

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-13

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	Reference format	8
4.3	Languages	10
4.4	Dictionaries	11
4.5	Options	14
5	Configuration	25
5.1	\zcsetup	25
5.2	\zcRefTypeSetup	25
5.3	\zcDeclareTranslations	27
6	User interface	29
6.1	\zceref	29
6.2	\zcpageref	31
7	Sorting	31
8	Typesetting	38
9	Special handling	59
9.1	\appendix	59
9.2	\newtheorem	59
9.3	enumitem package	59

*This file describes v0.1.0-alpha, released 2021-09-13.

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

10	Dictionaries	60
10.1	English	60
10.2	German	63
10.3	French	67
10.4	Portuguese	70
10.5	Spanish	73
Index		77

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

Taking a stance on backward compatibility of the package. During initial development, we have used freely recent features of the kernel (albeit refraining from `l3candidates`, even though I'd have loved to have used `\bool_case_true:...`). We presume `xparse` (which made to the kernel in the 2020-10-01 release), and `expl3` as well (which made to the kernel in the 2020-02-02 release). We also just use UTF-8 for the dictionaries (which became the default input encoding in the 2018-04-01 release). Hence, since we would not be able to go much backwards without special handling anyway, we make the cut with the inclusion of the new hook management system (`ltxcmdhooks`), which is bound to be useful for our purposes, and was released with the 2021-06-01 kernel.

```
3 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
4 \IfFormatAtLeastTF{2021-06-01}
5 {}
6 {%
7   \PackageError{zref-clever}{LaTeX kernel too old}
8   {%
9     'zref-clever' requires a LaTeX kernel newer than 2021-06-01.%
10    \MessageBreak Loading will abort!%
11   }%
12 \endinput
13 }%
```

Identify the package.

```
14 \ProvidesExplPackage {zref-clever} {2021-09-13} {0.1.0-alpha}
15 {Clever LaTeX cross-references based on zref}
```

2 Dependencies

Required packages. Besides these, `zref-hyperref` may also be required depending on the presence of `hyperref` itself and on the `hyperref` option.

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

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 `page` and `counter` properties are respectively provided by modules `zref-base` and `zref-counter`. The `zref-abspace` provides the `abspace` property which gives us a safe and easy way to sort labels for page references.

But 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 `zc@thecnt` for reserved use. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@...` prefix.

```
21 \zref@newprop { zc@thecnt } { \use:c { the \@currentcounter } }
22 \zref@addprop \ZREF@mainlist { zc@thecnt }
```

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`.

```
23 \zref@newprop { zc@type }
24 {
25   \prop_if_in:NVTF \l__zrefclever_counter_type_prop \@currentcounter
26   {
27     \exp_args:NNe \prop_item:Nn
28     \l__zrefclever_counter_type_prop { \@currentcounter }
29   }
30   { \@currentcounter }
31 }
32 \zref@addprop \ZREF@mainlist { zc@type }
```

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

```
33 \zref@newprop { zc@cntval } [0] { \int_use:c { c@ \@currentcounter } }
34 \zref@addprop \ZREF@mainlist { zc@cntval }
35 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
36 \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 (the counters’ names and values). Indeed, the set

of counters grouped into a single type cannot be arbitrary: all of them must belong to the same reset chain, and must be nested within each other (they cannot even just share the same parent).

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”, “supercounter” etc.). This information is a little trickier 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 section ‘ltxcounts.dtx’ in ‘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 included 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 (`\@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:n
__zrefclever_get_enclosing_counters_value:n
```

Recursively generate a *sequence* of “enclosing counters” and values, for a given `<counter>` and leave it in the input stream. These functions must be expandable, since they get called from `\zref@newprop` and are the ones responsible for generating the desired information when the label is being set. Note that the order in which we are getting this information is reversed, since we are navigating the counter reset chain bottom-up. But

it is very hard to do otherwise here where we need expandable functions, and easy to handle at the reading side.

```

    \_zrefclever_get_enclosing_counters:n {\counter}
    \_zrefclever_get_enclosing_counters_value:n {\counter}

37 \cs_new:Npn \_zrefclever_get_enclosing_counters:n #1
38 {
39   \cs_if_exist:cT { c@ \_zrefclever_counter_reset_by:n {#1} }
40   {
41     { \_zrefclever_counter_reset_by:n {#1} }
42     \_zrefclever_get_enclosing_counters:e
43     { \_zrefclever_counter_reset_by:n {#1} }
44   }
45 }
46 \cs_new:Npn \_zrefclever_get_enclosing_counters_value:n #1
47 {
48   \cs_if_exist:cT { c@ \_zrefclever_counter_reset_by:n {#1} }
49   {
50     { \int_use:c { c@ \_zrefclever_counter_reset_by:n {#1} } }
51     \_zrefclever_get_enclosing_counters_value:e
52     { \_zrefclever_counter_reset_by:n {#1} }
53   }
54 }

```

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 (see also https://tex.stackexchange.com/q/611370/#comment1529282_611385, thanks Enrico Gregorio, aka 'egreg').

```

55 \cs_generate_variant:Nn \_zrefclever_get_enclosing_counters:n { V , e }
56 \cs_generate_variant:Nn \_zrefclever_get_enclosing_counters_value:n { V , e }

```

(End definition for _zrefclever_get_enclosing_counters:n and _zrefclever_get_enclosing_counters_value:n.)

_zrefclever_counter_reset_by:n Auxiliary function for _zrefclever_get_enclosing_counters:n and _zrefclever_get_enclosing_counters_value:n. They are 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 `\counter`.

```

    \_zrefclever_counter_reset_by:n {\counter}

57 \cs_new:Npn \_zrefclever_counter_reset_by:n #1
58 {
59   \bool_if:nTF
60   { \prop_if_in_p:Nn \l_zrefclever_counter_resetby_prop {#1} }
61   { \prop_item:Nn \l_zrefclever_counter_resetby_prop {#1} }
62   {
63     \seq_map_tokens:Nn \l_zrefclever_counter_resettters_seq
64     { \_zrefclever_counter_reset_by_aux:nn {#1} }
65   }
66 }
67 \cs_new:Npn \_zrefclever_counter_reset_by_aux:nn #1#2
68 {

```

```

69 \cs_if_exist:cT { c@ #2 }
70 {
71     \tl_if_empty:cF { cl@ #2 }
72     {
73         \tl_map_tokens:cn { cl@ #2 }
74         { \__zrefclever_counter_reset_by_auxi:nnn {#2} {#1} }
75     }
76 }
77 }
78 \cs_new:Npn \__zrefclever_counter_reset_by_auxi:nnn #1#2#3
79 {
80     \str_if_eq:nnT {#2} {#3}
81     { \tl_map_break:n { \seq_map_break:n {#1} } }
82 }

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

83 \zref@newprop { zc@enclcnt }
84 { \__zrefclever_get_enclosing_counters:V \@currentcounter }
85 \zref@newprop { zc@enclval }
86 { \__zrefclever_get_enclosing_counters_value:V \@currentcounter }
87 \zref@addprop \ZREF@mainlist { zc@enclcnt }
88 \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}`.

```

89 \tl_new:N \g__zrefclever_page_format_tl
90 \cs_new_protected:Npx \__zrefclever_page_format_aux: { \int_eval:n { 1 } }
91 \AddToHook { shipout / before }
92 {
93     \group_begin:
94     \cs_set_eq:NN \c@page \__zrefclever_page_format_aux:
95     \exp_args:NNx \tl_gset:Nn \g__zrefclever_page_format_tl { \thepage }
96     \group_end:
97 }

```

```

98 \zref@newprop* { zc@pgfmt } { \g__zrefclever_page_format_tl }
99 \zref@addprop \ZREF@mainlist { zc@pgfmt }

```

Still another property which we don't need to handle at the data provision side, but need to cater for at the retrieval side, is the `url` property (or the equivalent `urluse`) from the `zref-xr` module, which is added to the labels imported from external documents, and needed to construct hyperlinks to them.

4 Plumbing

4.1 Messages

```

100 \msg_new:nnn { zref-clever } { option-not-type-specific }
101 {
102   Option~'#1'~is-not-type-specific~\msg_line_context:~
103   Set~it~in~'\iow_char:N\zcDeclareTranslations'~before~first~'type'~switch~
104   or~as~package~option.
105 }
106 \msg_new:nnn { zref-clever } { option-only-type-specific }
107 {
108   No~type~specified~for~option~'#1'~\msg_line_context:~
109   Set~it~after~'type'~switch~or~in~'\iow_char:N\zcRefTypeSetup'.
110 }
111 \msg_new:nnn { zref-clever } { key-requires-value }
112 { The~'#1'~key~'#2'~requires~a~value. }
113 \msg_new:nnn { zref-clever } { language-declared }
114 { Language~'#1'~is~already~declared.~Nothing~to~do. }
115 \msg_new:nnn { zref-clever } { alias-declared }
116 { Language~'#1'~is~already~an~alias~to~'#2'.~Nothing~to~do. }
117 \msg_new:nnn { zref-clever } { unknown-language-alias }
118 {
119   Language~'#1'~is~unknown,~cannot~alias~to~it.~See~documentation~for~
120   '\iow_char:N\zcDeclareLanguage'~and~'\iow_char:N\zcDeclareLanguageAlias'.
121 }
122 \msg_new:nnn { zref-clever } { unknown-language-transl }
123 {
124   Language~'#1'~is~unknown,~cannot~declare~translations~to~it.~
125   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
126   '\iow_char:N\zcDeclareLanguageAlias'.
127 }
128 \msg_new:nnn { zref-clever } { dict-loaded }
129 { Loaded~'#1'~dictionary. }
130 \msg_new:nnn { zref-clever } { dict-not-available }
131 { Dictionary~for~'#1'~not~available. }
132 \msg_new:nnn { zref-clever } { unknown-language-load }
133 {
134   Unable~to~load~dictionary.~Language~'#1'~is~unknown.~See~documentation~for~
135   '\iow_char:N\zcDeclareLanguage'~and~'\iow_char:N\zcDeclareLanguageAlias'.
136 }
137 \msg_new:nnn { zref-clever } { missing-zref-titleref }
138 {
139   Option~'ref=title'~requested~\msg_line_context:~
140   But~package~'zref-titleref'~is~not~loaded,~falling~back~to~default~'ref'.
141 }

```

```

142 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
143 {
144   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
145   Use~the~starred~version~of~'\iow_char:N\zcref'~instead.
146 }
147 \msg_new:nnn { zref-clever } { missing-hyperref }
148 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
149 \msg_new:nnn { zref-check } { check-document-only }
150 { Option~'check'~only~available~in~the~document. }
151 \msg_new:nnn { zref-clever } { missing-zref-check }
152 {
153   Option~'check'~requested~\msg_line_context:..~
154   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
155 }
156 \msg_new:nnn { zref-clever } { counters-not-nested }
157 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:.. }
158 \msg_new:nnn { zref-clever } { missing-type }
159 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
160 \msg_new:nnn { zref-clever } { missing-name }
161 { Name~undefined~for~type~'#1'~\msg_line_context:.. }
162 \msg_new:nnn { zref-clever } { single-element-range }
163 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:.. }

```

4.2 Reference format

Formatting how the reference is to be typeset is, quite naturally, a big part of the user interface of `zref-clever`. In this area, we tried to balance “flexibility” and “user friendliness”. But the former does place a big toll overall, since there are indeed many places where tweaking may be desired, and the settings may depend on at least two important dimensions of variation: the reference type and the language. Combination of those necessarily makes for a large set of possibilities. Hence, the attempt here is to provide a rich set of “handles” for fine tuning the reference format but, at the same time, do not *require* detailed setup by the users, unless they really want it.

With that in mind, we have settled with an user interface for reference formatting which allows settings to be done in different scopes, with more or less overarching effects, and some precedence rules to regulate the relation of settings given in each of these scopes. There are four scopes in which reference formatting can be specified by the user, in the following precedence order: i) as general *options*; ii) as *type-specific options*; iii) as *language-specific and type-specific translations*; and iv) as *default translations* (that is, language-specific but not type-specific). These precedence rules are handled / enforced in `_zrefclever_get_ref_string:nN` and `_zrefclever_get_ref_font:nN`, which are the basic functions to retrieve proper values for reference format settings.

General “options” (i) can be given by the user in the optional argument of `\zcref`, but just as well in `\zcsetup` or as package options at load-time (see Section 4.5). “Type-specific options” (ii) are handled by `\zcRefTypeSetup`. “Language-specific translations”, be they “type-specific” (iii) or “default” (iv) have their user interface in `\zcDeclareTranslations`, and have their values populated by the package’s dictionaries.

Not all reference format specifications can be given in all of these scopes. Some of them can’t be type-specific, others must be type-specific, so the set available in each scope depends on the pertinence of the case.

The package itself places the default setup for reference formatting at low precedence levels, and the users can easily and conveniently override them as desired. Indeed, I

expect most of the users’ needs to be normally achievable with the general options and type-specific options, since references will normally be typeset in a single language (the document’s main language) and, hence, multiple translations don’t need to be provided.

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

```
164 \tl_new:N \l__zrefclever_setup_type_tl
165 \tl_new:N \l__zrefclever_dict_language_tl
```

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

`\c__zrefclever_ref_options_necessarily_not_type_specific_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.

```
\c__zrefclever_ref_options_font_seq
\c__zrefclever_ref_options_typesetup_seq
\c__zrefclever_ref_options_reference_seq

166 \seq_const_from_clist:Nn
167   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
168   {
169     tpairsep ,
170     tlistsep ,
171     tlastsep ,
172     notesep ,
173   }
174 \seq_const_from_clist:Nn
175   \c__zrefclever_ref_options_possibly_type_specific_seq
176   {
177     namesep ,
178     pairsep ,
179     listsep ,
180     lastsep ,
181     rangesep ,
182     refpre ,
183     refpos ,
184     refpre-in ,
185     refpos-in ,
186   }
```

Only “type names” are “necessarily type-specific”, which makes them somewhat special on the retrieval side of things. In short, they don’t have their values queried by `__zrefclever_get_ref_string:nN`, but by `__zrefclever_type_name_setup:`.

