cmd / @makesidefntext / before }
3501 { \__zrefclever_zcsetup:n { currentcounter = sidefootnote } }

```

```

3502     \__zrefclever_zcsetup:n
3503     {
3504         countertype =
3505         {
3506             sidefootnote = footnote ,
3507             pagenote = endnote ,
3508         } ,
3509     }
3510     \AddToHook { file / \jobname.ent / before }
3511     { \__zrefclever_zcsetup:x { currentcounter = pagenote } }
3512     \msg_info:nnn { zref-clever } { compat-class } { memoir }
3513 }
3514 }

```

9.4 KOMA

Support for KOMA-Script document classes.

```

3515 \__zrefclever_compat_module:nn { KOMA }
3516 {
3517     \cs_if_exist:NT \KOMAClassName
3518     {

```

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.

```

3519     \AddToHook { env / captionbeside / after }
3520     {
3521         \tl_if_exist:NT \@capttype
3522         { \tl_set_eq:NN \@currentcounter \@capttype }
3523     }
3524     \tl_new:N \g__zrefclever_koma_captionofbeside_capttype_tl
3525     \AddToHook { env / captionofbeside / end }
3526     { \tl_gset_eq:NN \g__zrefclever_koma_capttype_tl \@capttype }
3527     \AddToHook { env / captionofbeside / after }
3528     {
3529         \tl_if_eq:NnF \@currentenvir { document }
3530         {
3531             \tl_if_empty:NF \g__zrefclever_koma_capttype_tl
3532             {
3533                 \tl_set_eq:NN
3534                 \@currentcounter \g__zrefclever_koma_capttype_tl
3535             }
3536         }
3537         \tl_gclear:N \g__zrefclever_koma_capttype_tl
3538     }

```

```

3539         \msg_info:nnx { zref-clever } { compat-class } { \KOMAClassName }
3540     }
3541 }

```

9.5 amsmath

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

```

3542 \__zrefclever_compat_module:nn { amsmath }
3543 {
3544     \__zrefclever_if_package_loaded:nT { amsmath }
3545     {

```

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...).

```

3546         \cs_set_nopar:Npn \__zrefclever_ltxlabel:n #1
3547         {
3548             \__zrefclever_orig_ltxlabel:n {#1}
3549             \zref@wrapper@babel \zref@label {#1}
3550         }

```

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>.

```

3551     \__zrefclever_if_package_loaded:nTF { hyperref }
3552     {
3553         \AddToHook { package / nameref / after }
3554         {
3555             \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label

```

```

3556         \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3557     }
3558 }
3559 {
3560     \cs_new_eq:NN \__zrefclever_orig_ltxlabel:n \ltx@label
3561     \cs_set_eq:NN \ltx@label \__zrefclever_ltxlabel:n
3562 }

```

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).

```

3563 \AddToHook { env / subequations / begin }
3564 {
3565     \__zrefclever_zcsetup:x
3566     {
3567         counterresetby =
3568         {
3569             parentequation =
3570             \__zrefclever_counter_reset_by:n { equation } ,
3571             equation = parentequation ,
3572         } ,
3573         currentcounter = parentequation ,
3574         countertype = { parentequation = equation } ,
3575     }
3576 }

```

`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.

```

3577 \clist_map_inline:nn
3578 {
3579     equation ,
3580     equation* ,
3581     align ,
3582     align* ,
3583     alignat ,
3584     alignat* ,

```

```

3585         flalign ,
3586         flalign* ,
3587         xalignat ,
3588         xalignat* ,
3589         gather ,
3590         gather* ,
3591         multiline ,
3592         multiline* ,
3593     }
3594 {
3595     \AddToHook { env / #1 / begin }
3596     { \_zrefclever_zcsetup:n { currentcounter = equation } }
3597 }

```

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

```

3598 \zcRefTypeSetup { equation }
3599 {
3600     reffont = \upshape ,
3601     preref = {\textup{()} } ,
3602     postref = {\textup{()}} ,
3603 }
3604 \msg_info:nnn { zref-clever } { compat-package } { amsmath }
3605 }
3606 }

```

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.

```

3607 \bool_new:N \l__zrefclever_mathtools_showonlyrefs_bool
3608 \__zrefclever_compat_module:nn { mathtools }
3609 {
3610     \__zrefclever_if_package_loaded:nT { mathtools }
3611     {
3612         \MH_if_boolean:nT { show_only_refs }
3613         {
3614             \bool_set_true:N \l__zrefclever_mathtools_showonlyrefs_bool
3615             \cs_new_protected:Npn \__zrefclever_mathtools_showonlyrefs:n #1
3616             {
3617                 \@bsphack
3618                 \seq_map_inline:Nn #1
3619                 {
3620                     \exp_args:Nx \tl_if_eq:nnTF
3621                     { \__zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3622                     { equation }

```

```

3623         {
3624             \protected@write \@auxout { }
3625             { \string \MT@newlabel {##1} }
3626         }
3627         {
3628             \exp_args:Nx \tl_if_eq:nnT
3629             { \_zrefclever_extract_unexp:nnn {##1} { zc@type } { } }
3630             { parentequation }
3631             {
3632                 \protected@write \@auxout { }
3633                 { \string \MT@newlabel {##1} }
3634             }
3635         }
3636     }
3637     \esphack
3638 }
3639 \msg_info:nnn { zref-clever } { compat-package } { mathtools }
3640 }
3641 }
3642 }

```

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”.

```

3643 \_zrefclever_compat_module:nn { breqn }
3644 {
3645     \_zrefclever_if_package_loaded:nT { breqn }
3646 }

```

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>).

```

3647 \AddToHook { env / dgroup / begin }
3648 {
3649     \_zrefclever_zcsetup:x
3650     {
3651         counterresetby =
3652         {
3653             parentequation =
3654             \_zrefclever_counter_reset_by:n { equation } ,
3655             equation = parentequation ,
3656         } ,
3657         currentcounter = parentequation ,
3658         countertype = { parentequation = equation } ,
3659     }

```

```

3660     }
3661     \clist_map_inline:nn
3662     {
3663         dmath ,
3664         dseries ,
3665         darray ,
3666     }
3667     {
3668         \AddToHook { env / #1 / begin }
3669         { \__zrefclever_zcsetup:n { currentcounter = equation } }
3670     }
3671     \msg_info:nnn { zref-clever } { compat-package } { breqn }
3672 }
3673 }

```

9.8 listings

```

3674 \__zrefclever_compat_module:nn { listings }
3675 {
3676     \__zrefclever_if_package_loaded:nT { listings }
3677     {
3678         \__zrefclever_zcsetup:n
3679         {
3680             countertype =
3681             {
3682                 lstlisting = listing ,
3683                 lstnumber = line ,
3684             } ,
3685             counterresetby = { lstnumber = lstlisting } ,
3686         }

```

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.

```

3687     \lst@AddToHook { PreInit }
3688     { \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.

```

3689     \lst@AddToHook { Init }
3690     { \__zrefclever_zcsetup:n { currentcounter = lstnumber } }
3691     \msg_info:nnn { zref-clever } { compat-package } { listings }
3692 }
3693 }

```


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

```

3694 \__zrefclever_compat_module:nn { enumitem }
3695 {
3696   \__zrefclever_if_package_loaded:nT { enumitem }
3697   {
3698     \int_set:Nn \l_tmpa_int { 5 }
3699     \bool_while_do:nn
3700     {
3701       \cs_if_exist_p:c
3702       { c@enum \int_to_roman:n { \l_tmpa_int } }
3703     }
3704     {
3705       \__zrefclever_zcsetup:x
3706       {
3707         counterresetby =
3708         {
3709           enum \int_to_roman:n { \l_tmpa_int } =
3710           enum \int_to_roman:n { \l_tmpa_int - 1 }
3711         } ,
3712         countertype =
3713         { enum \int_to_roman:n { \l_tmpa_int } = item } ,
3714       }
3715       \int_incr:N \l_tmpa_int
3716     }
3717     \int_compare:nNnT { \l_tmpa_int } > { 5 }
3718     { \msg_info:nnn { zref-clever } { compat-package } { enumitem } }
3719   }
3720 }

```

9.10 subcaption

```

3721 \__zrefclever_compat_module:nn { subcaption }
3722 {
3723   \__zrefclever_if_package_loaded:nT { subcaption }
3724   {
3725     \__zrefclever_zcsetup:n
3726     {
3727       countertype =
3728       {

```

```

3729         subfigure = figure ,
3730         subtable = table ,
3731     } ,
3732     counterresetby =
3733     {
3734         subfigure = figure ,
3735         subtable = table ,
3736     } ,
3737 }

```

Support for `subref` reference.

```

3738     \zref@newprop { subref }
3739     { \cs_if_exist_use:c { thesub \@captype } }
3740     \tl_put_right:Nn \caption@subtyphook
3741     { \zref@localaddprop \ZREF@mainlist { subref } }
3742 }
3743 }

```

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.

```

3744 \__zrefclever_compat_module:nn { subfig }
3745 {
3746     \__zrefclever_if_package_loaded:nT { subfig }
3747     {
3748         \__zrefclever_zcsetup:n
3749         {
3750             countertype =
3751             {
3752                 subfigure = figure ,
3753                 subtable = table ,
3754             } ,
3755             counterresetby =
3756             {
3757                 subfigure = figure ,
3758                 subtable = table ,
3759             } ,
3760         }
3761     }
3762 }
3763 </package>

```

10 Language files

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

10.1 English

English language file has been initially provided by the author.

```
3764 <*package>
3765 \zcDeclareLanguage { english }
3766 \zcDeclareLanguageAlias { american } { english }
3767 \zcDeclareLanguageAlias { australian } { english }
3768 \zcDeclareLanguageAlias { british } { english }
3769 \zcDeclareLanguageAlias { canadian } { english }
3770 \zcDeclareLanguageAlias { newzealand } { english }
3771 \zcDeclareLanguageAlias { UKenglish } { english }
3772 \zcDeclareLanguageAlias { USenglish } { english }
3773 </package>

3774 <*lang-english>

3775 namesep = {\nobreakspace} ,
3776 pairsep = {\~and\nobreakspace} ,
3777 listsep = {\~,~} ,
3778 lastsep = {\~and\nobreakspace} ,
3779 tpairsep = {\~and\nobreakspace} ,
3780 tlistsep = {\~,~} ,
3781 tlastsep = {\~,~and\nobreakspace} ,
3782 notesep = {\~} ,
3783 rangesep = {\~to\nobreakspace} ,
3784
3785 type = book ,
3786   Name-sg = Book ,
3787   name-sg = book ,
3788   Name-pl = Books ,
3789   name-pl = books ,
3790
3791 type = part ,
3792   Name-sg = Part ,
3793   name-sg = part ,
3794   Name-pl = Parts ,
3795   name-pl = parts ,
3796
3797 type = chapter ,
3798   Name-sg = Chapter ,
3799   name-sg = chapter ,
3800   Name-pl = Chapters ,
3801   name-pl = chapters ,
3802
3803 type = section ,
3804   Name-sg = Section ,
3805   name-sg = section ,
3806   Name-pl = Sections ,
3807   name-pl = sections ,
3808
3809 type = paragraph ,
3810   Name-sg = Paragraph ,
3811   name-sg = paragraph ,
3812   Name-pl = Paragraphs ,
3813   name-pl = paragraphs ,
```

```

3814 Name-sg-ab = Par. ,
3815 name-sg-ab = par. ,
3816 Name-pl-ab = Par. ,
3817 name-pl-ab = par. ,
3818
3819 type = appendix ,
3820 Name-sg = Appendix ,
3821 name-sg = appendix ,
3822 Name-pl = Appendices ,
3823 name-pl = appendices ,
3824
3825 type = page ,
3826 Name-sg = Page ,
3827 name-sg = page ,
3828 Name-pl = Pages ,
3829 name-pl = pages ,
3830 rangsep = {\textendash} ,
3831
3832 type = line ,
3833 Name-sg = Line ,
3834 name-sg = line ,
3835 Name-pl = Lines ,
3836 name-pl = lines ,
3837
3838 type = figure ,
3839 Name-sg = Figure ,
3840 name-sg = figure ,
3841 Name-pl = Figures ,
3842 name-pl = figures ,
3843 Name-sg-ab = Fig. ,
3844 name-sg-ab = fig. ,
3845 Name-pl-ab = Figs. ,
3846 name-pl-ab = figs. ,
3847
3848 type = table ,
3849 Name-sg = Table ,
3850 name-sg = table ,
3851 Name-pl = Tables ,
3852 name-pl = tables ,
3853
3854 type = item ,
3855 Name-sg = Item ,
3856 name-sg = item ,
3857 Name-pl = Items ,
3858 name-pl = items ,
3859
3860 type = footnote ,
3861 Name-sg = Footnote ,
3862 name-sg = footnote ,
3863 Name-pl = Footnotes ,
3864 name-pl = footnotes ,
3865
3866 type = endnote ,
3867 Name-sg = Note ,