```
187 \seq_const_from_clist:Nn
188   \c__zrefclever_ref_options_necessarily_type_specific_seq
189   {
190     Name-sg ,
191     name-sg ,
192     Name-pl ,
193     name-pl ,
194     Name-sg-ab ,
195     name-sg-ab ,
196     Name-pl-ab ,
197     name-pl-ab ,
198   }
```

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

```

199 \seq_const_from_clist:Nn
200 \c__zrefclever_ref_options_font_seq
201 {
202   namefont ,
203   reffont ,
204   reffont-in ,
205 }
206 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
207 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
208 \c__zrefclever_ref_options_possibly_type_specific_seq
209 \c__zrefclever_ref_options_necessarily_type_specific_seq
210 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
211 \c__zrefclever_ref_options_typesetup_seq
212 \c__zrefclever_ref_options_font_seq
213 \seq_new:N \c__zrefclever_ref_options_reference_seq
214 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
215 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
216 \c__zrefclever_ref_options_possibly_type_specific_seq
217 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
218 \c__zrefclever_ref_options_reference_seq
219 \c__zrefclever_ref_options_font_seq

```

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

4.3 Languages

```

220 \prop_new:N \g__zrefclever_language_aliases_prop
221
222 % {<base language>}
223 \NewDocumentCommand \zcDeclareLanguage { m }
224 {
225   \tl_if_empty:nF {#1}
226   {
227     \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#1}
228     {
229       \str_if_eq:eeTF {#1}
230       { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
231       { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
232       {
233         \msg_warning:nxxx { zref-clever } { alias-declared } {#1}
234         { \prop_item:Nn \g__zrefclever_language_aliases_prop {#1} }
235       }
236     }
237     { \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1} {#1} }
238   }
239 }
240 \@onlypreamble \zcDeclareLanguage
241
242 % {<alias>}{<base language>}
243 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
244 {
245   \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#2}

```

```

246     {
247         \exp_args:NNnx \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1}
248         { \prop_item:Nn \g__zrefclever_language_aliases_prop {#2} }
249     }
250     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
251 }
252 \@onlypreamble \zcDeclareLanguageAlias

```

4.4 Dictionaries

```

253 \seq_new:N \g__zrefclever_loaded_dictionaries_seq
254 \bool_new:N \l__zrefclever_load_dict_verbose_bool
255
256 % {<language>}
257 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
258 {
259     \group_begin:
260     \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
261     \l__zrefclever_dict_language_tl
262     {
263         \seq_if_in:NVF
264         \g__zrefclever_loaded_dictionaries_seq
265         \l__zrefclever_dict_language_tl
266         {
267             \tl_clear:N \l_tmpa_tl
268             \exp_args:Nx \file_get:nnNTF
269             { zref-clever- \l__zrefclever_dict_language_tl .dict }
270             { \ExplSyntaxOn }
271             \l_tmpa_tl
272             {
273                 \prop_if_exist:cF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
274                 { \prop_new:c { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
275                 \tl_clear:N \l__zrefclever_setup_type_tl
276                 \exp_args:NnV
277                 \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
278                 \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
279                 \l__zrefclever_dict_language_tl
280                 \msg_note:nnx { zref-clever } { dict-loaded }
281                 { \l__zrefclever_dict_language_tl }
282             }
283             {
284                 \bool_if:NT \l__zrefclever_load_dict_verbose_bool
285                 {
286                     \msg_warning:nnx { zref-clever } { dict-not-available }
287                     { \l__zrefclever_dict_language_tl }
288                 }
289             }
290         }
291     }
292     {
293         \bool_if:NT \l__zrefclever_load_dict_verbose_bool
294         { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
295     }
296     \group_end:
297 }

```

```

298 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }
299
300 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
301 {
302   \group_begin:
303   \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
304   \__zrefclever_provide_dictionary:n {#1}
305   \group_end:
306 }
307 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }
308
309 % {<key>}{<translation>}
310 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
311 {
312   \exp_args:Nnx \prop_gput_if_new:cnn
313   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
314   { type- \l__zrefclever_setup_type_tl - #1 } {#2}
315 }
316
317 % {<key>}{<translation>}
318 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
319 {
320   \prop_gput_if_new:cnn
321   { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
322   { default- #1 } {#2}
323 }
324
325 \keys_define:nn { zref-clever / dictionary }
326 {
327   type .code:n =
328   {
329     \tl_if_empty:NTF {#1}
330     { \tl_clear:N \l__zrefclever_setup_type_tl }
331     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
332   } ,
333 }
334 \seq_map_inline:Nn
335 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
336 {
337   \keys_define:nn { zref-clever / dictionary }
338   {
339     #1 .value_required:n = true ,
340     #1 .code:n =
341     {
342       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
343       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
344       {
345         \msg_info:nnn { zref-clever }
346         { option-not-type-specific } {#1}
347       }
348     } ,
349   }
350 }
351 \seq_map_inline:Nn
352 \c__zrefclever_ref_options_possibly_type_specific_seq

```

```

352 {
353   \keys_define:nn { zref-clever / dictionary }
354   {
355     #1 .value_required:n = true ,
356     #1 .code:n =
357     {
358       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
359       { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
360       { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
361     } ,
362   }
363 }
364 \seq_map_inline:Nn
365   \c__zrefclever_ref_options_necessarily_type_specific_seq
366   {
367     \keys_define:nn { zref-clever / dictionary }
368     {
369       #1 .value_required:n = true ,
370       #1 .code:n =
371       {
372         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
373         {
374           \msg_info:nnn { zref-clever }
375           { option-only-type-specific } {#1}
376         }
377         { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
378       } ,
379     }
380   }
381 % {<language>}{<type>}{<key>><tl var to set>
382 \prg_new_protected_conditional:Npnn \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
383 {
384   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
385   \l__zrefclever_dict_language_tl
386   {
387     \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
388     { type- #2 - #3 } #4
389     { \prg_return_true: }
390     { \prg_return_false: }
391   }
392   { \prg_return_false: }
393 }
394 \prg_generate_conditional_variant:Nnn \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F
395
396 % {<language>}{<key>><tl var to set>
397 \prg_new_protected_conditional:Npnn \__zrefclever_get_default_transl:nnN #1#2#3 { F }
398 {
399   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
400   \l__zrefclever_dict_language_tl
401   {
402     \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
403     { default- #2 } #3
404     { \prg_return_true: }
405     { \prg_return_false: }

```

```

406     }
407     { \prg_return_false: }
408 }
409 \prg_generate_conditional_variant:Nnn \__zrefclever_get_default_transl:nnN { xnN } { F }
410
411 % {<key>}<tl var to set>
412 \prg_new_protected_conditional:Npnn \__zrefclever_get_fallback_transl:nN #1#2 { F }
413 {
414   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
415     { #1 } #2
416     { \prg_return_true: }
417     { \prg_return_false: }
418 }

```

All “strings” queried with `__zrefclever_get_ref_string:nN` – in practice, those in either `\c__zrefclever_ref_options_necessarily_not_type_specific_seq` or `\c__zrefclever_ref_options_possibly_type_specific_seq` – must have their values set for “fallback”, even if to empty ones, since this is what will be retrieved in the absence of a proper translation, which will be the case if `babel` or `polyglossia` is loaded and sets a language which `zref-clever` does not know. On the other hand, “type names” are not looked for in “fallback”, since it is indeed impossible to provide any reasonable value for them for a “specified but unknown language”.

```

419 \prop_new:N \g__zrefclever_fallback_dict_prop
420 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
421 {
422   tpairsep = {,~} ,
423   tlistsep = {,~} ,
424   tlastsep = {,~} ,
425   notesep  = {~} ,
426   namesep  = {\nobreakspace} ,
427   pairsep  = {,~} ,
428   listsep  = {,~} ,
429   lastsep  = {,~} ,
430   rangesep = {\textendash} ,
431   refpre   = {} ,
432   refpos   = {} ,
433   refpre-in = {} ,
434   refpos-in = {} ,
435 }

```

4.5 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn \langle property list \rangle { \langle key \rangle } { \langle value \rangle }
436 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
437 {
438   \tl_if_empty:nTF {#3}
439     { \prop_remove:Nn #1 {#2} }
440     { \prop_put:Nnn #1 {#2} {#3} }
441 }

```

(End definition for `__zrefclever_prop_put_non_empty:Nnn`.)

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`.

```
442 \prop_new:N \l__zrefclever_counter_type_prop
443 \keys_define:nn { zref-clever / label }
444 {
445   countertype .code:n =
446   {
447     \keyval_parse:nnn
448     {
449       \msg_warning:nnnn { zref-clever }
450       { key-requires-value } { countertype }
451     }
452     {
453       \__zrefclever_prop_put_non_empty:Nnn
454       \l__zrefclever_counter_type_prop
455     }
456     {#1}
457   } ,
458   countertype .value_required:n = true ,
459   countertype .initial:n =
460   {
461     subsection      = section ,
462     subsubsection    = section ,
463     subparagraph     = paragraph ,
464     enumi            = item ,
465     enumii           = item ,
466     enumiii          = item ,
467     enumiv           = item ,
468   } ,
469 }
```

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclcnt` and `zc@enclval` properties, 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.

```
470 \seq_new:N \l__zrefclever_counter_resetters_seq
471 \keys_define:nn { zref-clever / label }
472 {
473   counterresetters .code:n =
474   {
475     \clist_map_inline:nn {#1}
476     {
477       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
```

```

478         {
479             \seq_put_right:Nn
480             \l__zrefclever_counter_resettters_seq {##1}
481         }
482     }
483 },
484 counterresetters .initial:n =
485 {
486     part ,
487     chapter ,
488     section ,
489     subsection ,
490     subsubsection ,
491     paragraph ,
492     subparagraph ,
493 },
494 typesort .value_required:n = true ,
495 }

```

counterresetby option

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

```

496 \prop_new:N \l__zrefclever_counter_resetby_prop
497 \keys_define:nn { zref-clever / label }
498 {
499     counterresetby .code:n =
500     {
501         \keyval_parse:nnn
502         {
503             \msg_warning:nnn { zref-clever }
504             { key-requires-value } { counterresetby }
505         }
506         {
507             \__zrefclever_prop_put_non_empty:Nnn
508             \l__zrefclever_counter_resetby_prop
509             {##1}
510         }
511     } ,
512     counterresetby .value_required:n = true ,
513     counterresetby .initial:n =
514     {

```

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.

```

515         enumii = enumi ,
516         enumiii = enumii ,
517         enumiv = enumiii ,
518     } ,
519 }

```


ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these two (or three) alternatives – `zc@thecnt`, `page`, and `title` if `zref-titleref` is loaded –, but there might be a case for making this more flexible. The infrastructure can already handle receiving an arbitrary property, as long as one is satisfied with sorting and compressing from the default counter. If more flexibility is granted, 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 (see <https://github.com/ho-tex/zref/issues/13>, thanks Ulrike Fischer). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door.

```
520 \tl_new:N \l__zrefclever_ref_property_tl
521 \keys_define:nn { zref-clever / reference }
522 {
523   ref .choice: ,
524   ref / zc@thecnt .code:n =
525     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
526   ref / page .code:n =
527     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
528   ref / title .code:n =
529     {
530       \AddToHook { begindocument }
531       {
532         \@ifpackageloaded { zref-titleref }
533         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
534         {
535           \msg_warning:nn { zref-clever } { missing-zref-titleref }
536           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
537         }
538       }
539     } ,
540   ref .initial:n = zc@thecnt ,
541   ref .value_required:n = true ,
542   page .meta:n = { ref = page },
543   page .value_forbidden:n = true ,
544 }
545 \AddToHook { begindocument }
546 {
547   \@ifpackageloaded { zref-titleref }
548   {
549     \keys_define:nn { zref-clever / reference }
550     {
551       ref / title .code:n =
552       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
553     }
554   }
555   {
556     \keys_define:nn { zref-clever / reference }
557     {
558       ref / title .code:n =
559       {
```

```

560             \msg_warning:nn { zref-clever } { missing-zref-titleref }
561             \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
562         }
563     }
564 }
565 }

```

typeset option

```

566 \bool_new:N \l__zrefclever_typeset_ref_bool
567 \bool_new:N \l__zrefclever_typeset_name_bool
568 \keys_define:nn { zref-clever / reference }
569 {
570     typeset .choice: ,
571     typeset / both .code:n =
572     {
573         \bool_set_true:N \l__zrefclever_typeset_ref_bool
574         \bool_set_true:N \l__zrefclever_typeset_name_bool
575     } ,
576     typeset / ref .code:n =
577     {
578         \bool_set_true:N \l__zrefclever_typeset_ref_bool
579         \bool_set_false:N \l__zrefclever_typeset_name_bool
580     } ,
581     typeset / name .code:n =
582     {
583         \bool_set_false:N \l__zrefclever_typeset_ref_bool
584         \bool_set_true:N \l__zrefclever_typeset_name_bool
585     } ,
586     typeset .initial:n = both ,
587     typeset .value_required:n = true ,
588
589     noname .meta:n = { typeset = ref },
590     noname .value_forbidden:n = true ,
591 }

```

sort option

```

592 \bool_new:N \l__zrefclever_typeset_sort_bool
593 \keys_define:nn { zref-clever / reference }
594 {
595     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
596     sort .initial:n = true ,
597     sort .default:n = true ,
598     nosort .meta:n = { sort = false },
599     nosort .value_forbidden:n = true ,
600 }

```

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.

```

601 \seq_new:N \l__zrefclever_typesort_seq

```

```

602 \keys_define:nn { zref-clever / reference }
603 {
604   typesort .code:n =
605   {
606     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
607     \seq_reverse:N \l__zrefclever_typesort_seq
608   } ,
609   typesort .initial:n =
610   { part , chapter , section , paragraph } ,
611   typesort .value_required:n = true ,
612   notypesort .code:n =
613   { \seq_clear:N \l__zrefclever_typesort_seq } ,
614   notypesort .value_forbidden:n = true ,
615 }

```

comp option

```

616 \bool_new:N \l__zrefclever_typeset_compress_bool
617 \keys_define:nn { zref-clever / reference }
618 {
619   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
620   comp .initial:n = true ,
621   comp .default:n = true ,
622   nocomp .meta:n = { comp = false } ,
623   nocomp .value_forbidden:n = true ,
624 }

```

range option

```

625 \bool_new:N \l__zrefclever_typeset_range_bool
626 \keys_define:nn { zref-clever / reference }
627 {
628   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
629   range .initial:n = false ,
630   range .default:n = true ,
631 }

```

hyperref option

```

632 \bool_new:N \l__zrefclever_use_hyperref_bool
633 \bool_new:N \l__zrefclever_warn_hyperref_bool
634 \keys_define:nn { zref-clever / reference }
635 {
636   hyperref .choice: ,
637   hyperref / auto .code:n =
638   {
639     \bool_set_true:N \l__zrefclever_use_hyperref_bool
640     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
641   } ,
642   hyperref / true .code:n =
643   {
644     \bool_set_true:N \l__zrefclever_use_hyperref_bool
645     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
646   } ,
647   hyperref / false .code:n =
648   {

```

```

649     \bool_set_false:N \l__zrefclever_use_hyperref_bool
650     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
651   } ,
652   hyperref .initial:n = auto ,
653   hyperref .default:n = auto
654 }
655 \AddToHook { begindocument }
656 {
657   \@ifpackageloaded { hyperref }
658   {
659     \bool_if:NT \l__zrefclever_use_hyperref_bool
660     { \RequirePackage { zref-hyperref } }
661   }
662   {
663     \bool_if:NT \l__zrefclever_warn_hyperref_bool
664     { \msg_warning:nn { zref-clever } { missing-hyperref } }
665     \bool_set_false:N \l__zrefclever_use_hyperref_bool
666   }
667   \keys_define:nn { zref-clever / reference }
668   {
669     hyperref .code:n =
670     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
671   }
672 }

```

nameinlink option

```

673 \str_new:N \l__zrefclever_nameinlink_str
674 \keys_define:nn { zref-clever / reference }
675 {
676   nameinlink .choice: ,
677   nameinlink / true .code:n =
678   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
679   nameinlink / false .code:n =
680   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
681   nameinlink / single .code:n =
682   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
683   nameinlink / tsingle .code:n =
684   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
685   nameinlink .initial:n = tsingle ,
686   nameinlink .default:n = true ,
687 }

```

cap and capfirst options

```

688 \bool_new:N \l__zrefclever_capitalize_bool
689 \bool_new:N \l__zrefclever_capitalize_first_bool
690 \keys_define:nn { zref-clever / reference }
691 {
692   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
693   cap .initial:n = false ,
694   cap .default:n = true ,
695   nocap .meta:n = { cap = false } ,
696   nocap .value_forbidden:n = true ,
697
698   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,

```

```

699     capfirst .initial:n = false ,
700     capfirst .default:n = true ,
701
702     C .meta:n =
703       { capfirst = true , noabbrevfirst = true },
704     C .value_forbidden:n = true ,
705   }

```

abbrev and noabbrevfirst options

```

706 \bool_new:N \l__zrefclever_abbrev_bool
707 \bool_new:N \l__zrefclever_noabbrevfirst_bool
708 \keys_define:nn { zref-clever / reference }
709 {
710   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
711   abbrev .initial:n = false ,
712   abbrev .default:n = true ,
713   noabbrev .meta:n = { abbrev = false },
714   noabbrev .value_forbidden:n = true ,
715
716   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrevfirst_bool ,
717   noabbrevfirst .initial:n = false ,
718   noabbrevfirst .default:n = true ,
719 }

```

lang option

`\l__zrefclever_current_language_tl` is an internal alias for babel’s `\language` or polyglossia’s `\mainbabelname`, 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 “main” and “current” document languages, this must be retrieved at a `begindocument` hook. The `begindocument` hook is responsible to get values for `\l__zrefclever_main_language_tl` and `\l__zrefclever_current_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 are set. 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 main language’s dictionary gets loaded, if it hadn’t been already.

For the `babel` and `polyglossia` variables which store the “main” and “current” 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`.

```

720 \tl_new:N \l__zrefclever_ref_language_tl

```

```

721 \tl_new:N \l__zrefclever_main_language_tl
722 \tl_new:N \l__zrefclever_current_language_tl
723 \AddToHook { begindocument }
724 {
725   \@ifpackageloaded { babel }
726   {
727     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
728     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
729   }
730   {
731     \@ifpackageloaded { polyglossia }
732     {
733       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
734       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
735     }
736     {
737       \tl_set:Nn \l__zrefclever_current_language_tl { english }
738       \tl_set:Nn \l__zrefclever_main_language_tl { english }
739     }
740   }

```

Provide default value for `\l__zrefclever_ref_language_tl` corresponding to option `main`, but do so outside of the `l3keys` machinery, so that we are able to distinguish when the user actually gave the option, in which case, the dictionary loading is done verbosely.

```

741   \tl_set:Nn \l__zrefclever_ref_language_tl { \l__zrefclever_main_language_tl }
742 }
743 \keys_define:nn { zref-clever / reference }
744 {
745   lang .code:n =
746   {
747     \AddToHook { begindocument }
748     {
749       \str_case:nnF {#1}
750       {
751         { main }
752         {
753           \tl_set:Nn \l__zrefclever_ref_language_tl
754             { \l__zrefclever_main_language_tl }
755           \__zrefclever_provide_dictionary_verbosely:x
756             { \l__zrefclever_ref_language_tl }
757         }
758       }
759       { current }
760       {
761         \tl_set:Nn \l__zrefclever_ref_language_tl
762           { \l__zrefclever_current_language_tl }
763         \__zrefclever_provide_dictionary_verbosely:x
764           { \l__zrefclever_ref_language_tl }
765       }
766     }
767   }
768   \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
769   \__zrefclever_provide_dictionary_verbosely:x
770     { \l__zrefclever_ref_language_tl }

```

```

771         }
772     } ,
773     lang .value_required:n = true ,
774 }
775
776 \AddToHook { begindocument / before }
777 {
778     \AddToHook { begindocument }
779     {

```

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

```

780     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Redefinition of the `lang` key option for the document body.

```

781     \keys_define:nn { zref-clever / reference }
782     {
783         lang .code:n =
784         {
785             \str_case:nnF {#1}
786             {
787                 { main }
788                 {
789                     \tl_set:Nn \l__zrefclever_ref_language_tl
790                     { \l__zrefclever_main_language_tl }
791                     \__zrefclever_provide_dictionary_verbos:x
792                     { \l__zrefclever_ref_language_tl }
793                 }
794
795                 { current }
796                 {
797                     \tl_set:Nn \l__zrefclever_ref_language_tl
798                     { \l__zrefclever_current_language_tl }
799                     \__zrefclever_provide_dictionary_verbos:x
800                     { \l__zrefclever_ref_language_tl }
801                 }
802             }
803             {
804                 \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
805                 \__zrefclever_provide_dictionary_verbos:x
806                 { \l__zrefclever_ref_language_tl }
807             }
808         } ,
809         lang .value_required:n = true ,
810     }
811 }
812 }

```

font option

```

813 \tl_new:N \l__zrefclever_ref_typeset_font_tl
814 \keys_define:nn { zref-clever / reference }
815 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

note option

```
816 \tl_new:N \l__zrefclever_zcref_note_tl
817 \keys_define:nn { zref-clever / reference }
818 {
819     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
820     note .value_required:n = true ,
821 }
```

check option

Integration with zref-check.

```
822 \bool_new:N \l__zrefclever_zrefcheck_available_bool
823 \bool_new:N \l__zrefclever_zcref_with_check_bool
824 \keys_define:nn { zref-clever / reference }
825 {
826     check .code:n =
827     { \msg_warning:nn { zref-clever } { check-document-only } } ,
828 }
829 \AddToHook { begindocument }
830 {
831     \@ifpackageloaded { zref-check }
832     {
833         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
834         \keys_define:nn { zref-clever / reference }
835         {
836             check .code:n =
837             {
838                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
839                 \keys_set:nn { zref-check / zcheck } {#1}
840             }
841         }
842     }
843     {
844         \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
845         \keys_define:nn { zref-clever / reference }
846         {
847             check .code:n =
848             { \msg_warning:nn { zref-clever } { missing-zref-check } }
849         }
850     }
851 }
```

Reference options

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

The keys are set so that any value, including an empty one, is added to `\l__zrefclever_ref_options_prop`, while a key with *no value* removes the property from

the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.2.

```

852 \prop_new:N \l__zrefclever_ref_options_prop
853 \seq_map_inline:Nn
854 \c__zrefclever_ref_options_reference_seq
855 {
856   \keys_define:nn { zref-clever / reference }
857   {
858     #1 .default:V = \c_novalue_tl ,
859     #1 .code:n =
860     {
861       \tl_if_novalue:nTF {##1}
862       { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
863       { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
864     } ,
865   }
866 }
```

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 the `\zcref`’s options. Anyway, for load-time package options and for `\zcsetup` we want the whole set, so we aggregate the two into `zref-clever/zcsetup`, and use that here.

```

867 \keys_define:nn { }
868 {
869   zref-clever / zcsetup .inherit:n = zref-clever / label ,
870   zref-clever / zcsetup .inherit:n = zref-clever / reference ,
871 }
```

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

```

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

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

873 \NewDocumentCommand \zcsetup { m }
874 { \keys_set:nn { zref-clever / zcsetup } {#1} }
```

(End definition for `\zcsetup`.)

5.2 \zcRefTypeSetup

`\zcRefTypeSetup` is the main user interface for “type-specific” reference formatting. Settings done by this command have a higher precedence than any translation, hence they override any language-specific setting, either done at `\zcDeclareTranslations` or by the package’s dictionaries. On the other hand, they have a lower precedence than non type-specific general options. The *options* should be given in the usual `key=val` format.

The $\langle type \rangle$ does not need to pre-exist, the property list variable to store the properties for the type gets created if need be.

```
\zcRefTypeSetup      \zcRefTypeSetup {\langle type \rangle} {\langle options \rangle}
875 \NewDocumentCommand \zcRefTypeSetup { m m }
876 {
877   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
878   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
879   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
880   \keys_set:nn { zref-clever / typesetup } {#2}
881 }
```

(End definition for $\backslash\text{zcRefTypeSetup}$.)

Inside $\backslash\text{zcRefTypeSetup}$ any of the options *can* receive empty values, and those values, if they exist in the property list, will override translations, regardless of their emptiness. In principle, we could live with the situation of, once a setting has been made in $\backslash\text{l__zrefclever_type_}\langle type \rangle_options_prop$ or in $\backslash\text{l__zrefclever_ref_options_prop}$ it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the translations at any given point. So both in $\backslash\text{zcRefTypeSetup}$ and in setting reference options (see Section 4.5), we leverage the distinction of an “empty valued key” (key= or $\text{key={}}$) from a “key with no value” (key). This distinction is captured internally by the lower-level key parsing, but must be made explicit at $\backslash\text{keys_set:nn}$ by means of the .default:V property of the key in $\backslash\text{keys_define:nn}$. For the technique and some discussion about it, see <https://tex.stackexchange.com/q/614690> (thanks Jonathan P. Spratte, aka ‘Skillmon’, and Phelype Oleinik) and <https://github.com/latex3/latex3/pull/988>.

```
882 \seq_map_inline:Nn
883   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
884   {
885     \keys_define:nn { zref-clever / typesetup }
886     {
887       #1 .code:n =
888       {
889         \msg_warning:nnn { zref-clever }
890         { option-not-type-specific } {#1}
891       } ,
892     }
893   }
894 \seq_map_inline:Nn
895   \c__zrefclever_ref_options_typesetup_seq
896   {
897     \keys_define:nn { zref-clever / typesetup }
898     {
899       #1 .default:V = \c_novaluel_tl ,
900       #1 .code:n =
901       {
902         \tl_if_novalue:nTF {##1}
903         {
904           \prop_remove:cn
905           {
906             l__zrefclever_type_
```

```

907         \l__zrefclever_setup_type_tl _options_prop
908     }
909     {#1}
910 }
911 {
912     \prop_put:cnn
913     {
914         l__zrefclever_type_
915         \l__zrefclever_setup_type_tl _options_prop
916     }
917     {#1} {##1}
918 }
919 } ,
920 }
921 }

```

5.3 \zcDeclareTranslations

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

```

\zcDeclareTranslations      \zcDeclareTranslations {\language} {\options}
922 \NewDocumentCommand \zcDeclareTranslations { m m }
923 {
924     \group_begin:
925     \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
926     \l__zrefclever_dict_language_tl
927     {
928         \tl_clear:N \l__zrefclever_setup_type_tl
929         \keys_set:nn { zref-clever / translations } {#2}
930     }
931     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
932     \group_end:
933 }
934 \@onlypreamble \zcDeclareTranslations

```

(End definition for `\zcDeclareTranslations`.)

```

935 \keys_define:nn { zref-clever / translations }
936 {
937     type .code:n =
938     {
939         \tl_if_empty:nTF {#1}
940         { \tl_clear:N \l__zrefclever_setup_type_tl }
941         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
942     } ,
943 }

```

```

944 % {<language>}{<type>}{<key>}{<translation>}
945 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
946 {
947   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
948   { type- #2 - #3 } {#4}
949 }
950 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
951
952 % {<language>}{<key>}{<translation>}
953 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
954 {
955   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
956   { default- #2 } {#3}
957 }
958 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }
959
960 \seq_map_inline:Nn
961   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
962   {
963     \keys_define:nn { zref-clever / translations }
964     {
965       #1 .value_required:n = true ,
966       #1 .code:n =
967       {
968         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
969         {
970           \__zrefclever_declare_default_transl:Vnn
971           \l__zrefclever_dict_language_tl
972           {#1} {##1}
973         }
974         {
975           \msg_warning:nnn { zref-clever }
976           { option-not-type-specific } {#1}
977         }
978       } ,
979     }
980
981 \seq_map_inline:Nn
982   \c__zrefclever_ref_options_possibly_type_specific_seq
983   {
984     \keys_define:nn { zref-clever / translations }
985     {
986       #1 .value_required:n = true ,
987       #1 .code:n =
988       {
989         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
990         {
991           \__zrefclever_declare_default_transl:Vnn
992           \l__zrefclever_dict_language_tl
993           {#1} {##1}
994         }
995         {
996           \__zrefclever_declare_type_transl:VVnn
997           \l__zrefclever_dict_language_tl

```

```

997             \l__zrefclever_setup_type_tl
998             {#1} {##1}
999         }
1000     } ,
1001 }
1002 }
1003 \seq_map_inline:Nn
1004   \c__zrefclever_ref_options_necessarily_type_specific_seq
1005   {
1006     \keys_define:nn { zref-clever / translations }
1007     {
1008       #1 .value_required:n = true ,
1009       #1 .code:n =
1010       {
1011         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1012         {
1013           \msg_warning:nnn { zref-clever }
1014             { option-only-type-specific } {#1}
1015         }
1016         {
1017           \__zrefclever_declare_type_transl:VVnn
1018             \l__zrefclever_dict_language_tl
1019             \l__zrefclever_setup_type_tl
1020             {#1} {##1}
1021         }
1022       } ,
1023     }
1024 }

```

6 User interface

6.1 \zcref

```

\zcref          \zcref<*>[<options>]{<labels>}
1025 \NewDocumentCommand \zcref { s O { } m }
1026   { \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>}
1027 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1028   {
1029     \group_begin:
Set options.
1030     \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1031     \seq_set_from_clist:Nn \l__zrefclever_zceref_labels_seq {#1}
1032     \bool_set:Nn \l__zrefclever_link_star_bool {#2}

```

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

```

1033     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with zref-check.

```

1034     \bool_lazy_and:nnT
1035     { \l__zrefclever_zrefcheck_available_bool }
1036     { \l__zrefclever_zceref_with_check_bool }
1037     { \zrefcheck_zceref_beg_label: }

```

Sort the labels.

```

1038     \bool_lazy_or:nnT
1039     { \l__zrefclever_typeset_sort_bool }
1040     { \l__zrefclever_typeset_range_bool }
1041     { \__zrefclever_sort_labels: }

```

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

```

1042     \group_begin:
1043     \l__zrefclever_ref_typeset_font_tl
1044     \__zrefclever_typeset_refs:
1045     \group_end:

```

Typeset note.

```

1046     \__zrefclever_get_ref_string:nN {notesep} \l__zrefclever_notesep_tl
1047     \l__zrefclever_notesep_tl
1048     \l__zrefclever_zceref_note_tl

```

Integration with zref-check.

```

1049     \bool_lazy_and:nnT
1050     { \l__zrefclever_zrefcheck_available_bool }
1051     { \l__zrefclever_zceref_with_check_bool }
1052     {
1053         \zrefcheck_zceref_end_label_maybe:
1054         \zrefcheck_zceref_run_checks_on_labels:n
1055         { \l__zrefclever_zceref_labels_seq }
1056     }
1057     \group_end:
1058 }

```

(End definition for `__zrefclever_zceref:nnnn`.)

```

\l__zrefclever_zceref_labels_seq
\l__zrefclever_link_star_bool

```

```

1059 \seq_new:N \l__zrefclever_zceref_labels_seq
1060 \bool_new:N \l__zrefclever_link_star_bool

```

(End definition for `\l__zrefclever_zceref_labels_seq` and `\l__zrefclever_link_star_bool`.)

6.2 \zcpageref

```
\zcpageref          \zcpageref{<*>[<options>]{<labels>}}
1061 \NewDocumentCommand \zcpageref { s O { } m }
1062 {
1063   \IfBooleanTF {#1}
1064     { \zcref*[#2, ref = page] {#3} }
1065     { \zcref [ #2, ref = page] {#3} }
1066 }
```

(End definition for \zcpageref.)

7 Sorting

Aux variables, for use in sorting and typesetting. I could probably let go some of them in favor of tmpa/tmpb, but they do improve code readability.

```
\l__zrefclever_label_a_tl 1067 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl 1068 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_label_type_a_tl 1069 \tl_new:N \l__zrefclever_label_type_a_tl
\l__zrefclever_label_type_b_tl 1070 \tl_new:N \l__zrefclever_label_type_b_tl
\l__zrefclever_label_enclcnt_a_tl 1071 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
\l__zrefclever_label_enclcnt_b_tl 1072 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
\l__zrefclever_label_enclval_a_tl 1073 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l__zrefclever_label_enclval_b_tl 1074 \tl_new:N \l__zrefclever_label_enclval_b_tl
```

(End definition for \l__zrefclever_label_a_tl and others.)

```
1075 \int_new:N \l__zrefclever_sort_prior_a_int
1076 \int_new:N \l__zrefclever_sort_prior_b_int
```

Auxiliary variable for __zrefclever_sort_default:nn, signals if the sorting between two labels has been decided or not.

```
1077 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for \l__zrefclever_sort_decided_bool.)

Variant not provided by the kernel.

```
1078 \cs_generate_variant:Nn \tl_reverse_items:n { V }
```

Auxiliary function used to store “new” label types (in order) as the sorting proceeds. It is expected to be run inside __zrefclever_sort_labels:, and stores new types in \l__zrefclever_label_types_seq.

```
\__zrefclever_label_type_put_new_right:n {<label>}
```

```
1079 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1080 {
1081   \tl_set:Nx \l__zrefclever_label_type_a_tl
1082     { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1083   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1084   {
1085     \seq_if_in:NVF
1086       \l__zrefclever_label_types_seq
1087       \l__zrefclever_label_type_a_tl
1088     {
```

```

1089         \seq_put_right:NV \l__zrefclever_label_types_seq
1090         \l__zrefclever_label_type_a_tl
1091     }
1092 }
1093 }

```

(End definition for __zrefclever_label_type_put_new_right:n.)

\l__zrefclever_label_types_seq Stores the order in which reference types appear in the label list supplied by the user in \zcref. This order is required as a “last resort” sort criterion between the reference types, for use in __zrefclever_sort_default:nn.

```

1094 \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.

```

1095 \cs_new_protected:Npn \__zrefclever_sort_labels:
1096 {

```

Store label types sequence.

```

1097     \seq_clear:N \l__zrefclever_label_types_seq
1098     \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1099     {
1100         \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1101         \__zrefclever_label_type_put_new_right:n
1102     }

```

Sort.