```

```

3868     name-sg = note ,
3869     Name-pl = Notes ,
3870     name-pl = notes ,
3871
3872 type = note ,
3873     Name-sg = Note ,
3874     name-sg = note ,
3875     Name-pl = Notes ,
3876     name-pl = notes ,
3877
3878 type = equation ,
3879     Name-sg = Equation ,
3880     name-sg = equation ,
3881     Name-pl = Equations ,
3882     name-pl = equations ,
3883     Name-sg-ab = Eq. ,
3884     name-sg-ab = eq. ,
3885     Name-pl-ab = Eqs. ,
3886     name-pl-ab = eqs. ,
3887     preref = {() ,
3888     postref = {()} ,
3889
3890 type = theorem ,
3891     Name-sg = Theorem ,
3892     name-sg = theorem ,
3893     Name-pl = Theorems ,
3894     name-pl = theorems ,
3895
3896 type = lemma ,
3897     Name-sg = Lemma ,
3898     name-sg = lemma ,
3899     Name-pl = Lemmas ,
3900     name-pl = lemmas ,
3901
3902 type = corollary ,
3903     Name-sg = Corollary ,
3904     name-sg = corollary ,
3905     Name-pl = Corollaries ,
3906     name-pl = corollaries ,
3907
3908 type = proposition ,
3909     Name-sg = Proposition ,
3910     name-sg = proposition ,
3911     Name-pl = Propositions ,
3912     name-pl = propositions ,
3913
3914 type = definition ,
3915     Name-sg = Definition ,
3916     name-sg = definition ,
3917     Name-pl = Definitions ,
3918     name-pl = definitions ,
3919
3920 type = proof ,
3921     Name-sg = Proof ,

```

```

3922   name-sg = proof ,
3923   Name-pl = Proofs ,
3924   name-pl = proofs ,
3925
3926   type = result ,
3927   Name-sg = Result ,
3928   name-sg = result ,
3929   Name-pl = Results ,
3930   name-pl = results ,
3931
3932   type = remark ,
3933   Name-sg = Remark ,
3934   name-sg = remark ,
3935   Name-pl = Remarks ,
3936   name-pl = remarks ,
3937
3938   type = example ,
3939   Name-sg = Example ,
3940   name-sg = example ,
3941   Name-pl = Examples ,
3942   name-pl = examples ,
3943
3944   type = algorithm ,
3945   Name-sg = Algorithm ,
3946   name-sg = algorithm ,
3947   Name-pl = Algorithms ,
3948   name-pl = algorithms ,
3949
3950   type = listing ,
3951   Name-sg = Listing ,
3952   name-sg = listing ,
3953   Name-pl = Listings ,
3954   name-pl = listings ,
3955
3956   type = exercise ,
3957   Name-sg = Exercise ,
3958   name-sg = exercise ,
3959   Name-pl = Exercises ,
3960   name-pl = exercises ,
3961
3962   type = solution ,
3963   Name-sg = Solution ,
3964   name-sg = solution ,
3965   Name-pl = Solutions ,
3966   name-pl = solutions ,
3967 </lang-english>

```

10.2 German

German language file has been initially provided by the author.

```

3968 <*package>
3969 \zcDeclareLanguage
3970 [ declension = { N , A , D , G } , gender = { f , m , n } , allcaps ]

```

```

3971 { german }
3972 \zcDeclareLanguageAlias { austrian      } { german }
3973 \zcDeclareLanguageAlias { germanb       } { german }
3974 \zcDeclareLanguageAlias { ngerman       } { german }
3975 \zcDeclareLanguageAlias { naustrian     } { german }
3976 \zcDeclareLanguageAlias { nswissgerman } { german }
3977 \zcDeclareLanguageAlias { swissgerman   } { german }
3978 \end{package}
3979 <lang-german>

3980 namesep = {\nobreakspace} ,
3981 pairsep = {\und\nobreakspace} ,
3982 listsep = {,~} ,
3983 lastsep = {\und\nobreakspace} ,
3984 tpairsep = {\und\nobreakspace} ,
3985 tlistsep = {,~} ,
3986 tlastsep = {\und\nobreakspace} ,
3987 notesep = {~} ,
3988 rangesep = {\bis\nobreakspace} ,
3989
3990 type = book ,
3991 gender = n ,
3992 case = N ,
3993   Name-sg = Buch ,
3994   Name-pl = Bücher ,
3995 case = A ,
3996   Name-sg = Buch ,
3997   Name-pl = Bücher ,
3998 case = D ,
3999   Name-sg = Buch ,
4000   Name-pl = Büchern ,
4001 case = G ,
4002   Name-sg = Buches ,
4003   Name-pl = Bücher ,
4004
4005 type = part ,
4006 gender = m ,
4007 case = N ,
4008   Name-sg = Teil ,
4009   Name-pl = Teile ,
4010 case = A ,
4011   Name-sg = Teil ,
4012   Name-pl = Teile ,
4013 case = D ,
4014   Name-sg = Teil ,
4015   Name-pl = Teilen ,
4016 case = G ,
4017   Name-sg = Teiles ,
4018   Name-pl = Teile ,
4019
4020 type = chapter ,
4021 gender = n ,
4022 case = N ,
4023   Name-sg = Kapitel ,

```

```

4024     Name-pl = Kapitel ,
4025     case = A ,
4026     Name-sg = Kapitel ,
4027     Name-pl = Kapitel ,
4028     case = D ,
4029     Name-sg = Kapitel ,
4030     Name-pl = Kapiteln ,
4031     case = G ,
4032     Name-sg = Kapiteln ,
4033     Name-pl = Kapitel ,
4034
4035 type = section ,
4036 gender = m ,
4037 case = N ,
4038     Name-sg = Abschnitt ,
4039     Name-pl = Abschnitte ,
4040 case = A ,
4041     Name-sg = Abschnitt ,
4042     Name-pl = Abschnitte ,
4043 case = D ,
4044     Name-sg = Abschnitt ,
4045     Name-pl = Abschnitten ,
4046 case = G ,
4047     Name-sg = Abschnitts ,
4048     Name-pl = Abschnitte ,
4049
4050 type = paragraph ,
4051 gender = m ,
4052 case = N ,
4053     Name-sg = Absatz ,
4054     Name-pl = Absätze ,
4055 case = A ,
4056     Name-sg = Absatz ,
4057     Name-pl = Absätze ,
4058 case = D ,
4059     Name-sg = Absatz ,
4060     Name-pl = Absätzen ,
4061 case = G ,
4062     Name-sg = Absatzes ,
4063     Name-pl = Absätze ,
4064
4065 type = appendix ,
4066 gender = m ,
4067 case = N ,
4068     Name-sg = Anhang ,
4069     Name-pl = Anhänge ,
4070 case = A ,
4071     Name-sg = Anhang ,
4072     Name-pl = Anhänge ,
4073 case = D ,
4074     Name-sg = Anhang ,
4075     Name-pl = Anhängen ,
4076 case = G ,
4077     Name-sg = Anhangs ,

```



```

4078     Name-pl = Anhänge ,
4079
4080 type = page ,
4081     gender = f ,
4082     case = N ,
4083     Name-sg = Seite ,
4084     Name-pl = Seiten ,
4085     case = A ,
4086     Name-sg = Seite ,
4087     Name-pl = Seiten ,
4088     case = D ,
4089     Name-sg = Seite ,
4090     Name-pl = Seiten ,
4091     case = G ,
4092     Name-sg = Seite ,
4093     Name-pl = Seiten ,
4094     rangeseq = {\textendash} ,
4095
4096 type = line ,
4097     gender = f ,
4098     case = N ,
4099     Name-sg = Zeile ,
4100     Name-pl = Zeilen ,
4101     case = A ,
4102     Name-sg = Zeile ,
4103     Name-pl = Zeilen ,
4104     case = D ,
4105     Name-sg = Zeile ,
4106     Name-pl = Zeilen ,
4107     case = G ,
4108     Name-sg = Zeile ,
4109     Name-pl = Zeilen ,
4110
4111 type = figure ,
4112     gender = f ,
4113     case = N ,
4114     Name-sg = Abbildung ,
4115     Name-pl = Abbildungen ,
4116     Name-sg-ab = Abb. ,
4117     Name-pl-ab = Abb. ,
4118     case = A ,
4119     Name-sg = Abbildung ,
4120     Name-pl = Abbildungen ,
4121     Name-sg-ab = Abb. ,
4122     Name-pl-ab = Abb. ,
4123     case = D ,
4124     Name-sg = Abbildung ,
4125     Name-pl = Abbildungen ,
4126     Name-sg-ab = Abb. ,
4127     Name-pl-ab = Abb. ,
4128     case = G ,
4129     Name-sg = Abbildung ,
4130     Name-pl = Abbildungen ,
4131     Name-sg-ab = Abb. ,

```

```

4132     Name-pl-ab = Abb. ,
4133
4134 type = table ,
4135     gender = f ,
4136     case = N ,
4137     Name-sg = Tabelle ,
4138     Name-pl = Tabellen ,
4139     case = A ,
4140     Name-sg = Tabelle ,
4141     Name-pl = Tabellen ,
4142     case = D ,
4143     Name-sg = Tabelle ,
4144     Name-pl = Tabellen ,
4145     case = G ,
4146     Name-sg = Tabelle ,
4147     Name-pl = Tabellen ,
4148
4149 type = item ,
4150     gender = m ,
4151     case = N ,
4152     Name-sg = Punkt ,
4153     Name-pl = Punkte ,
4154     case = A ,
4155     Name-sg = Punkt ,
4156     Name-pl = Punkte ,
4157     case = D ,
4158     Name-sg = Punkt ,
4159     Name-pl = Punkten ,
4160     case = G ,
4161     Name-sg = Punktes ,
4162     Name-pl = Punkte ,
4163
4164 type = footnote ,
4165     gender = f ,
4166     case = N ,
4167     Name-sg = Fußnote ,
4168     Name-pl = Fußnoten ,
4169     case = A ,
4170     Name-sg = Fußnote ,
4171     Name-pl = Fußnoten ,
4172     case = D ,
4173     Name-sg = Fußnote ,
4174     Name-pl = Fußnoten ,
4175     case = G ,
4176     Name-sg = Fußnote ,
4177     Name-pl = Fußnoten ,
4178
4179 type = endnote ,
4180     gender = f ,
4181     case = N ,
4182     Name-sg = Endnote ,
4183     Name-pl = Endnoten ,
4184     case = A ,
4185     Name-sg = Endnote ,

```

```

4186     Name-pl = Endnoten ,
4187     case = D ,
4188     Name-sg = Endnote ,
4189     Name-pl = Endnoten ,
4190     case = G ,
4191     Name-sg = Endnote ,
4192     Name-pl = Endnoten ,
4193
4194     type = note ,
4195     gender = f ,
4196     case = N ,
4197     Name-sg = Anmerkung ,
4198     Name-pl = Anmerkungen ,
4199     case = A ,
4200     Name-sg = Anmerkung ,
4201     Name-pl = Anmerkungen ,
4202     case = D ,
4203     Name-sg = Anmerkung ,
4204     Name-pl = Anmerkungen ,
4205     case = G ,
4206     Name-sg = Anmerkung ,
4207     Name-pl = Anmerkungen ,
4208
4209     type = equation ,
4210     gender = f ,
4211     case = N ,
4212     Name-sg = Gleichung ,
4213     Name-pl = Gleichungen ,
4214     case = A ,
4215     Name-sg = Gleichung ,
4216     Name-pl = Gleichungen ,
4217     case = D ,
4218     Name-sg = Gleichung ,
4219     Name-pl = Gleichungen ,
4220     case = G ,
4221     Name-sg = Gleichung ,
4222     Name-pl = Gleichungen ,
4223     preref = {() ,
4224     postref = {}} ,
4225
4226     type = theorem ,
4227     gender = n ,
4228     case = N ,
4229     Name-sg = Theorem ,
4230     Name-pl = Theoreme ,
4231     case = A ,
4232     Name-sg = Theorem ,
4233     Name-pl = Theoreme ,
4234     case = D ,
4235     Name-sg = Theorem ,
4236     Name-pl = Theoremen ,
4237     case = G ,
4238     Name-sg = Theorems ,
4239     Name-pl = Theoreme ,

```

```

4240
4241 type = lemma ,
4242   gender = n ,
4243   case = N ,
4244     Name-sg = Lemma ,
4245     Name-pl = Lemmata ,
4246   case = A ,
4247     Name-sg = Lemma ,
4248     Name-pl = Lemmata ,
4249   case = D ,
4250     Name-sg = Lemma ,
4251     Name-pl = Lemmata ,
4252   case = G ,
4253     Name-sg = Lemmas ,
4254     Name-pl = Lemmata ,
4255
4256 type = corollary ,
4257   gender = n ,
4258   case = N ,
4259     Name-sg = Korollar ,
4260     Name-pl = Korollare ,
4261   case = A ,
4262     Name-sg = Korollar ,
4263     Name-pl = Korollare ,
4264   case = D ,
4265     Name-sg = Korollar ,
4266     Name-pl = Korollaren ,
4267   case = G ,
4268     Name-sg = Korollars ,
4269     Name-pl = Korollare ,
4270
4271 type = proposition ,
4272   gender = m ,
4273   case = N ,
4274     Name-sg = Satz ,
4275     Name-pl = Sätze ,
4276   case = A ,
4277     Name-sg = Satz ,
4278     Name-pl = Sätze ,
4279   case = D ,
4280     Name-sg = Satz ,
4281     Name-pl = Sätzen ,
4282   case = G ,
4283     Name-sg = Satzes ,
4284     Name-pl = Sätze ,
4285
4286 type = definition ,
4287   gender = f ,
4288   case = N ,
4289     Name-sg = Definition ,
4290     Name-pl = Definitionen ,
4291   case = A ,
4292     Name-sg = Definition ,
4293     Name-pl = Definitionen ,

```

```

4294 case = D ,
4295     Name-sg = Definition ,
4296     Name-pl = Definitionen ,
4297 case = G ,
4298     Name-sg = Definition ,
4299     Name-pl = Definitionen ,
4300
4301 type = proof ,
4302     gender = m ,
4303     case = N ,
4304         Name-sg = Beweis ,
4305         Name-pl = Beweise ,
4306     case = A ,
4307         Name-sg = Beweis ,
4308         Name-pl = Beweise ,
4309     case = D ,
4310         Name-sg = Beweis ,
4311         Name-pl = Beweisen ,
4312     case = G ,
4313         Name-sg = Beweises ,
4314         Name-pl = Beweise ,
4315
4316 type = result ,
4317     gender = n ,
4318     case = N ,
4319         Name-sg = Ergebnis ,
4320         Name-pl = Ergebnisse ,
4321     case = A ,
4322         Name-sg = Ergebnis ,
4323         Name-pl = Ergebnisse ,
4324     case = D ,
4325         Name-sg = Ergebnis ,
4326         Name-pl = Ergebnissen ,
4327     case = G ,
4328         Name-sg = Ergebnisses ,
4329         Name-pl = Ergebnisse ,
4330
4331 type = remark ,
4332     gender = f ,
4333     case = N ,
4334         Name-sg = Bemerkung ,
4335         Name-pl = Bemerkungen ,
4336     case = A ,
4337         Name-sg = Bemerkung ,
4338         Name-pl = Bemerkungen ,
4339     case = D ,
4340         Name-sg = Bemerkung ,
4341         Name-pl = Bemerkungen ,
4342     case = G ,
4343         Name-sg = Bemerkung ,
4344         Name-pl = Bemerkungen ,
4345
4346 type = example ,
4347     gender = n ,

```

```

4348 case = N ,
4349     Name-sg = Beispiel ,
4350     Name-pl = Beispiele ,
4351 case = A ,
4352     Name-sg = Beispiel ,
4353     Name-pl = Beispiele ,
4354 case = D ,
4355     Name-sg = Beispiel ,
4356     Name-pl = Beispielen ,
4357 case = G ,
4358     Name-sg = Beispiels ,
4359     Name-pl = Beispiele ,
4360
4361 type = algorithm ,
4362     gender = m ,
4363     case = N ,
4364         Name-sg = Algorithmus ,
4365         Name-pl = Algorithmen ,
4366     case = A ,
4367         Name-sg = Algorithmus ,
4368         Name-pl = Algorithmen ,
4369     case = D ,
4370         Name-sg = Algorithmus ,
4371         Name-pl = Algorithmen ,
4372     case = G ,
4373         Name-sg = Algorithmus ,
4374         Name-pl = Algorithmen ,
4375
4376 type = listing ,
4377     gender = n ,
4378     case = N ,
4379         Name-sg = Listing ,
4380         Name-pl = Listings ,
4381     case = A ,
4382         Name-sg = Listing ,
4383         Name-pl = Listings ,
4384     case = D ,
4385         Name-sg = Listing ,
4386         Name-pl = Listings ,
4387     case = G ,
4388         Name-sg = Listings ,
4389         Name-pl = Listings ,
4390
4391 type = exercise ,
4392     gender = f ,
4393     case = N ,
4394         Name-sg = Übungsaufgabe ,
4395         Name-pl = Übungsaufgaben ,
4396     case = A ,
4397         Name-sg = Übungsaufgabe ,
4398         Name-pl = Übungsaufgaben ,
4399     case = D ,
4400         Name-sg = Übungsaufgabe ,
4401         Name-pl = Übungsaufgaben ,

```

```

4402 case = G ,
4403     Name-sg = Übungsaufgabe ,
4404     Name-pl = Übungsaufgaben ,
4405
4406 type = solution ,
4407     gender = f ,
4408     case = N ,
4409     Name-sg = Lösung ,
4410     Name-pl = Lösungen ,
4411     case = A ,
4412     Name-sg = Lösung ,
4413     Name-pl = Lösungen ,
4414     case = D ,
4415     Name-sg = Lösung ,
4416     Name-pl = Lösungen ,
4417     case = G ,
4418     Name-sg = Lösung ,
4419     Name-pl = Lösungen ,
4420 </lang-german>

```

10.3 French

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

```

4421 <*package>
4422 \zcDeclareLanguage [ gender = { f , m } ] { french }
4423 \zcDeclareLanguageAlias { acadian } { french }
4424 \zcDeclareLanguageAlias { canadien } { french }
4425 \zcDeclareLanguageAlias { francais } { french }
4426 \zcDeclareLanguageAlias { frenchb } { french }
4427 </package>
4428 <*lang-french>
4429 namesep = {\nobreakspace} ,
4430 pairsep = {\~et\nobreakspace} ,
4431 listsep = { , ~ } ,
4432 lastsep = {\~et\nobreakspace} ,
4433 tpairsep = {\~et\nobreakspace} ,
4434 tlistsep = { , ~ } ,
4435 tlastsep = {\~et\nobreakspace} ,
4436 notesep = { ~ } ,
4437 rangesep = {\~à\nobreakspace} ,
4438
4439 type = book ,
4440     gender = m ,
4441     Name-sg = Livre ,
4442     name-sg = livre ,
4443     Name-pl = Livres ,
4444     name-pl = livres ,
4445

```

```

4446 type = part ,
4447     gender = f ,
4448     Name-sg = Partie ,
4449     name-sg = partie ,
4450     Name-pl = Parties ,
4451     name-pl = parties ,
4452
4453 type = chapter ,
4454     gender = m ,
4455     Name-sg = Chapitre ,
4456     name-sg = chapitre ,
4457     Name-pl = Chapitres ,
4458     name-pl = chapitres ,
4459
4460 type = section ,
4461     gender = f ,
4462     Name-sg = Section ,
4463     name-sg = section ,
4464     Name-pl = Sections ,
4465     name-pl = sections ,
4466
4467 type = paragraph ,
4468     gender = m ,
4469     Name-sg = Paragraphe ,
4470     name-sg = paragraphe ,
4471     Name-pl = Paragraphes ,
4472     name-pl = paragraphes ,
4473
4474 type = appendix ,
4475     gender = f ,
4476     Name-sg = Annexe ,
4477     name-sg = annexe ,
4478     Name-pl = Annexes ,
4479     name-pl = annexes ,
4480
4481 type = page ,
4482     gender = f ,
4483     Name-sg = Page ,
4484     name-sg = page ,
4485     Name-pl = Pages ,
4486     name-pl = pages ,
4487     rangesep = {-} ,
4488
4489 type = line ,
4490     gender = f ,
4491     Name-sg = Ligne ,
4492     name-sg = ligne ,
4493     Name-pl = Lignes ,
4494     name-pl = lignes ,
4495
4496 type = figure ,
4497     gender = f ,
4498     Name-sg = Figure ,
4499     name-sg = figure ,

```



```

4500     Name-pl = Figures ,
4501     name-pl = figures ,
4502
4503     type = table ,
4504     gender = f ,
4505     Name-sg = Table ,
4506     name-sg = table ,
4507     Name-pl = Tables ,
4508     name-pl = tables ,
4509
4510     type = item ,
4511     gender = m ,
4512     Name-sg = Point ,
4513     name-sg = point ,
4514     Name-pl = Points ,
4515     name-pl = points ,
4516
4517     type = footnote ,
4518     gender = f ,
4519     Name-sg = Note ,
4520     name-sg = note ,
4521     Name-pl = Notes ,
4522     name-pl = notes ,
4523
4524     type = endnote ,
4525     gender = f ,
4526     Name-sg = Note ,
4527     name-sg = note ,
4528     Name-pl = Notes ,
4529     name-pl = notes ,
4530
4531     type = note ,
4532     gender = f ,
4533     Name-sg = Note ,
4534     name-sg = note ,
4535     Name-pl = Notes ,
4536     name-pl = notes ,
4537
4538     type = equation ,
4539     gender = f ,
4540     Name-sg = Équation ,
4541     name-sg = équation ,
4542     Name-pl = Équations ,
4543     name-pl = équations ,
4544     preref = {()} ,
4545     postref = {} } ,
4546
4547     type = theorem ,
4548     gender = m ,
4549     Name-sg = Théorème ,
4550     name-sg = théorème ,
4551     Name-pl = Théorèmes ,
4552     name-pl = théorèmes ,
4553

```

```

4554 type = lemma ,
4555     gender = m ,
4556     Name-sg = Lemme ,
4557     name-sg = lemme ,
4558     Name-pl = Lemmes ,
4559     name-pl = lemmes ,
4560
4561 type = corollary ,
4562     gender = m ,
4563     Name-sg = Corollaire ,
4564     name-sg = corollaire ,
4565     Name-pl = Corollaires ,
4566     name-pl = corollaires ,
4567
4568 type = proposition ,
4569     gender = f ,
4570     Name-sg = Proposition ,
4571     name-sg = proposition ,
4572     Name-pl = Propositions ,
4573     name-pl = propositions ,
4574
4575 type = definition ,
4576     gender = f ,
4577     Name-sg = Définition ,
4578     name-sg = définition ,
4579     Name-pl = Définitions ,
4580     name-pl = définitions ,
4581
4582 type = proof ,
4583     gender = f ,
4584     Name-sg = Démonstration ,
4585     name-sg = démonstration ,
4586     Name-pl = Démonstrations ,
4587     name-pl = démonstrations ,
4588
4589 type = result ,
4590     gender = m ,
4591     Name-sg = Résultat ,
4592     name-sg = résultat ,
4593     Name-pl = Résultats ,
4594     name-pl = résultats ,
4595
4596 type = remark ,
4597     gender = f ,
4598     Name-sg = Remarque ,
4599     name-sg = remarque ,
4600     Name-pl = Remarques ,
4601     name-pl = remarques ,
4602
4603 type = example ,
4604     gender = m ,
4605     Name-sg = Exemple ,
4606     name-sg = exemple ,
4607     Name-pl = Exemples ,

```

```

4608   name-pl = exemples ,
4609
4610   type = algorithm ,
4611   gender = m ,
4612   Name-sg = Algorithme ,
4613   name-sg = algorithme ,
4614   Name-pl = Algorithmes ,
4615   name-pl = algorithmes ,
4616
4617   type = listing ,
4618   gender = m ,
4619   Name-sg = Listing ,
4620   name-sg = listing ,
4621   Name-pl = Listings ,
4622   name-pl = listings ,
4623
4624   type = exercise ,
4625   gender = m ,
4626   Name-sg = Exercice ,
4627   name-sg = exercice ,
4628   Name-pl = Exercices ,
4629   name-pl = exercices ,
4630
4631   type = solution ,
4632   gender = f ,
4633   Name-sg = Solution ,
4634   name-sg = solution ,
4635   Name-pl = Solutions ,
4636   name-pl = solutions ,
4637 </lang-french>

```

10.4 Portuguese

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

```

4638 <*package>
4639 \zcDeclareLanguage [ gender = { f , m } ] { portuguese }
4640 \zcDeclareLanguageAlias { brazilian } { portuguese }
4641 \zcDeclareLanguageAlias { brazil   } { portuguese }
4642 \zcDeclareLanguageAlias { portuges } { portuguese }
4643 </package>
4644 <*lang-portuguese>
4645 namesep = {\nobreakspace} ,
4646 pairsep  = {\~e\nobreakspace} ,
4647 listsep  = { , ~ } ,
4648 lastsep  = {\~e\nobreakspace} ,
4649 tpairsep = {\~e\nobreakspace} ,
4650 tlistsep = { , ~ } ,
4651 tlastsep = {\~e\nobreakspace} ,
4652 notesep  = { ~ } ,
4653 rangesep  = {\~a\nobreakspace} ,