```

1103     \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1104     {
1105         \zref@ifrefundefined {##1}
1106         {
1107             \zref@ifrefundefined {##2}
1108             {
1109                 % Neither label is defined.
1110                 \sort_return_same:
1111             }
1112             {
1113                 % The second label is defined, but the first isn't, leave the
1114                 % undefined first (to be more visible).
1115                 \sort_return_same:
1116             }
1117         }
1118         {
1119             \zref@ifrefundefined {##2}
1120             {
1121                 % The first label is defined, but the second isn't, bring the
1122                 % second forward.
1123                 \sort_return_swapped:
1124             }
1125             {

```



```

1126             % The interesting case: both labels are defined. The
1127             % reference to the "default" property/counter or to the page
1128             % are quite different from our perspective, they rely on
1129             % different fields and even use different information for
1130             % sorting, so we branch them here to specialized functions.
1131             \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1132             { \__zrefclever_sort_page:nn {##1} {##2} }
1133             { \__zrefclever_sort_default:nn {##1} {##2} }
1134         }
1135     }
1136 }
1137 }

```

(End definition for __zrefclever_sort_labels:.)

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

```

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

1138 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1139 {
1140     \tl_set:Nx \l__zrefclever_label_type_a_tl
1141     { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1142     \tl_set:Nx \l__zrefclever_label_type_b_tl
1143     { \zref@extractdefault {#2} { zc@type } { \c_empty_tl } }
1144
1145     \bool_if:nTF
1146     {
1147         % The second label has a type, but the first doesn't, leave the
1148         % undefined first (to be more visible).
1149         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1150         ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1151     }
1152     { \sort_return_same: }
1153     {
1154         \bool_if:nTF
1155         {
1156             % The first label has a type, but the second doesn't, bring the
1157             % second forward.
1158             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1159             \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1160         }
1161         { \sort_return_swapped: }
1162         {
1163             \bool_if:nTF
1164             {
1165                 % The interesting case: both labels have a type...
1166                 ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1167                 ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1168             }
1169             {

```

```

1170 % Here we send this to a couple of auxiliary functions for no
1171 % other reason than to keep this long function a little less
1172 % unreadable.
1173 \tl_if_eq:NNTF
1174 \l__zrefclever_label_type_a_tl
1175 \l__zrefclever_label_type_b_tl
1176 {
1177 % ...and it's the same type.
1178 \__zrefclever_sort_default_same_type:nn {#1} {#2}
1179 }
1180 {
1181 % ...and they are different types.
1182 \__zrefclever_sort_default_different_types:nn {#1} {#2}
1183 }
1184 }
1185 {
1186 % Neither of the labels has a type. We can't do much of
1187 % meaningful here, but if it's the same counter, compare it.
1188 \exp_args:Nxx \tl_if_eq:nnTF
1189 { \zref@extractdefault {#1} { counter } { } }
1190 { \zref@extractdefault {#2} { counter } { } }
1191 {
1192 \int_compare:nNnTF
1193 { \zref@extractdefault {#1} { zc@cntval } {-1} }
1194 >
1195 { \zref@extractdefault {#2} { zc@cntval } {-1} }
1196 { \sort_return_swapped: }
1197 { \sort_return_same: }
1198 }
1199 { \sort_return_same: }
1200 }
1201 }
1202 }
1203 }

```

(End definition for __zrefclever_sort_default:nn.)

_zrefclever_sort_default_same_type:nn

```

1204 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1205 {
1206 \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1207 { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1208 \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1209 { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1210 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1211 { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1212 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1213 { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1214 \tl_set:Nx \l__zrefclever_label_enclval_c_tl
1215 { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1216 \tl_set:Nx \l__zrefclever_label_enclval_d_tl
1217 { \tl_reverse_items:V \l__zrefclever_label_enclval_c_tl }
1218 \tl_set:Nx \l__zrefclever_label_enclval_e_tl
1219 { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }

```

```

1220 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1221 { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1222
1223 \bool_set_false:N \l__zrefclever_sort_decided_bool
1224 % CHECK should I replace the tmp variables here?
1225 \tl_clear:N \l_tmpa_tl
1226 \tl_clear:N \l_tmpb_tl
1227 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1228 {
1229   \tl_set:Nx \l_tmpa_tl
1230   { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1231   \tl_set:Nx \l_tmpb_tl
1232   { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1233
1234   \bool_if:nTF
1235   {
1236     % Both are empty, meaning: neither labels have any (further)
1237     % ‘enclosing counters’ (left).
1238     \tl_if_empty_p:V \l_tmpa_tl &&
1239     \tl_if_empty_p:V \l_tmpb_tl
1240   }
1241   {
1242     \exp_args:Nxx \tl_if_eq:nTF
1243     { \zref@extractdefault {#1} { counter } { } }
1244     { \zref@extractdefault {#2} { counter } { } }
1245     {
1246       \bool_set_true:N \l__zrefclever_sort_decided_bool
1247       \int_compare:nNnTF
1248       { \zref@extractdefault {#1} { zc@cntval } {-1} }
1249       >
1250       { \zref@extractdefault {#2} { zc@cntval } {-1} }
1251       { \sort_return_swapped: }
1252       { \sort_return_same: }
1253     }
1254     {
1255       \msg_warning:nnnn { zref-clever }
1256       { counters-not-nested } {#1} {#2}
1257       \bool_set_true:N \l__zrefclever_sort_decided_bool
1258       \sort_return_same:
1259     }
1260   }
1261   {
1262     \bool_if:nTF
1263     {
1264       % ‘a’ is empty (and ‘b’ is not), meaning: ‘b’ is (possibly)
1265       % nested in ‘a’.
1266       \tl_if_empty_p:V \l_tmpa_tl
1267     }
1268     {
1269       \tl_set:Nx \l_tmpa_tl
1270       { {\zref@extractdefault {#1} { counter } { } } }
1271       \exp_args:NNx \tl_if_in:NnTF
1272       \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1273       {

```

```

1274         \bool_set_true:N \l__zrefclever_sort_decided_bool
1275         \sort_return_same:
1276     }
1277     {
1278         \msg_warning:nnnn { zref-clever }
1279         { counters-not-nested } {#1} {#2}
1280         \bool_set_true:N \l__zrefclever_sort_decided_bool
1281         \sort_return_same:
1282     }
1283 }
1284 {
1285     \bool_if:nTF
1286     {
1287         % 'b' is empty (and 'a' is not), meaning: 'a' is
1288         % (possibly) nested in 'b'.
1289         \tl_if_empty_p:V \l_tmpb_tl
1290     }
1291     {
1292         \tl_set:Nx \l_tmpb_tl
1293         { {\zref@extractdefault {#2} { counter } { }} }
1294         \exp_args:NNx \tl_if_in:NnTF
1295         \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1296         {
1297             \bool_set_true:N \l__zrefclever_sort_decided_bool
1298             \sort_return_swapped:
1299         }
1300         {
1301             \msg_warning:nnnn { zref-clever }
1302             { counters-not-nested } {#1} {#2}
1303             \bool_set_true:N \l__zrefclever_sort_decided_bool
1304             \sort_return_same:
1305         }
1306     }
1307 }
1308 % Neither is empty, meaning: we can (possibly) compare the
1309 % values of the current enclosing counter in the loop, if
1310 % they are equal, we are still in the loop, if they are
1311 % not, a sorting decision can be made directly.
1312 \tl_if_eq:NNTF \l_tmpa_tl \l_tmpb_tl
1313 {
1314     \int_compare:nNnTF
1315     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1316     =
1317     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1318     {
1319         \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1320         { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1321         \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1322         { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1323         \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1324         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1325         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1326         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1327     }

```

```

1328         {
1329             \bool_set_true:N \l__zrefclever_sort_decided_bool
1330             \int_compare:nNnTF
1331                 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1332                 >
1333                 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1334                 { \sort_return_swapped: }
1335                 { \sort_return_same: }
1336         }
1337     }
1338     {
1339         \msg_warning:nnnn { zref-clever }
1340         { counters-not-nested } {#1} {#2}
1341         \bool_set_true:N \l__zrefclever_sort_decided_bool
1342         \sort_return_same:
1343     }
1344 }
1345 }
1346 }
1347 }
1348 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

`_zrefclever_sort_default_different_types:nn`

```

1349 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1350 {
1351     \int_zero:N \l__zrefclever_sort_prior_a_int
1352     \int_zero:N \l__zrefclever_sort_prior_b_int
1353     % \cs{l__zrefclever_typesort_seq} was stored in reverse sequence, and
1354     % we compute the sort priorities in the negative range, so that we can
1355     % implicitly rely on '0' being the "last value".
1356     \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1357     {
1358         \tl_if_eq:nnTF {##2} {{othertypes}}
1359         {
1360             \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1361             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1362             \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }
1363             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1364         }
1365         {
1366             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1367             { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1368             {
1369                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1370                 { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1371             }
1372         }
1373     }
1374     \bool_if:nTF
1375     {
1376         \int_compare_p:nNn
1377         { \l__zrefclever_sort_prior_a_int } <

```

```

1378         { \l__zrefclever_sort_prior_b_int }
1379     }
1380     { \sort_return_same: }
1381     {
1382         \bool_if:nTF
1383         {
1384             \int_compare_p:nNn
1385             { \l__zrefclever_sort_prior_a_int } >
1386             { \l__zrefclever_sort_prior_b_int }
1387         }
1388         { \sort_return_swapped: }
1389         {
1390             % Sort priorities are equal for different types: the type that
1391             % occurs first in 'labels', as given by the user, is kept (or
1392             % brought) forward.
1393             \seq_map_inline:Nn \l__zrefclever_label_types_seq
1394             {
1395                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1396                 { \seq_map_break:n { \sort_return_same: } }
1397                 {
1398                     \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1399                     { \seq_map_break:n { \sort_return_swapped: } }
1400                 }
1401             }
1402         }
1403     }
1404 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

`__zrefclever_sort_page:nn` The sorting function for sorting of existing 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)}

1405 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1406 {
1407     \int_compare:nNnTF
1408     { \zref@extractdefault {#1} { abspage } {-1} }
1409     >
1410     { \zref@extractdefault {#2} { abspage } {-1} }
1411     { \sort_return_swapped: }
1412     { \sort_return_same: }
1413 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

About possible alternatives to signal compression inhibition for individual labels, see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik,

and Steven B. Segletes). Yet another alternative would be to receive an optional argument with the label(s) not to be compressed. This would be a repetition, but would keep the syntax “clean”. All in all, and rethinking this here, probably the best is simply to not allow individual inhibition of compression. We can already control compression of each individual call of `\zcref` with existing options, this should be enough. I don’t think the small extra flexibility this would grant is worth the syntax disruption it entails. Anyway, I have kept a “handle” to deal with this in case the need arises, in the form of `\l__zrefclever_range_inhibit_next_bool`, which is currently no-op, but is in place.

Variables

`\l_zrefclever_typeset_last_bool` Auxiliary variables for `__zrefclever_typeset_refs:`. `\l_zrefclever_typeset_last_bool` signals if the label list is over so that we can leave the loop. `\l_zrefclever_last_of_type_bool` signals if we are processing the last label of the current reference type.

```
1414 \bool_new:N \l__zrefclever_typeset_last_bool
1415 \bool_new:N \l__zrefclever_last_of_type_bool
```

(End definition for `\l__zrefclever_typeset_last_bool` and `\l__zrefclever_last_of_type_bool`.)

`\l_zrefclever_typeset_labels_seq` Auxiliary variables for `__zrefclever_typeset_refs:`. They store, respectively the “previous” and the “current” reference type information while they are being processed, since we cannot typeset them directly, given we can only know certain things when the (next) type list is over. The “queue” stores all references but the first of the type, and they are stored ready to be typeset. The “first_label” stores the *label* of the first reference for the type, because the name can only be determined at the end, and its (potential) hyperlink must be handled at that point.

```
1416 \seq_new:N \l__zrefclever_typeset_labels_seq
1417 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1418 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1419 \tl_new:N \l__zrefclever_type_first_label_tl
1420 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

`\l_zrefclever_label_count_int` Main counters for `__zrefclever_typeset_refs:`. They track the state of the parsing of the labels list. `\l_zrefclever_label_count_int` is stepped for every reference/label in the list, and reset at the start of a new type. `\l_zrefclever_type_count_int` is stepped at every reference type change.

```
1421 \int_new:N \l__zrefclever_label_count_int
1422 \int_new:N \l__zrefclever_type_count_int
```

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

`\l_zrefclever_range_count_int` Range related auxiliary variables for `__zrefclever_typeset_refs:`. `\l_zrefclever_range_count_int` counts how many references/labels are in the current ongoing range. `\l_zrefclever_range_same_count_int` counts how many of the references in the current ongoing range are repeated ones. `\l_zrefclever_range_beg_label_tl` stores the label of the reference that starts a range. `\l_zrefclever_next_maybe_range_bool` signals whether the next element is in sequence to the current one. `\l_zrefclever_next_is_same_bool` signals whether the next element repeats the current one. `\l_zrefclever_range_inhibit_next_bool` allows to control/track compression inhibition of the next label.

```

1423 \int_new:N \l__zrefclever_range_count_int
1424 \int_new:N \l__zrefclever_range_same_count_int
1425 \tl_new:N \l__zrefclever_range_beg_label_tl
1426 \bool_new:N \l__zrefclever_next_maybe_range_bool
1427 \bool_new:N \l__zrefclever_next_is_same_bool
1428 \bool_new:N \l__zrefclever_range_inhibit_next_bool

```

(End definition for \l__zrefclever_range_count_int and others.)

Aux variables for __zrefclever_typeset_refs:. Store separators and refpre/pos options.

```

1429 \tl_new:N \l__zrefclever_namefont_tl
1430 \tl_new:N \l__zrefclever_reffont_out_tl
1431 \tl_new:N \l__zrefclever_reffont_in_tl
1432
1433 \tl_new:N \l__zrefclever_namesep_tl
1434 \tl_new:N \l__zrefclever_rangesep_tl
1435 \tl_new:N \l__zrefclever_pairsep_tl
1436 \tl_new:N \l__zrefclever_listsep_tl
1437 \tl_new:N \l__zrefclever_lastsep_tl
1438 \tl_new:N \l__zrefclever_tpairsep_tl
1439 \tl_new:N \l__zrefclever_tlistsep_tl
1440 \tl_new:N \l__zrefclever_tlastsep_tl
1441 \tl_new:N \l__zrefclever_notesep_tl
1442 \tl_new:N \l__zrefclever_refpre_out_tl
1443 \tl_new:N \l__zrefclever_refpos_out_tl
1444 \tl_new:N \l__zrefclever_refpre_in_tl
1445 \tl_new:N \l__zrefclever_refpos_in_tl

```

(End definition for .)