```

```

4654
4655 type = book ,
4656     gender = m ,
4657     Name-sg = Livro ,
4658     name-sg = livro ,
4659     Name-pl = Livros ,
4660     name-pl = livros ,
4661
4662 type = part ,
4663     gender = f ,
4664     Name-sg = Parte ,
4665     name-sg = parte ,
4666     Name-pl = Partes ,
4667     name-pl = partes ,
4668
4669 type = chapter ,
4670     gender = m ,
4671     Name-sg = Capítulo ,
4672     name-sg = capítulo ,
4673     Name-pl = Capítulos ,
4674     name-pl = capítulos ,
4675
4676 type = section ,
4677     gender = f ,
4678     Name-sg = Seção ,
4679     name-sg = seção ,
4680     Name-pl = Seções ,
4681     name-pl = seções ,
4682
4683 type = paragraph ,
4684     gender = m ,
4685     Name-sg = Parágrafo ,
4686     name-sg = parágrafo ,
4687     Name-pl = Parágrafos ,
4688     name-pl = parágrafos ,
4689     Name-sg-ab = Par. ,
4690     name-sg-ab = par. ,
4691     Name-pl-ab = Par. ,
4692     name-pl-ab = par. ,
4693
4694 type = appendix ,
4695     gender = m ,
4696     Name-sg = Apêndice ,
4697     name-sg = apêndice ,
4698     Name-pl = Apêndices ,
4699     name-pl = apêndices ,
4700
4701 type = page ,
4702     gender = f ,
4703     Name-sg = Página ,
4704     name-sg = página ,
4705     Name-pl = Páginas ,
4706     name-pl = páginas ,
4707     rangesep = {\textendash} ,

```

```

4708
4709 type = line ,
4710     gender = f ,
4711     Name-sg = Linha ,
4712     name-sg = linha ,
4713     Name-pl = Linhas ,
4714     name-pl = linhas ,
4715
4716 type = figure ,
4717     gender = f ,
4718     Name-sg = Figura ,
4719     name-sg = figura ,
4720     Name-pl = Figuras ,
4721     name-pl = figuras ,
4722     Name-sg-ab = Fig. ,
4723     name-sg-ab = fig. ,
4724     Name-pl-ab = Figs. ,
4725     name-pl-ab = figs. ,
4726
4727 type = table ,
4728     gender = f ,
4729     Name-sg = Tabela ,
4730     name-sg = tabela ,
4731     Name-pl = Tabelas ,
4732     name-pl = tabelas ,
4733
4734 type = item ,
4735     gender = m ,
4736     Name-sg = Item ,
4737     name-sg = item ,
4738     Name-pl = Itens ,
4739     name-pl = itens ,
4740
4741 type = footnote ,
4742     gender = f ,
4743     Name-sg = Nota ,
4744     name-sg = nota ,
4745     Name-pl = Notas ,
4746     name-pl = notas ,
4747
4748 type = endnote ,
4749     gender = f ,
4750     Name-sg = Nota ,
4751     name-sg = nota ,
4752     Name-pl = Notas ,
4753     name-pl = notas ,
4754
4755 type = note ,
4756     gender = f ,
4757     Name-sg = Nota ,
4758     name-sg = nota ,
4759     Name-pl = Notas ,
4760     name-pl = notas ,
4761

```

```

4762 type = equation ,
4763     gender = f ,
4764     Name-sg = Equação ,
4765     name-sg = equação ,
4766     Name-pl = Equações ,
4767     name-pl = equações ,
4768     Name-sg-ab = Eq. ,
4769     name-sg-ab = eq. ,
4770     Name-pl-ab = Eqs. ,
4771     name-pl-ab = eqs. ,
4772     preref = {()} ,
4773     postref = {} ,
4774
4775 type = theorem ,
4776     gender = m ,
4777     Name-sg = Teorema ,
4778     name-sg = teorema ,
4779     Name-pl = Teoremas ,
4780     name-pl = teoremas ,
4781
4782 type = lemma ,
4783     gender = m ,
4784     Name-sg = Lema ,
4785     name-sg = lema ,
4786     Name-pl = Lemas ,
4787     name-pl = lemas ,
4788
4789 type = corollary ,
4790     gender = m ,
4791     Name-sg = Corolário ,
4792     name-sg = corolário ,
4793     Name-pl = Corolários ,
4794     name-pl = corolários ,
4795
4796 type = proposition ,
4797     gender = f ,
4798     Name-sg = Proposição ,
4799     name-sg = proposição ,
4800     Name-pl = Proposições ,
4801     name-pl = proposições ,
4802
4803 type = definition ,
4804     gender = f ,
4805     Name-sg = Definição ,
4806     name-sg = definição ,
4807     Name-pl = Definições ,
4808     name-pl = definições ,
4809
4810 type = proof ,
4811     gender = f ,
4812     Name-sg = Demonstração ,
4813     name-sg = demonstração ,
4814     Name-pl = Demonstrações ,
4815     name-pl = demonstrações ,

```

```

4816
4817 type = result ,
4818     gender = m ,
4819     Name-sg = Resultado ,
4820     name-sg = resultado ,
4821     Name-pl = Resultados ,
4822     name-pl = resultados ,
4823
4824 type = remark ,
4825     gender = f ,
4826     Name-sg = Observação ,
4827     name-sg = observação ,
4828     Name-pl = Observações ,
4829     name-pl = observações ,
4830
4831 type = example ,
4832     gender = m ,
4833     Name-sg = Exemplo ,
4834     name-sg = exemplo ,
4835     Name-pl = Exemplos ,
4836     name-pl = exemplos ,
4837
4838 type = algorithm ,
4839     gender = m ,
4840     Name-sg = Algoritmo ,
4841     name-sg = algoritmo ,
4842     Name-pl = Algoritmos ,
4843     name-pl = algoritmos ,
4844
4845 type = listing ,
4846     gender = f ,
4847     Name-sg = Listagem ,
4848     name-sg = listagem ,
4849     Name-pl = Listagens ,
4850     name-pl = listagens ,
4851
4852 type = exercise ,
4853     gender = m ,
4854     Name-sg = Exercício ,
4855     name-sg = exercício ,
4856     Name-pl = Exercícios ,
4857     name-pl = exercícios ,
4858
4859 type = solution ,
4860     gender = f ,
4861     Name-sg = Solução ,
4862     name-sg = solução ,
4863     Name-pl = Soluções ,
4864     name-pl = soluções ,
4865 </lang-portuguese>

```

10.5 Spanish

Spanish language file has been initially provided by the author.

```
4866 <*package>
4867 \zcDeclareLanguage [ gender = { f , m } ] { spanish }
4868 </package>

4869 <*lang-spanish>

4870 namesep = {\nobreakspace} ,
4871 pairsep = {\~y\nobreakspace} ,
4872 listsep = { ,~ } ,
4873 lastsep = {\~y\nobreakspace} ,
4874 tpairsep = {\~y\nobreakspace} ,
4875 tlistsep = { ,~ } ,
4876 tlastsep = {\~y\nobreakspace} ,
4877 notesep = {~} ,
4878 rangesep = {\~a\nobreakspace} ,
4879
4880 type = book ,
4881   gender = m ,
4882   Name-sg = Libro ,
4883   name-sg = libro ,
4884   Name-pl = Libros ,
4885   name-pl = libros ,
4886
4887 type = part ,
4888   gender = f ,
4889   Name-sg = Parte ,
4890   name-sg = parte ,
4891   Name-pl = Partes ,
4892   name-pl = partes ,
4893
4894 type = chapter ,
4895   gender = m ,
4896   Name-sg = Capítulo ,
4897   name-sg = capítulo ,
4898   Name-pl = Capítulos ,
4899   name-pl = capítulos ,
4900
4901 type = section ,
4902   gender = f ,
4903   Name-sg = Sección ,
4904   name-sg = sección ,
4905   Name-pl = Secciones ,
4906   name-pl = secciones ,
4907
4908 type = paragraph ,
4909   gender = m ,
4910   Name-sg = Párrafo ,
4911   name-sg = párrafo ,
4912   Name-pl = Párrafos ,
4913   name-pl = párrafos ,
4914
4915 type = appendix ,
```



```

4916     gender = m ,
4917     Name-sg = Apéndice ,
4918     name-sg = apéndice ,
4919     Name-pl = Apéndices ,
4920     name-pl = apéndices ,
4921
4922     type = page ,
4923     gender = f ,
4924     Name-sg = Página ,
4925     name-sg = página ,
4926     Name-pl = Páginas ,
4927     name-pl = páginas ,
4928     rangesep = {\textendash} ,
4929
4930     type = line ,
4931     gender = f ,
4932     Name-sg = Línea ,
4933     name-sg = línea ,
4934     Name-pl = Líneas ,
4935     name-pl = líneas ,
4936
4937     type = figure ,
4938     gender = f ,
4939     Name-sg = Figura ,
4940     name-sg = figura ,
4941     Name-pl = Figuras ,
4942     name-pl = figuras ,
4943
4944     type = table ,
4945     gender = m ,
4946     Name-sg = Cuadro ,
4947     name-sg = cuadro ,
4948     Name-pl = Cuadros ,
4949     name-pl = cuadros ,
4950
4951     type = item ,
4952     gender = m ,
4953     Name-sg = Punto ,
4954     name-sg = punto ,
4955     Name-pl = Puntos ,
4956     name-pl = puntos ,
4957
4958     type = footnote ,
4959     gender = f ,
4960     Name-sg = Nota ,
4961     name-sg = nota ,
4962     Name-pl = Notas ,
4963     name-pl = notas ,
4964
4965     type = endnote ,
4966     gender = f ,
4967     Name-sg = Nota ,
4968     name-sg = nota ,
4969     Name-pl = Notas ,

```

```

4970     name-pl = notas ,
4971
4972 type = note ,
4973     gender = f ,
4974     Name-sg = Nota ,
4975     name-sg = nota ,
4976     Name-pl = Notas ,
4977     name-pl = notas ,
4978
4979 type = equation ,
4980     gender = f ,
4981     Name-sg = Ecuación ,
4982     name-sg = ecuación ,
4983     Name-pl = Ecuaciones ,
4984     name-pl = ecuaciones ,
4985     preref = {(} ,
4986     postref = {)} ,
4987
4988 type = theorem ,
4989     gender = m ,
4990     Name-sg = Teorema ,
4991     name-sg = teorema ,
4992     Name-pl = Teoremas ,
4993     name-pl = teoremas ,
4994
4995 type = lemma ,
4996     gender = m ,
4997     Name-sg = Lema ,
4998     name-sg = lema ,
4999     Name-pl = Lemas ,
5000     name-pl = lemas ,
5001
5002 type = corollary ,
5003     gender = m ,
5004     Name-sg = Corolario ,
5005     name-sg = corolario ,
5006     Name-pl = Corolarios ,
5007     name-pl = corolarios ,
5008
5009 type = proposition ,
5010     gender = f ,
5011     Name-sg = Proposición ,
5012     name-sg = proposición ,
5013     Name-pl = Proposiciones ,
5014     name-pl = proposiciones ,
5015
5016 type = definition ,
5017     gender = f ,
5018     Name-sg = Definición ,
5019     name-sg = definición ,
5020     Name-pl = Definiciones ,
5021     name-pl = definiciones ,
5022
5023 type = proof ,

```

```

5024     gender = f ,
5025     Name-sg = Demostración ,
5026     name-sg = demostración ,
5027     Name-pl = Demostraciones ,
5028     name-pl = demostraciones ,
5029
5030 type = result ,
5031     gender = m ,
5032     Name-sg = Resultado ,
5033     name-sg = resultado ,
5034     Name-pl = Resultados ,
5035     name-pl = resultados ,
5036
5037 type = remark ,
5038     gender = f ,
5039     Name-sg = Observación ,
5040     name-sg = observación ,
5041     Name-pl = Observaciones ,
5042     name-pl = observaciones ,
5043
5044 type = example ,
5045     gender = m ,
5046     Name-sg = Ejemplo ,
5047     name-sg = ejemplo ,
5048     Name-pl = Ejemplos ,
5049     name-pl = ejemplos ,
5050
5051 type = algorithm ,
5052     gender = m ,
5053     Name-sg = Algoritmo ,
5054     name-sg = algoritmo ,
5055     Name-pl = Algoritmos ,
5056     name-pl = algoritmos ,
5057
5058 type = listing ,
5059     gender = m ,
5060     Name-sg = Listado ,
5061     name-sg = listado ,
5062     Name-pl = Listados ,
5063     name-pl = listados ,
5064
5065 type = exercise ,
5066     gender = m ,
5067     Name-sg = Ejercicio ,
5068     name-sg = ejercicio ,
5069     Name-pl = Ejercicios ,
5070     name-pl = ejercicios ,
5071
5072 type = solution ,
5073     gender = f ,
5074     Name-sg = Solución ,
5075     name-sg = solución ,
5076     Name-pl = Soluciones ,
5077     name-pl = soluciones ,

```

5078 `</lang-spanish>`

10.6 Dutch

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

```
5079 <*package>
5080 \zcDeclareLanguage [ gender = { f , m , n } ] { dutch }
5081 </package>
5082 <*lang-dutch>

5083 namesep = {\nobreakspace} ,
5084 pairsep = {\~en\nobreakspace} ,
5085 listsep = {,~} ,
5086 lastsep = {\~en\nobreakspace} ,
5087 tpairsep = {\~en\nobreakspace} ,
5088 tlistsep = {,~} ,
5089 tlastsep = {,~en\nobreakspace} ,
5090 notesep = {~} ,
5091 rangesep = {\~t/m\nobreakspace} ,
5092
5093 type = book ,
5094 gender = n ,
5095 Name-sg = Boek ,
5096 name-sg = boek ,
5097 Name-pl = Boeken ,
5098 name-pl = boeken ,
5099
5100 type = part ,
5101 gender = n ,
5102 Name-sg = Deel ,
5103 name-sg = deel ,
5104 Name-pl = Delen ,
5105 name-pl = delen ,
5106
5107 type = chapter ,
5108 gender = n ,
5109 Name-sg = Hoofdstuk ,
5110 name-sg = hoofdstuk ,
5111 Name-pl = Hoofdstukken ,
5112 name-pl = hoofdstukken ,
5113
5114 type = section ,
5115 gender = m ,
5116 Name-sg = Paragraaf ,
5117 name-sg = paragraaf ,
5118 Name-pl = Paragrafen ,
5119 name-pl = paragrafen ,
5120
5121 type = paragraph ,
5122 gender = f ,
5123 Name-sg = Alinea ,
5124 name-sg = alinea ,
5125 Name-pl = Alinea's ,
```

```

5126     name-pl = alinea's ,
5127
5128     type = appendix ,
5129     gender = { m , n } ,
5130     Name-sg = Appendix ,
5131     name-sg = appendix ,
5132     Name-pl = Appendices ,
5133     name-pl = appendices ,
5134
5135     type = page ,
5136     gender = { f , m } ,
5137     Name-sg = Pagina ,
5138     name-sg = pagina ,
5139     Name-pl = Pagina's ,
5140     name-pl = pagina's ,
5141     rangesep = {\textendash} ,
5142
5143     type = line ,
5144     gender = m ,
5145     Name-sg = Regel ,
5146     name-sg = regel ,
5147     Name-pl = Regels ,
5148     name-pl = regels ,
5149
5150     type = figure ,
5151     gender = { n , f , m } ,
5152     Name-sg = Figuur ,
5153     name-sg = figuur ,
5154     Name-pl = Figuren ,
5155     name-pl = figuren ,
5156
5157     type = table ,
5158     gender = { f , m } ,
5159     Name-sg = Tabel ,
5160     name-sg = tabel ,
5161     Name-pl = Tabellen ,
5162     name-pl = tabellen ,
5163
5164     type = item ,
5165     gender = n ,
5166     Name-sg = Punt ,
5167     name-sg = punt ,
5168     Name-pl = Punten ,
5169     name-pl = punten ,
5170
5171     type = footnote ,
5172     gender = { f , m } ,
5173     Name-sg = Voetnoot ,
5174     name-sg = voetnoot ,
5175     Name-pl = Voetnoten ,
5176     name-pl = voetnoten ,
5177
5178     type = endnote ,
5179     gender = { f , m } ,

```

```

5180 Name-sg = Eindnoot ,
5181 name-sg = eindnoot ,
5182 Name-pl = Eindnoten ,
5183 name-pl = eindnoten ,
5184
5185 type = note ,
5186 gender = f ,
5187 Name-sg = Opmerking ,
5188 name-sg = opmerking ,
5189 Name-pl = Opmerkingen ,
5190 name-pl = opmerkingen ,
5191
5192 type = equation ,
5193 gender = f ,
5194 Name-sg = Vergelijking ,
5195 name-sg = vergelijking ,
5196 Name-pl = Vergelijkingen ,
5197 name-pl = vergelijkingen ,
5198 Name-sg-ab = Vgl. ,
5199 name-sg-ab = vgl. ,
5200 Name-pl-ab = Vgl.'s ,
5201 name-pl-ab = vgl.'s ,
5202 preref = {()} ,
5203 postref = {} ,
5204
5205 type = theorem ,
5206 gender = f ,
5207 Name-sg = Stelling ,
5208 name-sg = stelling ,
5209 Name-pl = Stellingen ,
5210 name-pl = stellingen ,

```

2022-01-09, niluxv: An alternative plural is “lemmata”. That is also a correct English plural for lemma, but the English language file chooses “lemmas”. For consistency we therefore choose “lemma’s”.

```

5211 type = lemma ,
5212 gender = n ,
5213 Name-sg = Lemma ,
5214 name-sg = lemma ,
5215 Name-pl = Lemma's ,
5216 name-pl = lemma's ,
5217
5218 type = corollary ,
5219 gender = n ,
5220 Name-sg = Gevolg ,
5221 name-sg = gevolg ,
5222 Name-pl = Gevolgen ,
5223 name-pl = gevolgen ,
5224
5225 type = proposition ,
5226 gender = f ,
5227 Name-sg = Propositie ,
5228 name-sg = propositie ,
5229 Name-pl = Proposities ,

```

```

5230     name-pl = proposities ,
5231
5232 type = definition ,
5233     gender = f ,
5234     Name-sg = Definitie ,
5235     name-sg = definitie ,
5236     Name-pl = Definities ,
5237     name-pl = definities ,
5238
5239 type = proof ,
5240     gender = n ,
5241     Name-sg = Bewijs ,
5242     name-sg = bewijs ,
5243     Name-pl = Bewijzen ,
5244     name-pl = bewijzen ,
5245
5246 type = result ,
5247     gender = n ,
5248     Name-sg = Resultaat ,
5249     name-sg = resultaat ,
5250     Name-pl = Resultaten ,
5251     name-pl = resultaten ,
5252
5253 type = remark ,
5254     gender = f ,
5255     Name-sg = Opmerking ,
5256     name-sg = opmerking ,
5257     Name-pl = Opmerkingen ,
5258     name-pl = opmerkingen ,
5259
5260 type = example ,
5261     gender = n ,
5262     Name-sg = Voorbeeld ,
5263     name-sg = voorbeeld ,
5264     Name-pl = Voorbeelden ,
5265     name-pl = voorbeelden ,
5266
5267 type = algorithm ,
5268     gender = { n , f , m } ,
5269     Name-sg = Algoritme ,
5270     name-sg = algoritme ,
5271     Name-pl = Algoritmes ,
5272     name-pl = algoritmes ,

```

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

```

5273 type = listing ,
5274     gender = m ,
5275     Name-sg = Listing ,
5276     name-sg = listing ,
5277     Name-pl = Listings ,
5278     name-pl = listings ,
5279
5280 type = exercise ,

```

```

5281 gender = { f , m } ,
5282 Name-sg = Opgave ,
5283 name-sg = opgave ,
5284 Name-pl = Opgaven ,
5285 name-pl = opgaven ,
5286
5287 type = solution ,
5288 gender = f ,
5289 Name-sg = Oplossing ,
5290 name-sg = oplossing ,
5291 Name-pl = Oplossingen ,
5292 name-pl = oplossingen ,
5293 </lang-dutch>

```

Index

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

A	
<code>\AddToHook</code>	100, 983, 1026, 1051, 1087, 1089, 1137, 1210, 1226, 1247, 1378, 1391, 1399, 3366, 3402, 3419, 3421, 3426, 3471, 3473, 3475, 3477, 3479, 3481, 3483, 3485, 3489, 3493, 3495, 3500, 3510, 3519, 3525, 3527, 3553, 3563, 3595, 3647, 3668
<code>\appendix</code>	2, 86, 87, 89
<code>\appendixname</code>	86
<code>\AtEndOfPackage</code>	1389
B	
<code>\babelname</code>	1036
<code>\babelprovide</code>	18, 31
<code>\bicaption</code>	89
<code>\bionenumcaption</code>	89
<code>\bitwonumcaption</code>	89
bool commands:	
<code>\bool_case_true:</code>	2
<code>\bool_gset_true:N</code>	1365
<code>\bool_if:NTF</code>	562, 573, 987, 991, 1401, 1864, 2238, 2313, 2450, 2474, 2493, 2500, 2505, 2552, 2556, 2608, 2632, 2636, 2642, 2652, 2658, 2779, 2786, 2793, 2796, 2817, 2846, 2849, 2884, 2897, 2905, 2908, 2964, 2966
<code>\bool_if:NTF</code> 68, 1976, 1986, 2010, 2027, 2042, 2107, 2115, 2486, 2773, 2878, 3285
<code>\bool_lazy_all:nTF</code>	2580, 2976
<code>\bool_lazy_and:nnTF</code> 1838, 1856, 2949, 2991, 3231, 3311
<code>\bool_lazy_any:nTF</code>	3111, 3120
<code>\bool_lazy_or:nnTF</code>	910, 936, 1471, 1504, 1842, 2937, 3348
<code>\bool_new:N</code>	511, 834, 835, 862, 886, 895, 902, 928, 960, 961, 1016, 1145, 1146, 1147, 1148, 1149, 1240, 1241, 1358, 1872, 1885, 2147, 2148, 2158, 2165, 2166, 2179, 2180, 2181, 3607
<code>\bool_set:Nn</code>	1835
<code>\bool_set_false:N</code>	847, 851, 968, 977, 978, 993, 1157, 1161, 1168, 1176, 1177, 1178, 1263, 1968, 2205, 2244, 2255, 2462, 2606, 2607, 3118, 3135
<code>\bool_set_true:N</code>	583, 841, 842, 846, 852, 967, 972, 973, 1155, 1162, 1167, 1184, 1186, 1188, 1191, 1192, 1193, 1251, 1256, 1982, 1992, 1996, 2018, 2033, 2048, 2072, 2213, 2239, 2245, 2249, 2256, 3134, 3170, 3177, 3178, 3196, 3203, 3215, 3614
<code>\bool_until_do:Nn</code>	2008, 2206
<code>\bool_while_do:nn</code>	3699
C	
<code>\caption</code>	91
clist commands:	
<code>\clist_clear:N</code>	645, 1675
<code>\clist_if_empty:NTF</code>	656, 1686
<code>\clist_if_in:NnTF</code>	3003
<code>\clist_map_inline:nn</code>	646, 1179, 1306, 1367, 1676, 3577, 3661
<code>\clist_put_right:Nn</code>	649, 1679
<code>\clist_set:Nn</code>	3001