\l__zrefclever_type_name_tl Auxiliary variables for __zrefclever_get_ref_first: and __zrefclever_type_name_setup:.

```

1446 \tl_new:N \l__zrefclever_type_name_tl
1447 \bool_new:N \l__zrefclever_name_in_link_bool
1448 \tl_new:N \l__zrefclever_name_format_tl
1449 \tl_new:N \l__zrefclever_name_format_fallback_tl

```

(End definition for \l__zrefclever_type_name_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

```

1450 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1451 {
1452   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zceref_labels_seq
1453   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1454   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1455   \tl_clear:N \l__zrefclever_type_first_label_tl
1456   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1457   \tl_clear:N \l__zrefclever_range_beg_label_tl
1458   \int_zero:N \l__zrefclever_label_count_int
1459   \int_zero:N \l__zrefclever_type_count_int
1460   \int_zero:N \l__zrefclever_range_count_int

```



```

1461 \int_zero:N \l__zrefclever_range_same_count_int
1462
1463 % Get not-type-specific separators and refpre/pos options.
1464 \__zrefclever_get_ref_string:nN {tpairsep} \l__zrefclever_tpairsep_tl
1465 \__zrefclever_get_ref_string:nN {tlistsep} \l__zrefclever_tlistsep_tl
1466 \__zrefclever_get_ref_string:nN {tlastsep} \l__zrefclever_tlastsep_tl
1467
1468 % Loop over the label list in sequence.
1469 \bool_set_false:N \l__zrefclever_typeset_last_bool
1470 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1471 {
1472   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl
1473   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1474   {
1475     \tl_clear:N \l__zrefclever_label_b_tl
1476     \bool_set_true:N \l__zrefclever_typeset_last_bool
1477   }
1478   { \seq_get_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_b_tl }
1479
1480   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1481   {
1482     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1483     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1484   }
1485   {
1486     \tl_set:Nx \l__zrefclever_label_type_a_tl
1487     {
1488       \zref@extractdefault
1489       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1490     }
1491     \tl_set:Nx \l__zrefclever_label_type_b_tl
1492     {
1493       \zref@extractdefault
1494       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1495     }
1496   }
1497
1498   % First, we establish whether the ‘‘current label’’ (i.e. ‘a’) is the
1499   % last one of its type. This can happen because the ‘‘next label’’
1500   % (i.e. ‘b’) is of a different type (or different definition status),
1501   % or because we are at the end of the list.
1502   \bool_if:NTF \l__zrefclever_typeset_last_bool
1503   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1504   {
1505     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1506     {
1507       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1508       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1509       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1510     }
1511     {
1512       \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1513       { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1514       {

```

```

1515 % Neither is undefined, we must check the types.
1516 \bool_if:nTF
1517 % Both empty: same ‘type’.
1518 {
1519   \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1520   \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1521 }
1522 { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1523 {
1524   \bool_if:nTF
1525   % Neither empty: compare types.
1526   {
1527     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1528     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1529   }
1530   {
1531     \tl_if_eq:NNTF
1532     \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1533     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1534     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1535   }
1536   % One empty, the other not: different ‘types’.
1537   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1538 }
1539 }
1540 }
1541 }
1542
1543 % Handle warnings in case of reference or type undefined.
1544 \zref@refused { \l__zrefclever_label_a_tl }
1545 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1546 {}
1547 {
1548   \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1549   {
1550     \msg_warning:nxx { zref-clever } { missing-type }
1551     { \l__zrefclever_label_a_tl }
1552   }
1553 }
1554
1555 % Get type-specific separators, refpre/pos and font options, once per
1556 % type.
1557 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1558 {
1559   \__zrefclever_get_ref_font:nN {namefont} \l__zrefclever_namefont_tl
1560   \__zrefclever_get_ref_font:nN {reffont} \l__zrefclever_reffont_out_tl
1561   \__zrefclever_get_ref_font:nN {reffont-in} \l__zrefclever_reffont_in_tl
1562   \__zrefclever_get_ref_string:nN {namesep} \l__zrefclever_namesep_tl
1563   \__zrefclever_get_ref_string:nN {rangesep} \l__zrefclever_rangesep_tl
1564   \__zrefclever_get_ref_string:nN {pairsep} \l__zrefclever_pairsep_tl
1565   \__zrefclever_get_ref_string:nN {listsep} \l__zrefclever_listsep_tl
1566   \__zrefclever_get_ref_string:nN {lastsep} \l__zrefclever_lastsep_tl
1567   \__zrefclever_get_ref_string:nN {refpre} \l__zrefclever_refpre_out_tl
1568   \__zrefclever_get_ref_string:nN {refpos} \l__zrefclever_refpos_out_tl

```

```

1569     \_zrefclever_get_ref_string:nN {refpre-in} \l_zrefclever_refpre_in_tl
1570     \_zrefclever_get_ref_string:nN {refpos-in} \l_zrefclever_refpos_in_tl
1571   }
1572
1573   % Here we send this to a couple of auxiliary functions for no other
1574   % reason than to keep this long function a little less unreadable.
1575   \bool_if:NTF \l_zrefclever_last_of_type_bool
1576   {
1577     % There exists no next label of the same type as the current.
1578     \_zrefclever_typeset_refs_aux_last_of_type:
1579   }
1580   {
1581     % There exists a next label of the same type as the current.
1582     \_zrefclever_typeset_refs_aux_not_last_of_type:
1583   }
1584 }
1585 }

```

(End definition for _zrefclever_typeset_refs:.)

_zrefclever_typeset_refs_aux_last_of_type:

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

```

1586 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_last_of_type:
1587 {
1588   % Process the current label to the current queue.
1589   \int_case:nnF { \l_zrefclever_label_count_int }
1590   {
1591     % It is the last label of its type, but also the first one, and that's
1592     % what matters here: just store it.
1593     { 0 }
1594     {
1595       \tl_set:NV \l_zrefclever_type_first_label_tl \l_zrefclever_label_a_tl
1596       \tl_set:NV \l_zrefclever_type_first_label_type_tl \l_zrefclever_label_type_a_tl
1597     }
1598
1599     % The last is the second: we have a pair (if not repeated).
1600     { 1 }
1601     {
1602       \int_compare:nNnF { \l_zrefclever_range_same_count_int } = {1}
1603       {
1604         \tl_put_right:Nx \l_zrefclever_typeset_queue_curr_tl
1605         {
1606           \exp_not:V \l_zrefclever_pairsep_tl
1607           \_zrefclever_get_ref:V \l_zrefclever_label_a_tl
1608         }
1609       }
1610     }
1611   }
1612   % If neither the first, nor the second: we have the last label
1613   % on the current type list (if not repeated).
1614   {
1615     \int_case:nnF { \l_zrefclever_range_count_int }
1616     {
1617       % There was no range going on.
1618       {0}

```

```

1619 {
1620   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1621   {
1622     \exp_not:V \l__zrefclever_lastsep_tl
1623     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1624   }
1625 }
1626 % Last in the range is also the second in it.
1627 {1}
1628 {
1629   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1630   {
1631     % We know 'range_beg_label' is not empty, since this is the
1632     % second element in the range, but the third or more in the
1633     % type list.
1634     \exp_not:V \l__zrefclever_listsep_tl
1635     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1636     \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1637     {
1638       \exp_not:V \l__zrefclever_lastsep_tl
1639       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1640     }
1641   }
1642 }
1643 }
1644 % Last in the range is third or more in it.
1645 {
1646   \int_case:nnF
1647   { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1648   {
1649     % Repetition, not a range.
1650     {0}
1651     {
1652       % If 'range_beg_label' is empty, it means it was also the
1653       % first of the type, and hence was already handled.
1654       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1655       {
1656         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1657         {
1658           \exp_not:V \l__zrefclever_lastsep_tl
1659           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1660         }
1661       }
1662     }
1663     % A 'range', but with no skipped value, treat as list.
1664     {1}
1665     {
1666       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1667       {
1668         % Ditto.
1669         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1670         {
1671           \exp_not:V \l__zrefclever_listsep_tl
1672           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl

```

```

1673         }
1674         \exp_not:V \l__zrefclever_lastsep_tl
1675         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1676     }
1677 }
1678 }
1679 {
1680     % An actual range.
1681     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1682     {
1683         % Ditto.
1684         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1685         {
1686             \exp_not:V \l__zrefclever_lastsep_tl
1687             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1688         }
1689         \exp_not:V \l__zrefclever_rangesep_tl
1690         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1691     }
1692 }
1693 }
1694 }
1695
1696 % Handle ‘‘range’’ option. The idea is simple: if the queue is not empty,
1697 % we replace it with the end of the range (or pair). We can still
1698 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1699 % be processing the last label of its type at this point.
1700 \bool_if:NT \l__zrefclever_typeset_range_bool
1701 {
1702     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1703     {
1704         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1705         { }
1706         {
1707             \msg_warning:nxx { zref-clever } { single-element-range }
1708             { \l__zrefclever_type_first_label_type_tl }
1709         }
1710     }
1711     {
1712         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1713         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1714         { }
1715         {
1716             \__zrefclever_labels_in_sequence:nn
1717             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }
1718         }
1719         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1720         {
1721             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1722             { \exp_not:V \l__zrefclever_pairsep_tl }
1723             { \exp_not:V \l__zrefclever_rangesep_tl }
1724             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1725         }
1726     }

```

```

1727     }
1728
1729 % Now that the type is finished, we can add the name and the first ref to
1730 % the queue. Or, if ‘‘typset’’ option is not ‘‘both’’, handle it here
1731 % too.
1732 \__zrefclever_type_name_setup:
1733 \bool_if:nTF
1734 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1735 {
1736     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1737     { \__zrefclever_get_ref_first: }
1738 }
1739 {
1740     \bool_if:nTF
1741     { \l__zrefclever_typeset_ref_bool }
1742     {
1743         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1744         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1745     }
1746     {
1747         \bool_if:nTF
1748         { \l__zrefclever_typeset_name_bool }
1749         {
1750             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1751             {
1752                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1753                 {
1754                     \exp_not:N \group_begin:
1755                     \exp_not:V \l__zrefclever_namefont_tl
1756                     % It's two '@s', but escaped for DocStrip.
1757                     \exp_not:N \hyper@@link
1758                     {
1759                         \zref@ifrefcontainsprop
1760                         { \l__zrefclever_type_first_label_tl } { urluse }
1761                         {
1762                             \zref@extractdefault
1763                             { \l__zrefclever_type_first_label_tl }
1764                             { urluse } {}
1765                         }
1766                         {
1767                             \zref@extractdefault
1768                             { \l__zrefclever_type_first_label_tl }
1769                             { url } {}
1770                         }
1771                     }
1772                     {
1773                         \zref@extractdefault
1774                         { \l__zrefclever_type_first_label_tl } { anchor } {}
1775                     }
1776                     { \exp_not:V \l__zrefclever_type_name_tl }
1777                     \exp_not:N \group_end:
1778                 }
1779             }
1780             \exp_not:N \group_begin:

```

```

1781         \exp_not:V \l__zrefclever_namefont_tl
1782         \exp_not:V \l__zrefclever_type_name_tl
1783         \exp_not:N \group_end:
1784     }
1785 }
1786 }
1787 {
1788     % This case would correspond to "typeset=none" but should not
1789     % happen, given the options are set up to typeset at least one
1790     % of "ref" or "name", but a sensible fallback, equal to the
1791     % behavior of ‘‘both’’.
1792     \tl_put_left:Nx
1793         \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1794 }
1795 }
1796 }
1797
1798 % Typeset the previous type, if there is one.
1799 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1800 {
1801     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1802     { \l__zrefclever_tlistsep_tl }
1803     \l__zrefclever_typeset_queue_prev_tl
1804 }
1805
1806 % Wrap up loop, or prepare for next iteration.
1807 \bool_if:NTF \l__zrefclever_typeset_last_bool
1808 {
1809     % We are finishing, typeset the current queue.
1810     \int_case:nnF { \l__zrefclever_type_count_int }
1811     {
1812         % Single type.
1813         { 0 }
1814         { \l__zrefclever_typeset_queue_curr_tl }
1815         % Pair of types.
1816         { 1 }
1817         {
1818             \l__zrefclever_tpairsep_tl
1819             \l__zrefclever_typeset_queue_curr_tl
1820         }
1821     }
1822     {
1823         % Last in list of types.
1824         \l__zrefclever_tlastsep_tl
1825         \l__zrefclever_typeset_queue_curr_tl
1826     }
1827 }
1828 {
1829     % There are further labels, set variables for next iteration.
1830     \tl_set_eq:NN
1831         \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1832     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1833     \tl_clear:N \l__zrefclever_type_first_label_tl
1834     \tl_clear:N \l__zrefclever_type_first_label_type_tl

```

```

1835     \tl_clear:N \l__zrefclever_range_beg_label_tl
1836     \int_zero:N \l__zrefclever_label_count_int
1837     \int_incr:N \l__zrefclever_type_count_int
1838     \int_zero:N \l__zrefclever_range_count_int
1839     \int_zero:N \l__zrefclever_range_same_count_int
1840   }
1841 }

```

(End definition for _zrefclever_typeset_refs_aux_last_of_type:.)

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

```

1842 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_not_last_of_type:
1843 {
1844   % Signal if next label may form a range with the current one (of
1845   % course, only considered if compression is enabled in the first
1846   % place).
1847   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1848   \bool_set_false:N \l__zrefclever_next_is_same_bool
1849   \bool_lazy_and:nnT
1850     { \l__zrefclever_typeset_compress_bool }
1851     % Currently no-op, but kept as ‘handle’ to inhibit compression of
1852     % individual labels.
1853     { ! \l__zrefclever_range_inhibit_next_bool }
1854     {
1855       \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1856       { }
1857       {
1858         \_zrefclever_labels_in_sequence:nn
1859         { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1860       }
1861     }
1862
1863   % Process the current label to the current queue.
1864   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1865   {
1866     % Current label is the first of its type (also not the last, but it
1867     % doesn't matter here): just store the label.
1868     \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1869     \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1870
1871     % If the next label may be part of a range, we set ‘range_beg_label’
1872     % to ‘empty’ (we deal with it as the ‘first’, and must do it
1873     % there, to handle hyperlinking), but also step the range counters.
1874     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1875     {
1876       \tl_clear:N \l__zrefclever_range_beg_label_tl
1877       \int_incr:N \l__zrefclever_range_count_int
1878       \bool_if:NT \l__zrefclever_next_is_same_bool
1879       { \int_incr:N \l__zrefclever_range_same_count_int }
1880     }
1881   }
1882   {
1883     % Current label is neither the first (nor the last) of its
1884     % type.

```



```

1885 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1886 {
1887   % Starting, or continuing a range.
1888   \int_compare:nNnTF
1889     { \l__zrefclever_range_count_int } = {0}
1890   {
1891     % There was no range going, we are starting one.
1892     \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1893     \int_incr:N \l__zrefclever_range_count_int
1894     \bool_if:NT \l__zrefclever_next_is_same_bool
1895       { \int_incr:N \l__zrefclever_range_same_count_int }
1896   }
1897   {
1898     % Second or more in the range, but not the last.
1899     \int_incr:N \l__zrefclever_range_count_int
1900     \bool_if:NT \l__zrefclever_next_is_same_bool
1901       { \int_incr:N \l__zrefclever_range_same_count_int }
1902   }
1903 }
1904 {
1905   % Next element is not in sequence, meaning: there was no range, or
1906   % we are closing one.
1907   \int_case:nnF { \l__zrefclever_range_count_int }
1908   {
1909     % There was no range going on.
1910     {0}
1911     {
1912       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1913       {
1914         \exp_not:V \l__zrefclever_listsep_tl
1915         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1916       }
1917     }
1918     % Last is second in the range: if 'range_same_count' is also
1919     % '1', it's a repetition (drop it), otherwise, it's a 'pair
1920     % within a list'', treat as list.
1921     {1}
1922     {
1923       \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1924       {
1925         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1926         {
1927           \exp_not:V \l__zrefclever_listsep_tl
1928           \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1929         }
1930         \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1931         {
1932           \exp_not:V \l__zrefclever_listsep_tl
1933           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1934         }
1935       }
1936     }
1937   }
1938 }

```

```

1939 % Last is third or more in the range: if 'range_count' and
1940 % 'range_same_count' are the same, its a repetition (drop it),
1941 % if they differ by '1', its a list, if they differ by more,
1942 % it is a real range.
1943 \int_case:nnF
1944 { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1945 {
1946   {0}
1947   {
1948     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1949     {
1950       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1951       {
1952         \exp_not:V \l__zrefclever_listsep_tl
1953         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1954       }
1955     }
1956   }
1957   {1}
1958   {
1959     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1960     {
1961       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1962       {
1963         \exp_not:V \l__zrefclever_listsep_tl
1964         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1965       }
1966       \exp_not:V \l__zrefclever_listsep_tl
1967       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1968     }
1969   }
1970 }
1971 {
1972   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1973   {
1974     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1975     {
1976       \exp_not:V \l__zrefclever_listsep_tl
1977       \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1978     }
1979     \exp_not:V \l__zrefclever_rangesep_tl
1980     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1981   }
1982 }
1983 }
1984 % Reset counters.
1985 \int_zero:N \l__zrefclever_range_count_int
1986 \int_zero:N \l__zrefclever_range_same_count_int
1987 }
1988 }
1989 % Step label counter for next iteration.
1990 \int_incr:N \l__zrefclever_label_count_int
1991 }

```

(End definition for `__zrefclever_typeset_refs_aux_not_last_of_type:.`)

Aux functions

`_zrefclever_get_ref:n` Auxiliary function to `_zrefclever_typeset_refs:.` Handles a complete “ref-block”, including “pre” and “pos” elements, and *hyperlinking*. It does not handle the reference type “name”, for that use `_zrefclever_get_ref_first:.` It should get the reference with `\zref@extractdefault` as usual but, if the reference is not available, should put `\zref@default` on the stream protected, so that it can be accumulated in the queue. `\hyperlink` must also be protected from expansion for the same reason.

```

1992 \cs_new:Npn \_zrefclever_get_ref:n #1
1993 {
1994   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
1995   {
1996     \bool_if:nTF
1997       { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
1998       {
1999         \exp_not:N \group_begin:
2000         \exp_not:V \l__zrefclever_reffont_out_tl
2001         \exp_not:V \l__zrefclever_refpre_out_tl
2002         \exp_not:N \group_begin:
2003         \exp_not:V \l__zrefclever_reffont_in_tl
2004         % It's two '@s', but escaped for DocStrip.
2005         \exp_not:N \hyper@@link
2006         {
2007           \zref@ifrefcontainsprop {#1} { urluse }
2008           { \zref@extractdefault {#1} { urluse } {} }
2009           { \zref@extractdefault {#1} { url } {} }
2010         }
2011         { \zref@extractdefault {#1} { anchor } {} }
2012         {
2013           \exp_not:V \l__zrefclever_refpre_in_tl
2014           \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2015           \exp_not:V \l__zrefclever_refpos_in_tl
2016         }
2017         \exp_not:N \group_end:
2018         \exp_not:V \l__zrefclever_refpos_out_tl
2019         \exp_not:N \group_end:
2020       }
2021       {
2022         \exp_not:N \group_begin:
2023         \exp_not:V \l__zrefclever_reffont_out_tl
2024         \exp_not:V \l__zrefclever_refpre_out_tl
2025         \exp_not:N \group_begin:
2026         \exp_not:V \l__zrefclever_reffont_in_tl
2027         \exp_not:V \l__zrefclever_refpre_in_tl
2028         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2029         \exp_not:V \l__zrefclever_refpos_in_tl
2030         \exp_not:N \group_end:
2031         \exp_not:V \l__zrefclever_refpos_out_tl
2032         \exp_not:N \group_end:
2033       }
2034     }
2035     { \exp_not:N \zref@default }
2036   }
2037 \cs_generate_variant:Nn \_zrefclever_get_ref:n { V }

```

(End definition for _zrefclever_get_ref:n.)