\clist_use:Nn	659, 1691	2802, 2806, 2835, 2836, 2838, 2839,
\l_tmpa_clist	645, 649, 656,	2841, 2847, 2850, 2854, 2855, 2857,
	659, 1675, 1679, 1686, 1691, 3001, 3004	2865, 2869, 2870, 2872, 2885, 2898,
\contsubbottom	89	2900, 2906, 2909, 2912, 2914, 2919
\contsubcaption	89	\ExplSyntaxOn 18, 525
\counterwithin	5	
cs commands:		F
\cs_generate_variant:Nn	65, 276, 282, 579, 587,	file commands:
	594, 601, 1449, 1624, 1631, 2811, 3148	\file_get:nnNTF 523
\cs_if_exist:NTF	25,	\fmtversion 3
	28, 46, 49, 58, 78, 3390, 3396, 3517	\footnote 2, 89, 90
\cs_if_exist_p:N	3701	
\cs_if_exist_use:N	3488, 3739	G
\cs_new:Npn	56,	group commands:
	66, 76, 87, 277, 283, 2769, 2812, 3138	\group_begin: 102, 344,
\cs_new_eq:NN	3555, 3560	514, 582, 1582, 1832, 1846, 2507,
\cs_new_protected:Npn	271, 397, 512, 580, 588, 595,	2524, 2788, 2801, 2834, 2840, 2853,
	811, 1397, 1447, 1619, 1626, 1830,	2868, 2899, 2913, 3476, 3480, 3484
	1889, 1931, 1942, 1955, 2085, 2137,	\group_end: 105, 356,
	2182, 2320, 2602, 2765, 2767, 2926,	577, 585, 1616, 1849, 1869, 2521,
	3149, 3225, 3279, 3305, 3461, 3615	2527, 2792, 2805, 2837, 2845, 2856,
\cs_new_protected:Npx	99	2871, 2904, 2918, 3478, 3482, 3486
\cs_set:Npn	3464	
\cs_set_eq:NN	103, 3463, 3469, 3556, 3561	I
\cs_set_nopar:Npn	3546	\IfBooleanTF 1875
		\IfClassLoadedTF 112
E		\ifdraft 1160
\endinput	12	\IfFormatAtLeastTF 3, 4
\endsidecaption	88, 89	\ifoptionfinal 1166
exp commands:		\IfPackageLoadedTF 110
\exp_args:NNe	35, 38	\input 18
\exp_args:NNNo	273	int commands:
\exp_args:NNnx	365	\int_case:nnTF
\exp_args:NNo	273, 279	.. 2323, 2353, 2387, 2559, 2665, 2707
\exp_args:NNx	399, 403, 443, 529, 545, 1587, 1603	\int_compare:nNnTF 2019, 2034, 2049,
\exp_args:Nnx	590, 658	2061, 2073, 2093, 2095, 2139, 2286,
\exp_args:No	279	2340, 2376, 2544, 2546, 2619, 2645,
\exp_args:Nx	523, 3620, 3628	2692, 3166, 3172, 3192, 3198, 3717
\exp_args:Nxx	3162, 3184, 3188, 3205	\int_compare_p:nNn 2109,
\exp_not:N	73,	2117, 2584, 2941, 2952, 2981, 3131
	2507, 2510, 2521, 2524, 2527, 2782,	\int_eval:n 99
	2788, 2792, 2801, 2805, 2824, 2834,	\int_incr:N 2597, 2635, 2637,
	2837, 2840, 2845, 2853, 2856, 2868,	2651, 2653, 2657, 2659, 2763, 3715
	2871, 2887, 2899, 2904, 2913, 2918	\int_new:N
\exp_not:n	280, 2344, 2361, 2374, 2379, 2403,	.. 1886, 1887, 2149, 2150, 2162, 2163
	2418, 2422, 2435, 2439, 2475, 2476,	\int_set:Nn 2094, 2096, 2100, 2103, 3698
	2508, 2520, 2525, 2526, 2673, 2688,	\int_to_roman:n 3702, 3709, 3710, 3713
	2695, 2720, 2733, 2737, 2748, 2752,	\int_use:N 47, 50, 54, 60
	2780, 2787, 2789, 2794, 2797, 2800,	\int_zero:N
		.. 2087, 2088, 2191, 2192, 2193,
		2194, 2596, 2598, 2599, 2758, 2759
		\l_tmpa_int 3698,
		3702, 3709, 3710, 3713, 3715, 3717

iow commands:

- \iow_char:N 116, 136, 137, 142, 143, 148, 149, 154, 155, 207, 216, 217, 228
- \iow_newline: 268

J

- \jobname 3510

K

keys commands:

- \l_keys_choice_tl 672, 676, 689, 693, 1705, 1711, 1725, 1731
- \keys_define:nn 43, 373, 602, 703, 720, 734, 818, 836, 863, 872, 887, 896, 903, 929, 954, 962, 995, 1002, 1017, 1047, 1092, 1132, 1139, 1151, 1212, 1219, 1221, 1228, 1235, 1242, 1252, 1264, 1273, 1302, 1328, 1352, 1360, 1380, 1410, 1421, 1436, 1457, 1528, 1540, 1565, 1632, 1741, 1762, 1782, 1800
- \keys_set:nn 15, 18, 34, 35, 43, 51, 353, 555, 1257, 1448, 1455, 1613, 1833

keyval commands:

- \keyval_parse:nnn 1277, 1332
- \KOMAClassName 3517, 3539

L

- \label 89, 92, 95, 3463, 3469
- \labelformat 3
- \language name 30, 1030

M

- \mainbabelname 30, 1037
- \MessageBreak 10

MH commands:

- \MH_if_boolean:nTF 3612

msg commands:

- \msg_info:nnn 641, 711, 741, 3440, 3512, 3539, 3604, 3639, 3671, 3691, 3718
- \msg_info:nnnn 615, 622, 651
- \msg_info:nnnnn 635
- \msg_line_context: 115, 121, 125, 129, 132, 135, 141, 147, 153, 159, 164, 169, 174, 179, 185, 190, 193, 196, 201, 205, 211, 214, 221, 226, 234, 238, 246, 250, 252, 254, 257, 261, 268
- \msg_new:nnn 113, 119, 124, 126, 131, 133, 139, 145, 151, 157, 162, 167, 172, 177, 182, 187, 192, 194, 199, 204, 206, 208, 210, 212, 219, 224, 230, 232, 237, 239, 244, 249, 251, 253, 255, 260, 262, 264, 266
- \msg_note:nnn 558
- \msg_warning:nn 992, 998, 1231, 1268, 2586
- \msg_warning:nnn 348, 369, 564, 574, 824, 915, 941, 1075, 1118, 1135, 1197, 1208, 1334, 1384, 1394, 1483, 1516, 1532, 1615, 1671, 1753, 1807, 2270, 2457, 2970, 2986, 3271
- \msg_warning:nnnn 417, 434, 471, 494, 1279, 1426, 1432, 1570, 1576, 1645, 1652, 1681, 1787, 1793, 2278, 3050, 3098
- \msg_warning:nnnnn 457, 501, 1665, 3009
- \msg_warning:nnnnnn 3016

N

- \newcounter 5, 3386, 3387
- \NewDocumentCommand 342, 359, 1445, 1450, 1580, 1828, 1873
- \newfloat 89
- \newsfloat 89
- \nobreakspace 762, 3775, 3776, 3778, 3779, 3781, 3783, 3980, 3981, 3983, 3984, 3986, 3988, 4429, 4430, 4432, 4433, 4435, 4437, 4645, 4646, 4648, 4649, 4651, 4653, 4870, 4871, 4873, 4874, 4876, 4878, 5083, 5084, 5086, 5087, 5089, 5091

P

- \PackageError 7
- \pagenote 89, 90
- \pagenumbering 7
- \pageref 52
- \paragraph 37

prg commands:

- \prg_generate_conditional_variant:Nnn 784, 800
- \prg_new_conditional:Npnn 109, 111
- \prg_new_protected_conditional:Npnn 770, 786, 803
- \prg_return_false: 110, 112, 780, 782, 796, 798, 809
- \prg_return_true: 110, 112, 779, 795, 808

\ProcessKeysOptions 1444

prop commands:

- \prop_get:NnN 3248, 3328
- \prop_get:NnNTF 399, 516, 773, 776, 789, 792, 806, 1583, 3029, 3057, 3065, 3228, 3282, 3294, 3308
- \prop_gput:Nnn 350, 366, 377, 384, 391, 1621, 1628
- \prop_gput_if_new:Nnn 590, 597

\prop_gset_from_keyval:Nn	756
\prop_if_exist:NTF	1452
\prop_if_exist_p:N	3233, 3287, 3313
\prop_if_in:NnTF	35, 347, 363, 1072, 1115
\prop_if_in_p:Nn	69, 3240, 3320
\prop_item:Nn	38, 70, 367, 406, 446, 481, 532, 548, 1590, 1606
\prop_new:N	341, 351, 755, 1272, 1327, 1406, 1453
\prop_put:Nnn	489, 815, 913, 939, 1417, 1475, 1508, 1555
\prop_remove:Nn	814, 908, 934, 1416, 1463, 1496, 1547
\providecommand	3
\ProvidesExplPackage	14
\ProvidesFile	18
R	
\refstepcounter	3, 88, 89, 91, 93, 95, 96
\renewlist	97
\RequirePackage	16, 17, 18, 19, 20, 988, 1223, 1244
S	
\scantokens	87
seq commands:	
\seq_clear:N	883, 1891
\seq_const_from_clist:Nn	290, 298, 309, 321
\seq_gconcat:NNN	328, 331, 335, 338
\seq_get_left:NN	427, 438, 542, 624, 1600, 1654, 2216
\seq_gput_right:Nn	556, 567, 1371
\seq_gremove_all:Nn	1403
\seq_if_empty:NTF	413, 453, 539, 613, 633, 1597, 1643, 1663, 2210
\seq_if_in:NnTF	431, 468, 519, 619, 648, 1308, 1369, 1402, 1649, 1678, 1935
\seq_map_break:n	90, 2128, 2131
\seq_map_function:NN	1894
\seq_map_indexed_inline:Nn	27, 2089
\seq_map_inline:Nn	700, 717, 731, 1393, 1407, 1525, 1537, 1738, 1759, 1797, 2125, 3618
\seq_map_tokens:Nn	72
\seq_new:N	288, 289, 327, 334, 510, 871, 1301, 1359, 1871, 1888, 2146
\seq_pop_left:NN	2208
\seq_put_right:Nn	1310, 1938
\seq_reverse:N	877
\seq_set_eq:NN	2184
\seq_set_from_clist:Nn	403, 443, 529, 545, 876, 1587, 1603, 1834
\seq_sort:Nn	54, 1897
\setcounter	3388, 3389, 3405, 3420, 3424
\sidefootnote	89, 90
sort commands:	
\sort_return_same:	54, 59, 1904, 1909, 1983, 2003, 2024, 2039, 2053, 2078, 2113, 2128, 2144
\sort_return_swapped:	54, 59, 1917, 1993, 2002, 2023, 2038, 2054, 2077, 2121, 2131, 2143
\stepcounter	95, 3404, 3423
str commands:	
\str_case:nnTF	1053, 1096, 1181
\str_compare:nNnTF	1999
\str_if_eq:nnTF	89, 479
\str_if_eq_p:nn	911, 912, 937, 938, 1472, 1473, 1505, 1506, 3116, 3122, 3124, 3128, 3349, 3350
\str_new:N	1001
\str_set:Nn	1006, 1008, 1010, 1012
\string	3625, 3633
\subbottom	89
\subcaption	89
\subcaptionref	89
\subref	98
\subsection	37
\subsubsection	37
\subtop	89
T	
\tag	93, 95
T _E X and L ^A T _E X 2 _ε commands:	
\@@sidecaption	88
\@Alph	86
\@addtoreset	5
\@auxout	3624, 3632
\@bsphack	515, 3617
\@capttype	91, 3488, 3521, 3522, 3526, 3739
\@chapapp	86
\@currentcounter	2, 3, 5, 39, 89, 91, 93, 96, 28, 29, 49, 50, 1356, 3522, 3534
\@currentlabel	3, 89, 91, 96
\@currenvir	3529
\@elt	5
\@esphack	576, 3637
\@ifl@t@r	3
\@mem@escap@afterhook	89
\@memsubcaption	89
\@onlypreamble	358, 372, 1618
\@raw@opt@{package}.sty	35
\bbl@loaded	31
\bbl@main@language	30, 1031
\c@lstnumber	96

<code>\c@page</code>	7, 103	1669, 1701, 1721, 1746, 1767, 1805,
<code>\caption@subtypehook</code>	3740	1811, 1850, 2452, 2862, 2946, 2968,
<code>\hyper@@link</code> 73, 2510, 2782, 2824, 2887		3007, 3027, 3037, 3073, 3531, 3688
<code>\lst@AddToHook</code>	3687, 3689	<code>\tl_if_empty:nTF</code>
<code>\lst@Init</code>	96 345, 361, 606, 813, 907,
<code>\lst@label</code>	3688	933, 1364, 1461, 1494, 1636, 2399,
<code>\lst@MakeCaption</code>	96	2416, 2433, 2686, 2718, 2731, 2746
<code>\ltx@gobble</code>	92	<code>\tl_if_empty_p:N</code>
<code>\ltx@label</code> .. 92, 3555, 3556, 3560, 3561	 2979, 2993, 3115, 3125, 3129
<code>\m@mscaplabel</code>	88, 89	<code>\tl_if_empty_p:n</code>
<code>\MT@newlabel</code>	3625, 3633 1978, 1979,
<code>\protected@write</code>	3624, 3632	1988, 1989, 2014, 2015, 2030, 2045
<code>\zref@addprop</code> 22, 32, 43, 53, 55, 97, 108		<code>\tl_if_eq:NNTF</code> 1949, 1972, 2252, 3156
<code>\zref@default</code>	73, 2766, 2768	<code>\tl_if_eq:NnTF</code>
<code>\zref@extractdefault</code> 1892, 1924, 2099, 2102, 2127,
..... 11, 81, 274, 280, 284		2130, 2220, 2268, 2931, 3160, 3529
<code>\zref@ifpropdefined</code> .. 26, 822, 3140		<code>\tl_if_eq:nnTF</code>
<code>\zref@ifrefcontainsprop</code> 2091,
..... 26, 2273, 2771, 2819, 2874, 3143		3162, 3184, 3188, 3205, 3620, 3628
<code>\zref@ifrefundefined</code>		<code>\tl_if_exist:NnTF</code>
1899, 1901, 1913, 2241, 2243, 2248,	 3521
2265, 2454, 2463, 2610, 2814, 2928		<code>\tl_if_novalue:nTF</code>
<code>\zref@label</code>	92, 3549 1415, 1545
<code>\zref@localaddprop</code>	3490, 3741	<code>\tl_map_break:n</code>
<code>\ZREF@mainlist</code>	22, 90
32, 43, 53, 55, 97, 108, 3490, 3741		<code>\tl_map_tokens:Nn</code>
<code>\zref@newprop</code>	5, 7, 82
21, 23, 33, 44, 54, 92, 107, 3487, 3738		<code>\tl_new:N</code>
<code>\zref@refused</code>	2263 98, 285,
<code>\zref@wrapper@babel</code> 50, 92, 1829, 3549		286, 287, 817, 1023, 1024, 1025,
<code>\textendash</code> 766, 3830, 4094, 4707, 4928, 5141		1131, 1150, 1218, 1234, 1351, 1879,
<code>\textup</code>	94, 3601, 3602	1880, 1881, 1882, 1883, 1884, 2151,
<code>\thechapter</code>	86	2152, 2153, 2154, 2155, 2156, 2157,
<code>\thelstnumber</code>	96	2159, 2160, 2161, 2164, 2167, 2168,
<code>\thepage</code>	7, 104	2169, 2170, 2171, 2172, 2173, 2174,
<code>\thesection</code>	86	2175, 2176, 2177, 2178, 3491, 3524
tl commands:		<code>\tl_put_left:Nn</code> .. 2489, 2496, 2537,
<code>\c_empty_tl</code> 1934, 1958, 1961, 1964,		3039, 3040, 3075, 3077, 3079, 3081
1966, 3141, 3144, 3145, 3152, 3154		<code>\tl_put_right:Nn</code> .. 2342, 2359, 2369,
<code>\c_novalue_tl</code>	1412, 1542	2401, 2413, 2430, 2671, 2684, 2716,
<code>\tl_clear:N</code>		2729, 2744, 2947, 2948, 2959, 3740
.. 421, 462, 475, 497, 506, 528, 540,		<code>\tl_reverse:N</code>
607, 1586, 1598, 1637, 2186, 2187,	 1959, 1962
2188, 2189, 2190, 2212, 2592, 2593,		<code>\tl_set:Nn</code>
2594, 2595, 2634, 2929, 2933, 2961,	 273, 352, 608, 620, 825, 827,
3000, 3049, 3097, 3270, 3300, 3302		1030, 1031, 1036, 1037, 1040, 1041,
<code>\tl_gclear:N</code>	3537	1044, 1057, 1065, 1073, 1077, 1100,
<code>\tl_gset:Nn</code>	104	1108, 1116, 1120, 1454, 1638, 1650,
<code>\tl_gset_eq:NN</code>	3526	2066, 2068, 2222, 2223, 2330, 2332,
<code>\tl_head:N</code>		2472, 2503, 2623, 2625, 2649, 2943,
.. 2037, 2050, 2062, 2064, 2074, 2076		2944, 2957, 3351, 3354, 3492, 3494
<code>\tl_if_empty:NnTF</code>		<code>\tl_set_eq:NN</code>
.. 80, 415, 425, 455, 466, 492, 499,	 2590, 3522, 3533
639, 669, 686, 708, 725, 739, 745,		<code>\tl_show:N</code>
	 2553
		<code>\tl_tail:N</code>
	 2067, 2069
		<code>\l_tmpa_tl</code>
		.. 526, 555, 1852, 1853, 3308, 3333,
		3340, 3345, 3351, 3354, 3361, 3362
		U
<code>\upshape</code>	3600	
use commands:		
<code>\use:N</code>	26, 29, 3362	

V

\value 3405, 3424
\verbfootnote 89, 90

Z

\zcDeclareLanguage 13, 15, 51,
342, 3765, 3969, 4422, 4639, 4867, 5080
\zcDeclareLanguageAlias
..... 14, 359, 3766, 3767,
3768, 3769, 3770, 3771, 3772, 3972,
3973, 3974, 3975, 3976, 3977, 4423,
4424, 4425, 4426, 4640, 4641, 4642
\zcLanguageSetup
..... 12, 17, 18, 20, 42, 45, 46, 1580
\zcpageref 52, 1873
\zceref 32, 35, 40, 41,
50, 52–54, 60, 62, 94, 1828, 1876, 1877
\zcRefTypeSetup 12, 42, 1450, 3598
\zcsetup 31, 35, 40–42, 1445
\zlabel . 88, 89, 92, 93, 95, 96, 3467, 3688
zrefcheck commands:
 \zrefcheck_zceref_beg_label: .. 1841
 \zrefcheck_zceref_end_label_-
 maybe: 1860
 \zrefcheck_zceref_run_checks_on_-
 labels:n 1861
zrefclever internal commands:
 \l__zrefclever_abbrev_bool
 2167, 2309, 2950
 \l__zrefclever_base_language_tl .
 285, 352, 378, 385, 392, 401,
 409, 449, 484, 517, 521, 524, 535,
 551, 557, 559, 565, 568, 591, 598,
 616, 623, 636, 653, 774, 777, 790,
 793, 1584, 1593, 1609, 1646, 1653,
 1666, 1683, 1689, 1704, 1709, 1724,
 1729, 1749, 1770, 1775, 1814, 1820
 \l__zrefclever_capitalize_bool ..
 2167, 2307, 2938
 \l__zrefclever_capitalize_first_-
 bool 902, 924, 2940
 __zrefclever_compat_module:nn ..
 40,
 1397, 3364, 3382, 3443, 3515, 3542,
 3608, 3643, 3674, 3694, 3721, 3744
 __zrefclever_counter_reset_by:n
 6, 38, 58, 60, 62, 66, 3570, 3654
 __zrefclever_counter_reset_by_-
 aux:nn 73, 76
 __zrefclever_counter_reset_by_-
 aux:nnn 83, 87
 \l__zrefclever_counter_resetby_-
 prop 5, 38, 69, 70, 1327, 1339
 \l__zrefclever_counter_resettters_-
 seq 5, 38, 72, 1301, 1308, 1311
 \l__zrefclever_counter_type_prop
 4, 37, 35, 38, 1272, 1284
 \l__zrefclever_current_counter_-
 tl 3, 5, 39, 21, 25,
 26, 36, 39, 41, 46, 47, 95, 1351, 1354
 \l__zrefclever_current_language_-
 tl 30, 31, 1024, 1030, 1036,
 1040, 1045, 1058, 1078, 1101, 1121
 __zrefclever_declare_lang_opt_-
 default:nnn
 46, 1619, 1703, 1723, 1748, 1769
 __zrefclever_declare_lang_opt_-
 type:nnnn 46, 1619,
 1688, 1708, 1728, 1774, 1813, 1819
 __zrefclever_extract:nnn
 11, 283, 2020, 2022,
 2035, 2052, 2140, 2142, 3167, 3169,
 3173, 3175, 3193, 3195, 3199, 3201
 __zrefclever_extract_default:Nnnn
 11,
 271, 1933, 1944, 1946, 1957, 1960,
 1963, 1965, 2226, 2229, 3151, 3153
 __zrefclever_extract_unexp:nnn .
 11, 81, 277, 2516, 2784, 2790, 2803,
 2830, 2842, 2893, 2901, 2915, 3141,
 3144, 3145, 3163, 3164, 3185, 3186,
 3189, 3190, 3207, 3211, 3621, 3629
 __zrefclever_extract_url_-
 unexp:n 2512, 2783, 2826, 2889, 3138
 \g__zrefclever_fallback_unknown_-
 lang_prop 11, 755, 756, 806
 __zrefclever_get_enclosing_-
 counters_value:n . 5, 6, 56, 61, 94
 __zrefclever_get_fallback_-
 unknown_lang_opt:nN 804
 __zrefclever_get_fallback_-
 unknown_lang_opt:nNTF 25, 802, 3268
 __zrefclever_get_lang_opt_-
 default:nnN 12, 787, 801
 __zrefclever_get_lang_opt_-
 default:nnNTF . 25, 786, 3263, 3343
 __zrefclever_get_lang_opt_-
 type:nnnN 12, 771, 785
 __zrefclever_get_lang_opt_-
 type:nnnNTF 24,
 770, 2995, 3043, 3085, 3091, 3257, 3337
 __zrefclever_get_ref:n
 73, 2345, 2362,
 2375, 2380, 2404, 2419, 2423, 2436,
 2440, 2477, 2497, 2674, 2689, 2696,
 2721, 2734, 2738, 2749, 2753, 2769

`__zrefclever_get_ref_first:` ... 2333, 2626, 3236, 3243, 3251, 3259, 3290, 3297, 3316, 3323, 3331, 3339
 73, 77, 2490, 2538, 2812
`__zrefclever_get_ref_opt_-`
 `bool:nN` 83, 85
`__zrefclever_get_ref_opt_-`
 `bool:nnN` 2306, 2308, 3305
`__zrefclever_get_ref_opt_-`
 `font:nN` 11, 24, 41, 83, 84, 2302, 2304, 3279
`__zrefclever_get_ref_opt_-`
 `typeset:nN` . 11, 12, 24, 41, 83, 1852, 2197, 2199, 2201, 2288, 2290, 2292, 2294, 2296, 2298, 2300, 3225
`__zrefclever_if_class_loaded:n` 109
`__zrefclever_if_class_loaded:nTF` 3445
`__zrefclever_if_package_-`
 `loaded:n` 109
`__zrefclever_if_package_-`
 `loaded:nTF` 985, 1028, 1034, 1249, 3384, 3544, 3551, 3610, 3645, 3676, 3696, 3723, 3746
`\g__zrefclever_koma_captionofbeside_-`
 `capttype_tl` 3524
`\g__zrefclever_koma_capttype_tl` 3526, 3531, 3534, 3537
`\l__zrefclever_label_a_tl` 59, 2151, 2209, 2228, 2241, 2263, 2265, 2271, 2274, 2280, 2331, 2345, 2362, 2380, 2423, 2440, 2468, 2477, 2610, 2614, 2624, 2650, 2674, 2697, 2738, 2753
`\l__zrefclever_label_b_tl` 59, 2151, 2212, 2217, 2231, 2243, 2248, 2614
`\l__zrefclever_label_count_int` 60, 2149, 2191, 2286, 2323, 2596, 2619, 2763, 2982
`\l__zrefclever_label_enclval_a_-`
 `tl` 1879, 1957, 1959, 2014, 2030, 2050, 2062, 2066, 2067, 2074
`\l__zrefclever_label_enclval_b_-`
 `tl` 1879, 1960, 1962, 2015, 2037, 2045, 2064, 2068, 2069, 2076
`\l__zrefclever_label_extdoc_a_tl` 1879, 1963, 1973, 1978, 1988, 2001, 3151, 3157
`\l__zrefclever_label_extdoc_b_tl` 1879, 1965, 1974, 1979, 1989, 2000, 3153, 3158
`\l__zrefclever_label_type_a_tl` 83, 1879, 1934, 1936, 1939, 1945, 1950, 2099, 2127, 2222, 2227, 2253, 2268, 2333, 2626, 3236, 3243, 3251, 3259, 3290, 3297, 3316, 3323, 3331, 3339
`\l__zrefclever_label_type_b_tl` 1879, 1947, 1951, 2102, 2130, 2223, 2230, 2254
`__zrefclever_label_type_put_-`
 `new_right:n` 53, 54, 1895, 1931
`\l__zrefclever_label_types_seq` 54, 1888, 1891, 1935, 1938, 2125
`__zrefclever_labels_in_sequence:nn` 60, 81, 2466, 2613, 3149
`\g__zrefclever_lang_(language)_prop` 18
`\l__zrefclever_lang_decl_case_tl` 285, 540, 543, 620, 625, 745, 749, 1598, 1601, 1650, 1655, 1811, 1822
`\l__zrefclever_lang_declension_-`
 `seq` 285, 404, 413, 427, 431, 438, 530, 539, 542, 613, 619, 624, 1588, 1597, 1600, 1643, 1649, 1654
`\l__zrefclever_lang_gender_seq` 285, 444, 453, 468, 546, 633, 648, 1604, 1663, 1678
`\g__zrefclever_languages_prop` 14, 341, 347, 350, 363, 366, 367, 399, 516, 773, 789, 1072, 1115, 1583
`\l__zrefclever_last_of_type_bool` 60, 2146, 2239, 2244, 2245, 2249, 2255, 2256, 2313
`\l__zrefclever_lastsep_tl` . 2167, 2295, 2361, 2379, 2403, 2422, 2435
`\l__zrefclever_link_star_bool` 1835, 1871, 2776, 2881, 3114
`\l__zrefclever_listsep_tl` 2167, 2293, 2374, 2418, 2673, 2688, 2695, 2720, 2733, 2737, 2748
`\l__zrefclever_load_langfile_-`
 `verbose_bool` 511, 562, 573, 583
`\g__zrefclever_loaded_langfiles_-`
 `seq` 510, 520, 556, 567
`__zrefclever_ltxlabel:n` 92, 3546, 3556, 3561
`\l__zrefclever_main_language_tl` 30, 31, 1025, 1031, 1037, 1041, 1066, 1109
`__zrefclever_mathtools_showonlyrefs:n` 1866, 3615
`\l__zrefclever_mathtools_-`
 `showonlyrefs_bool` 1864, 3607, 3614
`__zrefclever_memoir_both_-`
 `labels:` 3461, 3472, 3474, 3476, 3480, 3484
`\l__zrefclever_memoir_footnote_-`
 `type_tl` 3491, 3492, 3494, 3498