_zrefclever_type_name_setup: Auxiliary function to _zrefclever_typeset_refs:. Sets the type name variable \l_zrefclever_type_name_tl. When it cannot be found, clears it.

```

2038 \cs_new_protected:Npn \_zrefclever_type_name_setup:
2039 {
2040   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
2041   { \tl_clear:N \l\_zrefclever_type_name_tl }
2042   {
2043     \tl_if_empty:nTF \l\_zrefclever_type_first_label_type_tl
2044     { \tl_clear:N \l\_zrefclever_type_name_tl }
2045     {

```

Determine whether we should use capitalization, abbreviation, and plural.

```

2046   \bool_lazy_or:nnTF
2047   { \l\_zrefclever_capitalize_bool }
2048   {
2049     \l\_zrefclever_capitalize_first_bool &&
2050     \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 }
2051   }
2052   { \tl_set:Nn \l\_zrefclever_name_format_tl {Name} }
2053   { \tl_set:Nn \l\_zrefclever_name_format_tl {name} }
2054   % If the queue is empty, we have a singular, otherwise, plural.
2055   \tl_if_empty:NTF \l\_zrefclever_typeset_queue_curr_tl
2056   { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -sg } }
2057   { \tl_put_right:Nn \l\_zrefclever_name_format_tl { -pl } }
2058   \bool_lazy_and:nnTF
2059   { \l\_zrefclever_abbrev_bool }
2060   {
2061     ! \int_compare_p:nNn { \l\_zrefclever_type_count_int } = { 0 } ||
2062     ! \l\_zrefclever_noabbrev_first_bool
2063   }
2064   {
2065     \tl_set:NV \l\_zrefclever_name_format_fallback_tl \l\_zrefclever_name_format
2066     \tl_put_right:Nn \l\_zrefclever_name_format_tl { -ab }
2067   }
2068   { \tl_clear:N \l\_zrefclever_name_format_fallback_tl }
2069
2070   \tl_if_empty:NTF \l\_zrefclever_name_format_fallback_tl
2071   {
2072     \prop_get:cVNF
2073     { \l\_zrefclever_type_ \l\_zrefclever_type_first_label_type_tl _options_pro
2074       \l\_zrefclever_name_format_tl
2075       \l\_zrefclever_type_name_tl
2076     }
2077     \_zrefclever_get_type_transl:xxxNF
2078     { \l\_zrefclever_ref_language_tl }
2079     { \l\_zrefclever_type_first_label_type_tl }
2080     { \l\_zrefclever_name_format_tl }
2081     \l\_zrefclever_type_name_tl
2082     {
2083       \tl_clear:N \l\_zrefclever_type_name_tl
2084       \msg_warning:nxx { zref-clever } { missing-name }
2085       { \l\_zrefclever_type_first_label_type_tl }

```

```

2086         }
2087     }
2088 }
2089 {
2090     \prop_get:cVNF
2091     { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2092     \l__zrefclever_name_format_tl
2093     \l__zrefclever_type_name_tl
2094     {
2095         \prop_get:cVNF
2096         { l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options
2097         \l__zrefclever_name_format_fallback_tl
2098         \l__zrefclever_type_name_tl
2099         {
2100             \__zrefclever_get_type_transl:xxxNF
2101             { \l__zrefclever_ref_language_tl }
2102             { \l__zrefclever_type_first_label_type_tl }
2103             { \l__zrefclever_name_format_tl }
2104             \l__zrefclever_type_name_tl
2105             {
2106                 \__zrefclever_get_type_transl:xxxNF
2107                 { \l__zrefclever_ref_language_tl }
2108                 { \l__zrefclever_type_first_label_type_tl }
2109                 { \l__zrefclever_name_format_fallback_tl }
2110                 \l__zrefclever_type_name_tl
2111                 {
2112                     \tl_clear:N \l__zrefclever_type_name_tl
2113                     \msg_warning:nxx { zref-clever } { missing-name }
2114                     { \l__zrefclever_type_first_label_type_tl }
2115                 }
2116             }
2117         }
2118     }
2119 }
2120 }
2121 }

```

Signal whether the type name is to be included in the hyperlink or not.

```

2122 \bool_lazy_any:nTF
2123 {
2124     { ! \l__zrefclever_use_hyperref_bool }
2125     { \l__zrefclever_link_star_bool }
2126     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2127     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2128 }
2129 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2130 {
2131     \bool_lazy_any:nTF
2132     {
2133         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2134         {
2135             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2136             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2137         }
2138     }

```

```

2139         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2140         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2141         \l__zrefclever_typeset_last_bool &&
2142         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2143     }
2144 }
2145 { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2146 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2147 }
2148 }

```

(End definition for __zrefclever_type_name_setup:.)

__zrefclever_get_ref_first: Auxiliary function to __zrefclever_typeset_refs:. Handles a complete “ref-block”, including “pre” and “pos” elements, *hyperlinking*, and the reference type “name”. For use on the first reference of each type.

```

2149 \cs_new:Npn \__zrefclever_get_ref_first:
2150 {
2151     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2152     { \exp_not:N \zref@default }
2153     {
2154         \bool_if:NTF \l__zrefclever_name_in_link_bool
2155         {
2156             \zref@ifrefcontainsprop
2157             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2158             {
2159                 % It's two '@s', but escaped for DocStrip.
2160                 \exp_not:N \hyper@@link
2161                 {
2162                     \zref@ifrefcontainsprop
2163                     { \l__zrefclever_type_first_label_tl } { urluse }
2164                     {
2165                         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2166                         { urluse } {}
2167                     }
2168                     {
2169                         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2170                         { url } {}
2171                     }
2172                 }
2173             }
2174             \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2175             { anchor } {}
2176         }
2177         {
2178             \exp_not:N \group_begin:
2179             \exp_not:V \l__zrefclever_namefont_tl
2180             \exp_not:V \l__zrefclever_type_name_tl
2181             \exp_not:N \group_end:
2182             \exp_not:V \l__zrefclever_namesep_tl
2183             \exp_not:N \group_begin:
2184             \exp_not:V \l__zrefclever_reffont_out_tl
2185             \exp_not:V \l__zrefclever_refpre_out_tl
2186             \exp_not:N \group_end:

```

```

2187         \exp_not:V \l__zrefclever_reffont_in_tl
2188         \exp_not:V \l__zrefclever_refpre_in_tl
2189         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2190         { \l__zrefclever_ref_property_tl } {}
2191         \exp_not:V \l__zrefclever_refpos_in_tl
2192         \exp_not:N \group_end:
2193         % hyperlink makes it's own group, we'd like to close the
2194         % 'refpre-out' group after 'refpos-out', but... we close
2195         % it here, and give the trailing 'refpos-out' its own
2196         % group. This will result that formatting given to
2197         % 'refpre-out' will not reach 'refpos-out', but I see no
2198         % alternative, and this has to be handled specially.
2199         \exp_not:N \group_end:
2200     }
2201     \exp_not:N \group_begin:
2202     % Ditto: special treatment.
2203     \exp_not:V \l__zrefclever_reffont_out_tl
2204     \exp_not:V \l__zrefclever_refpos_out_tl
2205     \exp_not:N \group_end:
2206 }
2207 {
2208     \exp_not:N \group_begin:
2209     \exp_not:V \l__zrefclever_namefont_tl
2210     \exp_not:V \l__zrefclever_type_name_tl
2211     \exp_not:N \group_end:
2212     \exp_not:V \l__zrefclever_namesep_tl
2213     \exp_not:N \zref@default
2214 }
2215 }
2216 {
2217     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2218     {
2219         \exp_not:N \zref@default
2220         \exp_not:V \l__zrefclever_namesep_tl
2221     }
2222     {
2223         \exp_not:N \group_begin:
2224         \exp_not:V \l__zrefclever_namefont_tl
2225         \exp_not:V \l__zrefclever_type_name_tl
2226         \exp_not:N \group_end:
2227         \exp_not:V \l__zrefclever_namesep_tl
2228     }
2229     \zref@ifrefcontainsprop
2230     { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2231     {
2232         \bool_if:nTF
2233         {
2234             \l__zrefclever_use_hyperref_bool &&
2235             ! \l__zrefclever_link_star_bool
2236         }
2237         {
2238             \exp_not:N \group_begin:
2239             \exp_not:V \l__zrefclever_reffont_out_tl
2240             \exp_not:V \l__zrefclever_refpre_out_tl

```

```

2241 \exp_not:N \group_begin:
2242 \exp_not:V \l__zrefclever_reffont_in_tl
2243 % It's two '@s', but escaped for DocStrip.
2244 \exp_not:N \hyper@@link
2245 {
2246   \zref@ifrefcontainsprop
2247     { \l__zrefclever_type_first_label_tl } { urluse }
2248     {
2249       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2250       { urluse } {}
2251     }
2252     {
2253       \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2254       { url } {}
2255     }
2256   }
2257 {
2258   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2259   { anchor } {}
2260 }
2261 {
2262   \exp_not:V \l__zrefclever_refpre_in_tl
2263   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2264   { \l__zrefclever_ref_property_tl } {}
2265   \exp_not:V \l__zrefclever_refpos_in_tl
2266 }
2267 \exp_not:N \group_end:
2268 \exp_not:V \l__zrefclever_refpos_out_tl
2269 \exp_not:N \group_end:
2270 }
2271 {
2272   \exp_not:N \group_begin:
2273   \exp_not:V \l__zrefclever_reffont_out_tl
2274   \exp_not:V \l__zrefclever_refpre_out_tl
2275   \exp_not:N \group_begin:
2276   \exp_not:V \l__zrefclever_reffont_in_tl
2277   \exp_not:V \l__zrefclever_refpre_in_tl
2278   \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2279   { \l__zrefclever_ref_property_tl } {}
2280   \exp_not:V \l__zrefclever_refpos_in_tl
2281   \exp_not:N \group_end:
2282   \exp_not:V \l__zrefclever_refpos_out_tl
2283   \exp_not:N \group_end:
2284 }
2285 }
2286 { \exp_not:N \zref@default }
2287 }
2288 }
2289 }

```

(End definition for __zrefclever_get_ref_first:.)

__zrefclever_get_ref_string:nN

```

2290 % \Arg{option} \Arg{var to store result}

```



```

2291 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2292 {
2293   % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2294   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2295   {
2296     % If not found, try the type specific options.
2297     \bool_lazy_all:nTF
2298     {
2299       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2300       {
2301         \prop_if_exist_p:c
2302         { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2303       }
2304       {
2305         \prop_if_in_p:cn
2306         { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2307       }
2308     }
2309     {
2310       \prop_get:cnN
2311       { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2312     }
2313     {
2314       % If not found, try the type specific translations.
2315       \__zrefclever_get_type_transl:xxnNF
2316       { \l__zrefclever_ref_language_tl }
2317       { \l__zrefclever_label_type_a_tl }
2318       {#1} #2
2319       {
2320         % If not found, try default translations.
2321         \__zrefclever_get_default_transl:xxnNF
2322         { \l__zrefclever_ref_language_tl }
2323         {#1} #2
2324         {
2325           % If not found, try fallback.
2326           \__zrefclever_get_fallback_transl:nNF {#1} #2
2327           { \tl_clear:N #2 }
2328         }
2329       }
2330     }
2331   }
2332 }

```

(End definition for __zrefclever_get_ref_string:nN.)

__zrefclever_get_ref_font:nN

```

2333 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2334 {
2335   % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2336   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2337   {
2338     % If not found, try the type specific options.
2339     \bool_lazy_and:nnTF
2340     { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }

```

```

2341     {
2342       \prop_if_exist_p:c
2343       { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2344     }
2345     {
2346       \prop_get:cnNF
2347       { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2348       { \tl_clear:N #2 }
2349     }
2350     { \tl_clear:N #2 }
2351   }
2352 }

```

(End definition for __zrefclever_get_ref_font:nN.)

__zrefclever_labels_in_sequence:nn Sets \l__zrefclever_next_maybe_range_bool to true if label ‘1’ comes in immediate sequence from label ‘2’. And sets both \l__zrefclever_next_maybe_range_bool and \l__zrefclever_next_is_same_bool if the labels are the “same”.

```

2353 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2354 {
2355   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2356   {
2357     \exp_args:Nxx \tl_if_eq:nnT
2358     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2359     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2360     {
2361       \int_compare:nNnTF
2362       { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2363       =
2364       { \zref@extractdefault {#2} { zc@pgval } {-1} }
2365       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2366       {
2367         \int_compare:nNnT
2368         { \zref@extractdefault {#1} { zc@pgval } {-1} }
2369         =
2370         { \zref@extractdefault {#2} { zc@pgval } {-1} }
2371         {
2372           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2373           \bool_set_true:N \l__zrefclever_next_is_same_bool
2374         }
2375       }
2376     }
2377   }
2378   {
2379     \exp_args:Nxx \tl_if_eq:nnT
2380     { \zref@extractdefault {#1} { counter } { } }
2381     { \zref@extractdefault {#2} { counter } { } }
2382     {
2383       \exp_args:Nxx \tl_if_eq:nnT
2384       { \zref@extractdefault {#1} { zc@enclval } { } }
2385       { \zref@extractdefault {#2} { zc@enclval } { } }
2386       {
2387         \int_compare:nNnTF
2388         { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }

```

```

2389         =
2390         { \zref@extractdefault {#2} { zc@cntval } {-1} }
2391         { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2392         {
2393             \int_compare:nNnT
2394                 { \zref@extractdefault {#1} { zc@cntval } {-1} }
2395                 =
2396                 { \zref@extractdefault {#2} { zc@cntval } {-1} }
2397             {
2398                 \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2399                 \bool_set_true:N \l__zrefclever_next_is_same_bool
2400             }
2401         }
2402     }
2403 }
2404 }
2405 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

9 Special handling

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. It is not meant to be a “kitchen sink of workarounds”. Rather, I intend to keep this as lean as possible, trying to add things selectively when they are safe and reasonable. And, hopefully, doing so by proper setting of `zref-clever`’s options, not by messing with other packages’ code. In particular, I do not mean to compensate for “lack of support for `zref`” by individual packages here, unless there is really no alternative.

9.1 `\appendix`

Another relevant use case of the same general problem of different types for 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).

9.2 `\newtheorem`

9.3 `enumitem` package

TODO Option `counterresetby` should probably be extended for `enumitem`, conditioned on it being loaded.

```

2406 </package>

```

10 Dictionaries

10.1 English

```
2407 <package>\zcDeclareLanguage { english }
2408 <package>\zcDeclareLanguageAlias { american } { english }
2409 <package>\zcDeclareLanguageAlias { australian } { english }
2410 <package>\zcDeclareLanguageAlias { british } { english }
2411 <package>\zcDeclareLanguageAlias { canadian } { english }
2412 <package>\zcDeclareLanguageAlias { newzealand } { english }
2413 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2414 <package>\zcDeclareLanguageAlias { USenglish } { english }
2415 <*dict-english>

2416 namesep = {\nobreakspace} ,
2417 pairsep = {\and\nobreakspace} ,
2418 listsep = {,~} ,
2419 lastsep = {\and\nobreakspace} ,
2420 tpairsep = {\and\nobreakspace} ,
2421 tlistsep = {,~} ,
2422 tlastsep = {,~\and\nobreakspace} ,
2423 notesep = {~} ,
2424 rangesep = {\to\nobreakspace} ,
2425
2426 type = part ,
2427   Name-sg = Part ,
2428   name-sg = part ,
2429   Name-pl = Parts ,
2430   name-pl = parts ,
2431
2432 type = chapter ,
2433   Name-sg = Chapter ,
2434   name-sg = chapter ,
2435   Name-pl = Chapters ,
2436   name-pl = chapters ,
2437
2438 type = section ,
2439   Name-sg = Section ,
2440   name-sg = section ,
2441   Name-pl = Sections ,
2442   name-pl = sections ,
2443
2444 type = paragraph ,
2445   Name-sg = Paragraph ,
2446   name-sg = paragraph ,
2447   Name-pl = Paragraphs ,
2448   name-pl = paragraphs ,
2449   Name-sg-ab = Par. ,
2450   name-sg-ab = par. ,
2451   Name-pl-ab = Par. ,
2452   name-pl-ab = par. ,
2453
2454 type = appendix ,
2455   Name-sg = Appendix ,
2456   name-sg = appendix ,
```

```

2457     Name-pl = Appendices ,
2458     name-pl = appendices ,
2459
2460     type = page ,
2461     Name-sg = Page ,
2462     name-sg = page ,
2463     Name-pl = Pages ,
2464     name-pl = pages ,
2465     name-sg-ab = p. ,
2466     name-pl-ab = pp. ,
2467
2468     type = line ,
2469     Name-sg = Line ,
2470     name-sg = line ,
2471     Name-pl = Lines ,
2472     name-pl = lines ,
2473
2474     type = figure ,
2475     Name-sg = Figure ,
2476     name-sg = figure ,
2477     Name-pl = Figures ,
2478     name-pl = figures ,
2479     Name-sg-ab = Fig. ,
2480     name-sg-ab = fig. ,
2481     Name-pl-ab = Figs. ,
2482     name-pl-ab = figs. ,
2483
2484     type = table ,
2485     Name-sg = Table ,
2486     name-sg = table ,
2487     Name-pl = Tables ,
2488     name-pl = tables ,
2489
2490     type = item ,
2491     Name-sg = Item ,
2492     name-sg = item ,
2493     Name-pl = Items ,
2494     name-pl = items ,
2495
2496     type = footnote ,
2497     Name-sg = Footnote ,
2498     name-sg = footnote ,
2499     Name-pl = Footnotes ,
2500     name-pl = footnotes ,
2501
2502     type = note ,
2503     Name-sg = Note ,
2504     name-sg = note ,
2505     Name-pl = Notes ,
2506     name-pl = notes ,
2507
2508     type = equation ,
2509     Name-sg = Equation ,
2510     name-sg = equation ,

```

```

2511 Name-pl = Equations ,
2512 name-pl = equations ,
2513 Name-sg-ab = Eq. ,
2514 name-sg-ab = eq. ,
2515 Name-pl-ab = Eqs. ,
2516 name-pl-ab = eqs. ,
2517 refpre-in = {()} ,
2518 refpos-in = {} ,
2519
2520 type = theorem ,
2521 Name-sg = Theorem ,
2522 name-sg = theorem ,
2523 Name-pl = Theorems ,
2524 name-pl = theorems ,
2525
2526 type = lemma ,
2527 Name-sg = Lemma ,
2528 name-sg = lemma ,
2529 Name-pl = Lemmas ,
2530 name-pl = lemmas ,
2531
2532 type = corollary ,
2533 Name-sg = Corollary ,
2534 name-sg = corollary ,
2535 Name-pl = Corollaries ,
2536 name-pl = corollaries ,
2537
2538 type = proposition ,
2539 Name-sg = Proposition ,
2540 name-sg = proposition ,
2541 Name-pl = Propositions ,
2542 name-pl = propositions ,
2543
2544 type = definition ,
2545 Name-sg = Definition ,
2546 name-sg = definition ,
2547 Name-pl = Definitions ,
2548 name-pl = definitions ,
2549
2550 type = proof ,
2551 Name-sg = Proof ,
2552 name-sg = proof ,
2553 Name-pl = Proofs ,
2554 name-pl = proofs ,
2555
2556 type = result ,
2557 Name-sg = Result ,
2558 name-sg = result ,
2559 Name-pl = Results ,
2560 name-pl = results ,
2561
2562 type = example ,
2563 Name-sg = Example ,
2564 name-sg = example ,

```

```

2565   Name-pl = Examples ,
2566   name-pl = examples ,
2567
2568   type = remark ,
2569   Name-sg = Remark ,
2570   name-sg = remark ,
2571   Name-pl = Remarks ,
2572   name-pl = remarks ,
2573
2574   type = algorithm ,
2575   Name-sg = Algorithm ,
2576   name-sg = algorithm ,
2577   Name-pl = Algorithms ,
2578   name-pl = algorithms ,
2579
2580   type = listing ,
2581   Name-sg = Listing ,
2582   name-sg = listing ,
2583   Name-pl = Listings ,
2584   name-pl = listings ,
2585
2586   type = exercise ,
2587   Name-sg = Exercise ,
2588   name-sg = exercise ,
2589   Name-pl = Exercises ,
2590   name-pl = exercises ,
2591
2592   type = solution ,
2593   Name-sg = Solution ,
2594   name-sg = solution ,
2595   Name-pl = Solutions ,
2596   name-pl = solutions ,
2597   </dict-english>

```

10.2 German

```

2598 <package>\zcDeclareLanguage { german }
2599 <package>\zcDeclareLanguageAlias { austrian      } { german }
2600 <package>\zcDeclareLanguageAlias { germanb       } { german }
2601 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2602 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2603 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2604 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2605 <dict-german>

2606 namesep = {\nobreakspace} ,
2607 pairsep  = {\und\nobreakspace} ,
2608 listsep  = {,~} ,
2609 lastsep  = {\und\nobreakspace} ,
2610 tpairsep = {\und\nobreakspace} ,
2611 tlistsep = {,~} ,
2612 tlastsep = {\und\nobreakspace} ,
2613 notesep  = {~} ,
2614 rangesep = {\bis\nobreakspace} ,
2615

```

```

2616 type = part ,
2617     Name-sg = Teil ,
2618     name-sg = Teil ,
2619     Name-pl = Teile ,
2620     name-pl = Teile ,
2621
2622 type = chapter ,
2623     Name-sg = Kapitel ,
2624     name-sg = Kapitel ,
2625     Name-pl = Kapitel ,
2626     name-pl = Kapitel ,
2627
2628 type = section ,
2629     Name-sg = Abschnitt ,
2630     name-sg = Abschnitt ,
2631     Name-pl = Abschnitte ,
2632     name-pl = Abschnitte ,
2633
2634 type = paragraph ,
2635     Name-sg = Absatz ,
2636     name-sg = Absatz ,
2637     Name-pl = Absätze ,
2638     name-pl = Absätze ,
2639
2640 type = appendix ,
2641     Name-sg = Anhang ,
2642     name-sg = Anhang ,
2643     Name-pl = Anhänge ,
2644     name-pl = Anhänge ,
2645
2646 type = page ,
2647     Name-sg = Seite ,
2648     name-sg = Seite ,
2649     Name-pl = Seiten ,
2650     name-pl = Seiten ,
2651
2652 type = line ,
2653     Name-sg = Zeile ,
2654     name-sg = Zeile ,
2655     Name-pl = Zeilen ,
2656     name-pl = Zeilen ,
2657
2658 type = figure ,
2659     Name-sg = Abbildung ,
2660     name-sg = Abbildung ,
2661     Name-pl = Abbildungen ,
2662     name-pl = Abbildungen ,
2663     Name-sg-ab = Abb. ,
2664     name-sg-ab = Abb. ,
2665     Name-pl-ab = Abb. ,
2666     name-pl-ab = Abb. ,
2667
2668 type = table ,
2669     Name-sg = Tabelle ,

```



```

2670     name-sg = Tabelle ,
2671     Name-pl = Tabellen ,
2672     name-pl = Tabellen ,
2673
2674 type = item ,
2675     Name-sg = Punkt ,
2676     name-sg = Punkt ,
2677     Name-pl = Punkte ,
2678     name-pl = Punkte ,
2679
2680 type = footnote ,
2681     Name-sg = Fußnote ,
2682     name-sg = Fußnote ,
2683     Name-pl = Fußnoten ,
2684     name-pl = Fußnoten ,
2685
2686 type = note ,
2687     Name-sg = Anmerkung ,
2688     name-sg = Anmerkung ,
2689     Name-pl = Anmerkungen ,
2690     name-pl = Anmerkungen ,
2691
2692 type = equation ,
2693     Name-sg = Gleichung ,
2694     name-sg = Gleichung ,
2695     Name-pl = Gleichungen ,
2696     name-pl = Gleichungen ,
2697     refpre-in = {() ,
2698     refpos-in = {} } ,
2699
2700 type = theorem ,
2701     Name-sg = Theorem ,
2702     name-sg = Theorem ,
2703     Name-pl = Theoreme ,
2704     name-pl = Theoreme ,
2705
2706 type = lemma ,
2707     Name-sg = Lemma ,
2708     name-sg = Lemma ,
2709     Name-pl = Lemmata ,
2710     name-pl = Lemmata ,
2711
2712 type = corollary ,
2713     Name-sg = Korollar ,
2714     name-sg = Korollar ,
2715     Name-pl = Korollare ,
2716     name-pl = Korollare ,
2717
2718 type = proposition ,
2719     Name-sg = Satz ,
2720     name-sg = Satz ,
2721     Name-pl = Sätze ,
2722     name-pl = Sätze ,
2723

```

```

2724 type = definition ,
2725     Name-sg = Definition ,
2726     name-sg = Definition ,
2727     Name-pl = Definitionen ,
2728     name-pl = Definitionen ,
2729
2730 type = proof ,
2731     Name-sg = Beweis ,
2732     name-sg = Beweis ,
2733     Name-pl = Beweise ,
2734     name-pl = Beweise ,
2735
2736 type = result ,
2737     Name-sg = Ergebnis ,
2738     name-sg = Ergebnis ,
2739     Name-pl = Ergebnisse ,
2740     name-pl = Ergebnisse ,
2741
2742 type = example ,
2743     Name-sg = Beispiel ,
2744     name-sg = Beispiel ,
2745     Name-pl = Beispiele ,
2746     name-pl = Beispiele ,
2747
2748 type = remark ,
2749     Name-sg = Bemerkung ,
2750     name-sg = Bemerkung ,
2751     Name-pl = Bemerkungen ,
2752     name-pl = Bemerkungen ,
2753
2754 type = algorithm ,
2755     Name-sg = Algorithmus ,
2756     name-sg = Algorithmus ,
2757     Name-pl = Algorithmen ,
2758     name-pl = Algorithmen ,
2759
2760 type = listing ,
2761     Name-sg = Listing , % CHECK
2762     name-sg = Listing , % CHECK
2763     Name-pl = Listings , % CHECK
2764     name-pl = Listings , % CHECK
2765
2766 type = exercise ,
2767     Name-sg = Übungsaufgabe ,
2768     name-sg = Übungsaufgabe ,
2769     Name-pl = Übungsaufgaben ,
2770     name-pl = Übungsaufgaben ,
2771
2772 type = solution ,
2773     Name-sg = Lösung ,
2774     name-sg = Lösung ,
2775     Name-pl = Lösungen ,
2776     name-pl = Lösungen ,
2777 </dict-german>

```

10.3 French

```
2778 <package>\zcDeclareLanguage { french }
2779 <package>\zcDeclareLanguageAlias { acadian } { french }
2780 <package>\zcDeclareLanguageAlias { canadien } { french }
2781 <package>\zcDeclareLanguageAlias { francais } { french }
2782 <package>\zcDeclareLanguageAlias { frenchb } { french }
2783 <*dict-french>

2784 namesep = {\nobreakspace} ,
2785 pairsep = {\~et\nobreakspace} ,
2786 listsep = {,~} ,
2787 lastsep = {\~et\nobreakspace} ,
2788 tpairsep = {\~et\nobreakspace} ,
2789 tlistsep = {,~} ,
2790 tlastsep = {\~et\nobreakspace} ,
2791 notesep = {\~} ,
2792 rangesep = {\~à\nobreakspace} ,
2793
2794 type = part ,
2795   Name-sg = Partie ,
2796   name-sg = partie ,
2797   Name-pl = Parties ,
2798   name-pl = parties ,
2799
2800 type = chapter ,
2801   Name-sg = Chapitre ,
2802   name-sg = chapitre ,
2803   Name-pl = Chapitres ,
2804   name-pl = chapitres ,
2805
2806 type = section ,
2807   Name-sg = Section ,
2808   name-sg = section ,
2809   Name-pl = Sections ,
2810   name-pl = sections ,
2811
2812 type = paragraph ,
2813   Name-sg = Paragraphe ,
2814   name-sg = paragraphe ,
2815   Name-pl = Paragraphes ,
2816   name-pl = paragraphes ,
2817
2818 type = appendix ,
2819   Name-sg = Annexe ,
2820   name-sg = annexe ,
2821   Name-pl = Annexes ,
2822   name-pl = annexes ,
2823
2824 type = page ,
2825   Name-sg = Page ,
2826   name-sg = page ,
2827   Name-pl = Pages ,
2828   name-pl = pages ,
2829
```

```

2830 type = line ,
2831     Name-sg = Ligne ,
2832     name-sg = ligne ,
2833     Name-pl = Lignes ,
2834     name-pl = lignes ,
2835
2836 type = figure ,
2837     Name-sg = Figure ,
2838     name-sg = figure ,
2839     Name-pl = Figures ,
2840     name-pl = figures ,
2841
2842 type = table ,
2843     Name-sg = Table ,
2844     name-sg = table ,
2845     Name-pl = Tables ,
2846     name-pl = tables ,
2847
2848 type = item ,
2849     Name-sg = Point ,
2850     name-sg = point ,
2851     Name-pl = Points ,
2852     name-pl = points ,
2853
2854 type = footnote ,
2855     Name-sg = Note ,
2856     name-sg = note ,
2857     Name-pl = Notes ,
2858     name-pl = notes ,
2859
2860 type = note ,
2861     Name-sg = Note ,
2862     name-sg = note ,
2863     Name-pl = Notes ,
2864     name-pl = notes ,
2865
2866 type = equation ,
2867     Name-sg = Équation ,
2868     name-sg = équation ,
2869     Name-pl = Équations ,
2870     name-pl = équations ,
2871     refpre-in = {()} ,
2872     refpos-in = {} } ,
2873
2874 type = theorem ,
2875     Name-sg = Théorème ,
2876     name-sg = théorème ,
2877     Name-pl = Théorèmes ,
2878     name-pl = théorèmes ,
2879
2880 type = lemma ,
2881     Name-sg = Lemme ,
2882     name-sg = lemme ,
2883     Name-pl = Lemmes ,

```