<code>__zrefclever_memoir_label_and-</code>	<code>\g__zrefclever_page_format_tl ...</code>
<code>zlabel:n</code> 3464, 3469 7, 98, 104, 107
<code>__zrefclever_memoir_orig-</code>	<code>\l__zrefclever_pairsep_tl</code>
<code>label:n</code> 3463, 3466 2167, 2291, 2344, 2475
<code>__zrefclever_name_default:</code>	<code>\l__zrefclever_postref_tl</code>
..... 2765, 2864 2167, 2301, 2794, 2797,
<code>\l__zrefclever_name_format-</code>	2806, 2847, 2850, 2906, 2909, 2919
<code>fallback_tl</code> 2157, 2957,	<code>\l__zrefclever_preposinlink_bool</code>
2961, 3027, 3070, 3080, 3082, 3094	1016, 1019, 2779, 2786, 2793, 2796,
<code>\l__zrefclever_name_format_tl</code> ...	2846, 2849, 2884, 2897, 2905, 2908
... 2157, 2943, 2944, 2947, 2948,	<code>\l__zrefclever_preref_tl</code>
2958, 2959, 3034, 3039, 3040, 3046, 2167, 2299, 2780,
3051, 3062, 3076, 3077, 3088, 3100	2787, 2800, 2839, 2885, 2898, 2912
<code>\l__zrefclever_name_in_link_bool</code>	<code>__zrefclever_process_language-</code>
..... 74,	<code>options:</code> 34, 35, 397, 1837
77, 2157, 2505, 2817, 3118, 3134, 3135	<code>__zrefclever_prop_put_non-</code>
<code>\l__zrefclever_namefont_tl</code> 2167,	<code>empty:Nnn</code> 26, 811, 1283, 1338
2303, 2508, 2525, 2835, 2854, 2869	<code>__zrefclever_provide_lang_opt-</code>
<code>\l__zrefclever_nameinlink_str</code> ...	<code>default:nn</code> 20, 588, 671, 688, 709, 726
..... 1001, 1006, 1008,	<code>__zrefclever_provide_lang_opt-</code>
1010, 1012, 3116, 3122, 3124, 3128	<code>type:nn</code> 20,
<code>\l__zrefclever_namesep_tl</code>	588, 658, 675, 692, 727, 746, 748
... 2167, 2289, 2838, 2857, 2865, 2872	<code>__zrefclever_provide_langfile:n</code>
<code>\l__zrefclever_next_is_same_bool</code> 12, 18, 20, 21, 51,
..... 60, 81, 2162,	512, 584, 1091, 1102, 1110, 1123, 1836
2607, 2636, 2652, 2658, 3178, 3216	<code>__zrefclever_provide_langfile-</code>
<code>\l__zrefclever_next_maybe_range-</code>	<code>verbose:n</code> 20, 580, 1059, 1067, 1080
<code>bool</code>	<code>\l__zrefclever_range_beg_label-</code>
... 60, 81, 2162, 2462, 2474, 2606,	<code>tl</code> 60, 2162, 2190,
2632, 2642, 3170, 3177, 3196, 3204	2375, 2399, 2405, 2416, 2420, 2433,
<code>\l__zrefclever_noabbrev_first-</code>	2437, 2595, 2634, 2649, 2686, 2690,
<code>bool</code> 928, 950, 2954	2718, 2722, 2731, 2735, 2746, 2750
<code>\g__zrefclever_nocompat_bool</code> ...	<code>\l__zrefclever_range_count_int</code> ..
..... 1358, 1365, 1401 60,
<code>\l__zrefclever_nocompat_bool</code> ... 40	2162, 2193, 2353, 2389, 2598, 2635,
<code>\g__zrefclever_nocompat_modules-</code>	2646, 2651, 2657, 2665, 2709, 2758
<code>seq</code> 1359, 1369, 1372, 1393, 1402, 1403	<code>\l__zrefclever_range_same_count-</code>
<code>\l__zrefclever_nocompat_modules-</code>	<code>int</code> 60,
<code>seq</code> 40	2162, 2194, 2340, 2377, 2390, 2599,
<code>\l__zrefclever_nudge_comptosing-</code>	2637, 2653, 2659, 2693, 2710, 2759
<code>bool</code> ... 1147, 1177, 1186, 1192, 2978	<code>\l__zrefclever_rangesep_tl</code>
<code>\l__zrefclever_nudge_enabled-</code> 2167, 2297, 2439, 2476, 2752
<code>bool</code> 1145, 1155, 1157,	<code>\l__zrefclever_ref_decl_case_tl</code> .
1161, 1162, 1167, 1168, 2582, 2964 15, 415, 420, 421, 425, 428,
<code>\l__zrefclever_nudge_gender_bool</code>	432, 436, 439, 492, 495, 497, 1131,
..... 1149, 1178, 1188, 1193, 2992	1141, 3037, 3041, 3073, 3078, 3083
<code>\l__zrefclever_nudge_multitype-</code>	<code>__zrefclever_ref_default:</code>
<code>bool</code> ... 1146, 1176, 1184, 1191, 2583 2765, 2809, 2815, 2858, 2922
<code>\l__zrefclever_nudge_singular-</code>	<code>\l__zrefclever_ref_gender_tl</code> ...
<code>bool</code> 1148, 1204, 2966 16, 455, 461,
<code>__zrefclever_orig_ltxlabel:n</code> ...	462, 466, 469, 474, 475, 499, 505,
..... 3548, 3555, 3560	506, 1150, 1214, 2993, 3005, 3011, 3019
<code>__zrefclever_page_format_aux:</code> ..	<code>\l__zrefclever_ref_language_tl</code> ..
..... 99, 103	... 15, 31, 400, 419, 437, 459, 473,

496, 503, 1023, 1044, 1057, 1060,
 1065, 1068, 1073, 1077, 1081, 1091,
 1100, 1103, 1108, 1111, 1116, 1120,
 1124, 1836, 2996, 3013, 3021, 3044,
 3086, 3092, 3258, 3264, 3338, 3344
 \c_zrefclever_ref_options_font_-
 seq 12, 24, 290
 \c_zrefclever_ref_options_-
 necessarily_not_type_specific_-
 seq 24, 290, 701, 1526, 1739
 \c_zrefclever_ref_options_-
 possibly_type_specific_seq ..
 24, 290, 718, 1760
 \l_zrefclever_ref_options_prop .
 ... 17, 40–42, 489, 908, 913, 934,
 939, 1406, 1416, 1417, 3228, 3282, 3308
 \c_zrefclever_ref_options_-
 reference_seq 290, 1408
 \c_zrefclever_ref_options_type_-
 names_seq 290, 732, 1798
 \c_zrefclever_ref_options_-
 typesetup_seq 290, 1538
 \l_zrefclever_ref_property_tl ..
 26, 817, 825, 827, 1892, 1924, 2220,
 2275, 2279, 2771, 2821, 2876, 3160
 \l_zrefclever_ref_typeset_font_-
 tl 1218, 1220, 1847
 \l_zrefclever_reffont_tl . 2167,
 2305, 2789, 2802, 2841, 2900, 2914
 \l_zrefclever_setup_type_tl ...
 20, 285, 528,
 592, 607, 608, 639, 669, 686, 708,
 725, 739, 1454, 1466, 1478, 1499,
 1511, 1550, 1558, 1586, 1637, 1638,
 1669, 1690, 1701, 1710, 1721, 1730,
 1746, 1767, 1776, 1805, 1815, 1821
 \l_zrefclever_sort_decided_bool
 1885, 1968, 1982, 1992,
 1996, 2008, 2018, 2033, 2048, 2072
 _zrefclever_sort_default:nn ...
 55, 1926, 1942
 _zrefclever_sort_default_-
 different_types:nn
 27, 53, 57, 1953, 2085
 _zrefclever_sort_default_same_-
 type:nn 53, 55, 1952, 1955
 _zrefclever_sort_labels:
 53, 54, 58, 1845, 1889
 _zrefclever_sort_page:nn
 59, 1925, 2137
 \l_zrefclever_sort_prior_a_int .
 1886,
 2087, 2093, 2094, 2100, 2110, 2118
 \l_zrefclever_sort_prior_b_int .
 1886,
 2088, 2095, 2096, 2103, 2111, 2119
 \l_zrefclever_tlastsep_tl
 2167, 2202, 2576
 \l_zrefclever_tlistsep_tl
 2167, 2200, 2547
 \l_zrefclever_tpairsep_tl
 2167, 2198, 2569
 \l_zrefclever_type_<type>-
 options_prop 42
 \l_zrefclever_type_count_int ...
 ... 60, 77, 2149, 2192, 2544, 2546,
 2559, 2584, 2597, 2941, 2953, 3131
 \l_zrefclever_type_first_label_-
 tl 60, 74, 2151, 2188, 2330, 2454,
 2463, 2467, 2497, 2513, 2517, 2593,
 2623, 2814, 2820, 2827, 2831, 2843,
 2875, 2890, 2894, 2902, 2916, 2928
 \l_zrefclever_type_first_label_-
 type_tl ... 60, 77, 2151, 2189,
 2332, 2458, 2594, 2625, 2932, 2972,
 2988, 2997, 3012, 3018, 3032, 3045,
 3052, 3060, 3068, 3087, 3093, 3101
 \l_zrefclever_type_name_gender_-
 tl 2157, 2999, 3000, 3002, 3007, 3020
 _zrefclever_type_name_setup: ..
 11, 12, 74, 2485, 2926
 \l_zrefclever_type_name_tl
 74, 77,
 2157, 2520, 2526, 2836, 2855, 2862,
 2870, 2929, 2933, 3035, 3047, 3049,
 3063, 3071, 3089, 3095, 3097, 3115
 \l_zrefclever_typeset_compress_-
 bool 886, 889, 2608
 \l_zrefclever_typeset_labels_-
 seq 59, 2146, 2184, 2208, 2210, 2216
 \l_zrefclever_typeset_last_bool
 60, 2146,
 2205, 2206, 2213, 2238, 2556, 3130
 \l_zrefclever_typeset_name_bool
 835, 842, 847, 852, 2487, 2500
 \l_zrefclever_typeset_queue_-
 curr_tl 60, 61, 73, 77,
 2151, 2187, 2342, 2359, 2369, 2401,
 2413, 2430, 2452, 2472, 2489, 2496,
 2503, 2537, 2553, 2564, 2570, 2577,
 2591, 2592, 2671, 2684, 2716, 2729,
 2744, 2946, 2968, 2979, 3125, 3129
 \l_zrefclever_typeset_queue_-
 prev_tl . 60, 2151, 2186, 2548, 2590
 \l_zrefclever_typeset_range_-
 bool 895, 898, 1844, 2450

\l_zrefclever_typeset_ref_bool .	\l_zrefclever_warn_hyperref_
..... 834, 841, 846, 851, 2487, 2493	bool 961, 968, 973, 978, 991
_zrefclever_typeset_refs:	_zrefclever_zcref:nnn 15, 1829, 1830
..... 59–61, 1848, 2182	_zrefclever_zcref:nnnn 50, 53, 1830
_zrefclever_typeset_refs_last_	\l_zrefclever_zcref_labels_seq .
of_type: . 64, 73, 74, 77, 2315, 2320 53, 54, 1834,
_zrefclever_typeset_refs_not_	1862, 1867, 1871, 1894, 1897, 2185
last_of_type:	\l_zrefclever_zcref_note_tl ...
..... 60, 64, 73, 81, 2317, 2602 1234, 1237, 1850, 1854
\l_zrefclever_typeset_sort_bool	\l_zrefclever_zcref_with_check_
..... 862, 865, 1843	bool 1241, 1256, 1840, 1858
\l_zrefclever_typesort_seq	_zrefclever_zcsetup:n
.... 27, 57, 871, 876, 877, 883, 2089 42, 1446, 1447, 3368,
\l_zrefclever_use_hyperref_bool	3392, 3398, 3406, 3428, 3447, 3497,
..... 960, 967,	3501, 3502, 3511, 3565, 3596, 3649,
972, 977, 987, 993, 2775, 2880, 3113	3669, 3678, 3690, 3705, 3725, 3748
\l_zrefclever_verbose_testing_	\l_zrefclever_zrefcheck_
bool 2181, 2552	available_bool
 1240, 1251, 1263, 1839, 1857