```

2884     name-pl = lemmes ,
2885
2886 type = corollary ,
2887     Name-sg = Corollaire ,
2888     name-sg = corollaire ,
2889     Name-pl = Corollaires ,
2890     name-pl = corollaires ,
2891
2892 type = proposition ,
2893     Name-sg = Proposition ,
2894     name-sg = proposition ,
2895     Name-pl = Propositions ,
2896     name-pl = propositions ,
2897
2898 type = definition ,
2899     Name-sg = Définition ,
2900     name-sg = définition ,
2901     Name-pl = Définitions ,
2902     name-pl = définitions ,
2903
2904 type = proof ,
2905     Name-sg = Démonstration ,
2906     name-sg = démonstration ,
2907     Name-pl = Démonstrations ,
2908     name-pl = démonstrations ,
2909
2910 type = result ,
2911     Name-sg = Résultat ,
2912     name-sg = résultat ,
2913     Name-pl = Résultats ,
2914     name-pl = résultats ,
2915
2916 type = example ,
2917     Name-sg = Exemple ,
2918     name-sg = exemple ,
2919     Name-pl = Exemples ,
2920     name-pl = exemples ,
2921
2922 type = remark ,
2923     Name-sg = Remarque ,
2924     name-sg = remarque ,
2925     Name-pl = Remarques ,
2926     name-pl = remarques ,
2927
2928 type = algorithm ,
2929     Name-sg = Algorithme ,
2930     name-sg = algorithme ,
2931     Name-pl = Algorithmes ,
2932     name-pl = algorithmes ,
2933
2934 type = listing ,
2935     Name-sg = Liste ,
2936     name-sg = liste ,
2937     Name-pl = Listes ,

```

```

2938     name-pl = listes ,
2939
2940 type = exercise ,
2941     Name-sg = Exercice ,
2942     name-sg = exercice ,
2943     Name-pl = Exercices ,
2944     name-pl = exercices ,
2945
2946 type = solution ,
2947     Name-sg = Solution ,
2948     name-sg = solution ,
2949     Name-pl = Solutions ,
2950     name-pl = solutions ,
2951 </dict-french>

```

10.4 Portuguese

```

2952 <package>\zcDeclareLanguage { portuguese }
2953 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
2954 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
2955 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
2956 <*dict-portuguese>

2957 namesep = {\nobreakspace} ,
2958 pairsep = {\sim\nobreakspace} ,
2959 listsep = {,~} ,
2960 lastsep = {\sim\nobreakspace} ,
2961 tpairsep = {\sim\nobreakspace} ,
2962 tlistsep = {,~} ,
2963 tlastsep = {\sim\nobreakspace} ,
2964 notesep = {\sim} ,
2965 rangesep = {\sim\nobreakspace} ,
2966
2967 type = part ,
2968     Name-sg = Parte ,
2969     name-sg = parte ,
2970     Name-pl = Partes ,
2971     name-pl = partes ,
2972
2973 type = chapter ,
2974     Name-sg = Capítulo ,
2975     name-sg = capítulo ,
2976     Name-pl = Capítulos ,
2977     name-pl = capítulos ,
2978
2979 type = section ,
2980     Name-sg = Seção ,
2981     name-sg = seção ,
2982     Name-pl = Seções ,
2983     name-pl = seções ,
2984
2985 type = paragraph ,
2986     Name-sg = Parágrafo ,
2987     name-sg = parágrafo ,
2988     Name-pl = Parágrafos ,

```

```

2989     name-pl = parágrafos ,
2990     Name-sg-ab = Par. ,
2991     name-sg-ab = par. ,
2992     Name-pl-ab = Par. ,
2993     name-pl-ab = par. ,
2994
2995 type = appendix ,
2996     Name-sg = Apêndice ,
2997     name-sg = apêndice ,
2998     Name-pl = Apêndices ,
2999     name-pl = apêndices ,
3000
3001 type = page ,
3002     Name-sg = Página ,
3003     name-sg = página ,
3004     Name-pl = Páginas ,
3005     name-pl = páginas ,
3006     name-sg-ab = p. ,
3007     name-pl-ab = pp. ,
3008
3009 type = line ,
3010     Name-sg = Linha ,
3011     name-sg = linha ,
3012     Name-pl = Linhas ,
3013     name-pl = linhas ,
3014
3015 type = figure ,
3016     Name-sg = Figura ,
3017     name-sg = figura ,
3018     Name-pl = Figuras ,
3019     name-pl = figuras ,
3020     Name-sg-ab = Fig. ,
3021     name-sg-ab = fig. ,
3022     Name-pl-ab = Figs. ,
3023     name-pl-ab = figs. ,
3024
3025 type = table ,
3026     Name-sg = Tabela ,
3027     name-sg = tabela ,
3028     Name-pl = Tabelas ,
3029     name-pl = tabelas ,
3030
3031 type = item ,
3032     Name-sg = Item ,
3033     name-sg = item ,
3034     Name-pl = Itens ,
3035     name-pl = itens ,
3036
3037 type = footnote ,
3038     Name-sg = Nota ,
3039     name-sg = nota ,
3040     Name-pl = Notas ,
3041     name-pl = notas ,
3042

```

```

3043 type = note ,
3044     Name-sg = Nota ,
3045     name-sg = nota ,
3046     Name-pl = Notas ,
3047     name-pl = notas ,
3048
3049 type = equation ,
3050     Name-sg = Equação ,
3051     name-sg = equação ,
3052     Name-pl = Equações ,
3053     name-pl = equações ,
3054     Name-sg-ab = Eq. ,
3055     name-sg-ab = eq. ,
3056     Name-pl-ab = Eqs. ,
3057     name-pl-ab = eqs. ,
3058     refpre-in = {()} ,
3059     refpos-in = {} ,
3060
3061 type = theorem ,
3062     Name-sg = Teorema ,
3063     name-sg = teorema ,
3064     Name-pl = Teoremas ,
3065     name-pl = teoremas ,
3066
3067 type = lemma ,
3068     Name-sg = Lema ,
3069     name-sg = lema ,
3070     Name-pl = Lemas ,
3071     name-pl = lemas ,
3072
3073 type = corollary ,
3074     Name-sg = Corolário ,
3075     name-sg = corolário ,
3076     Name-pl = Corolários ,
3077     name-pl = corolários ,
3078
3079 type = proposition ,
3080     Name-sg = Proposição ,
3081     name-sg = proposição ,
3082     Name-pl = Proposições ,
3083     name-pl = proposições ,
3084
3085 type = definition ,
3086     Name-sg = Definição ,
3087     name-sg = definição ,
3088     Name-pl = Definições ,
3089     name-pl = definições ,
3090
3091 type = proof ,
3092     Name-sg = Demonstração ,
3093     name-sg = demonstração ,
3094     Name-pl = Demonstrações ,
3095     name-pl = demonstrações ,
3096

```



```

3097 type = result ,
3098   Name-sg = Resultado ,
3099   name-sg = resultado ,
3100   Name-pl = Resultados ,
3101   name-pl = resultados ,
3102
3103 type = example ,
3104   Name-sg = Exemplo ,
3105   name-sg = exemplo ,
3106   Name-pl = Exemplos ,
3107   name-pl = exemplos ,
3108
3109 type = remark ,
3110   Name-sg = Observação ,
3111   name-sg = observação ,
3112   Name-pl = Observações ,
3113   name-pl = observações ,
3114
3115 type = algorithm ,
3116   Name-sg = Algoritmo ,
3117   name-sg = algoritmo ,
3118   Name-pl = Algoritmos ,
3119   name-pl = algoritmos ,
3120
3121 type = listing ,
3122   Name-sg = Listagem ,
3123   name-sg = listagem ,
3124   Name-pl = Listagens ,
3125   name-pl = listagens ,
3126
3127 type = exercise ,
3128   Name-sg = Exercício ,
3129   name-sg = exercício ,
3130   Name-pl = Exercícios ,
3131   name-pl = exercícios ,
3132
3133 type = solution ,
3134   Name-sg = Solução ,
3135   name-sg = solução ,
3136   Name-pl = Soluções ,
3137   name-pl = soluções ,
3138 </dict-portuguese>

```

10.5 Spanish

```

3139 <package>\zcDeclareLanguage { spanish }
3140 <*dict-spanish>
3141 namesep = {\nobreakspace} ,
3142 pairsep = {\sim\nobreakspace} ,
3143 listsep = {,~} ,
3144 lastsep = {\sim\nobreakspace} ,
3145 tpairsep = {\sim\nobreakspace} ,
3146 tlistsep = {,~} ,
3147 tlastsep = {\sim\nobreakspace} ,

```

```

3148 notesep = {\~} ,
3149 rangesep = {\~a\nobreakspace} ,
3150
3151 type = part ,
3152   Name-sg = Parte ,
3153   name-sg = parte ,
3154   Name-pl = Partes ,
3155   name-pl = partes ,
3156
3157 type = chapter ,
3158   Name-sg = Capítulo ,
3159   name-sg = capítulo ,
3160   Name-pl = Capítulos ,
3161   name-pl = capítulos ,
3162
3163 type = section ,
3164   Name-sg = Sección ,
3165   name-sg = sección ,
3166   Name-pl = Secciones ,
3167   name-pl = secciones ,
3168
3169 type = paragraph ,
3170   Name-sg = Párrafo ,
3171   name-sg = párrafo ,
3172   Name-pl = Párrafos ,
3173   name-pl = párrafos ,
3174
3175 type = appendix ,
3176   Name-sg = Apéndice ,
3177   name-sg = apéndice ,
3178   Name-pl = Apéndices ,
3179   name-pl = apéndices ,
3180
3181 type = page ,
3182   Name-sg = Página ,
3183   name-sg = página ,
3184   Name-pl = Páginas ,
3185   name-pl = páginas ,
3186
3187 type = line ,
3188   Name-sg = Línea ,
3189   name-sg = línea ,
3190   Name-pl = Líneas ,
3191   name-pl = líneas ,
3192
3193 type = figure ,
3194   Name-sg = Figura ,
3195   name-sg = figura ,
3196   Name-pl = Figuras ,
3197   name-pl = figuras ,
3198
3199 type = table ,
3200   Name-sg = Cuadro ,
3201   name-sg = cuadro ,

```

```

3202     Name-pl = Cuadros ,
3203     name-pl = cuadros ,
3204
3205 type = item ,
3206     Name-sg = Punto ,
3207     name-sg = punto ,
3208     Name-pl = Puntos ,
3209     name-pl = puntos ,
3210
3211 type = footnote ,
3212     Name-sg = Nota ,
3213     name-sg = nota ,
3214     Name-pl = Notas ,
3215     name-pl = notas ,
3216
3217 type = note ,
3218     Name-sg = Nota ,
3219     name-sg = nota ,
3220     Name-pl = Notas ,
3221     name-pl = notas ,
3222
3223 type = equation ,
3224     Name-sg = Ecuación ,
3225     name-sg = ecuación ,
3226     Name-pl = Ecuaciones ,
3227     name-pl = ecuaciones ,
3228     refpre-in = {(} ,
3229     refpos-in = {)} ,
3230
3231 type = theorem ,
3232     Name-sg = Teorema ,
3233     name-sg = teorema ,
3234     Name-pl = Teoremas ,
3235     name-pl = teoremas ,
3236
3237 type = lemma ,
3238     Name-sg = Lema ,
3239     name-sg = lema ,
3240     Name-pl = Lemas ,
3241     name-pl = lemas ,
3242
3243 type = corollary ,
3244     Name-sg = Corolario ,
3245     name-sg = corolario ,
3246     Name-pl = Corolarios ,
3247     name-pl = corolarios ,
3248
3249 type = proposition ,
3250     Name-sg = Proposición ,
3251     name-sg = proposición ,
3252     Name-pl = Proposiciones ,
3253     name-pl = proposiciones ,
3254
3255 type = definition ,

```

```

3256     Name-sg = Definición ,
3257     name-sg = definición ,
3258     Name-pl = Definiciones ,
3259     name-pl = definiciones ,
3260
3261     type = proof ,
3262     Name-sg = Demostración ,
3263     name-sg = demostración ,
3264     Name-pl = Demostraciones ,
3265     name-pl = demostraciones ,
3266
3267     type = result ,
3268     Name-sg = Resultado ,
3269     name-sg = resultado ,
3270     Name-pl = Resultados ,
3271     name-pl = resultados ,
3272
3273     type = example ,
3274     Name-sg = Ejemplo ,
3275     name-sg = ejemplo ,
3276     Name-pl = Ejemplos ,
3277     name-pl = ejemplos ,
3278
3279     type = remark ,
3280     Name-sg = Observación ,
3281     name-sg = observación ,
3282     Name-pl = Observaciones ,
3283     name-pl = observaciones ,
3284
3285     type = algorithm ,
3286     Name-sg = Algoritmo ,
3287     name-sg = algoritmo ,
3288     Name-pl = Algoritmos ,
3289     name-pl = algoritmos ,
3290
3291     type = listing ,
3292     Name-sg = Listado ,
3293     name-sg = listado ,
3294     Name-pl = Listados ,
3295     name-pl = listados ,
3296
3297     type = exercise ,
3298     Name-sg = Ejercicio ,
3299     name-sg = ejercicio ,
3300     Name-pl = Ejercicios ,
3301     name-pl = ejercicios ,
3302
3303     type = solution ,
3304     Name-sg = Solución ,
3305     name-sg = solución ,
3306     Name-pl = Soluciones ,
3307     name-pl = soluciones ,
3308 </dict-spanish>

```

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.

Symbols	
\\	103, 109, 120, 125, 126, 135, 145
A	
\AddToHook	91, 530, 545, 655, 723, 747, 776, 778, 829
\appendix	59
\appendixname	59
\Arg	2290
B	
\babelname	733
\babelprovide	21
bool commands:	
\bool_case_true:	2
\bool_if:NTF	284, 293, 659, 663, 1502, 1575, 1700, 1721, 1752, 1807, 1874, 1878, 1885, 1894, 1900, 2154
\bool_if:nTF	59, 1145, 1154, 1163, 1234, 1262, 1285, 1374, 1382, 1516, 1524, 1733, 1740, 1747, 1996, 2232
\bool_lazy_all:nTF	2297
\bool_lazy_and:nnTF	1034, 1049, 1849, 2058, 2339
\bool_lazy_any:nTF	2122, 2131
\bool_lazy_or:nnTF	1038, 2046
\bool_new:N	254, 566, 567, 592, 616, 625, 632, 633, 688, 689, 706, 707, 822, 823, 1060, 1077, 1414, 1415, 1426, 1427, 1428, 1447
\bool_set:Nn	1032
\bool_set_false:N	579, 583, 640, 649, 650, 665, 844, 1223, 1469, 1508, 1522, 1533, 1712, 1847, 1848, 2129, 2146
\bool_set_true:N	303, 573, 574, 578, 584, 639, 644, 645, 833, 838, 1246, 1257, 1274, 1280, 1297, 1303, 1329, 1341, 1476, 1503, 1509, 1513, 1534, 1537, 2145, 2365, 2372, 2373, 2391, 2398, 2399
\bool_until_do:Nn	1227, 1470
C	
clist commands:	
\clist_map_inline:nn	475
\counterwithin	4
\cs	1353, 1698, 2293, 2335
cs commands:	
\cs_generate_variant:Nn	55, 56, 298, 307, 950, 958, 1078, 2037
\cs_if_exist:NTF	39, 48, 69
\cs_new:Npn	37, 46, 57, 67, 78, 1992, 2149
\cs_new_protected:Npn	257, 300, 310, 318, 436, 945, 953, 1027, 1079, 1095, 1138, 1204, 1349, 1405, 1450, 1586, 1842, 2038, 2291, 2333, 2353
\cs_new_protected:Npx	90
\cs_set_eq:NN	94
E	
\endinput	12
exp commands:	
\exp_args:NNe	27
\exp_args:NNnx	247
\exp_args:NnV	276
\exp_args:NNx	95, 1271, 1294
\exp_args:Nnx	312
\exp_args:Nx	268
\exp_args:Nxx	1188, 1242, 2357, 2379, 2383
\exp_not:N	1754, 1757, 1777, 1780, 1783, 1999, 2002, 2005, 2017, 2019, 2022, 2025, 2030, 2032, 2035, 2152, 2160, 2178, 2181, 2183, 2186, 2192, 2199, 2201, 2205, 2208, 2211, 2213, 2219, 2223, 2226, 2238, 2241, 2244, 2267, 2269, 2272, 2275, 2281, 2283, 2286
\exp_not:n	1606, 1622, 1634, 1638, 1658, 1671, 1674, 1686, 1689, 1722, 1723, 1755, 1776, 1781, 1782, 1914, 1927, 1932, 1952, 1963, 1966, 1976, 1979, 2000, 2001, 2003, 2013, 2015, 2018, 2023, 2024, 2026, 2027, 2029, 2031, 2179, 2180, 2182, 2184, 2185, 2187, 2188, 2191, 2203, 2204, 2209, 2210, 2212, 2220, 2224, 2225, 2227, 2239, 2240, 2242, 2262, 2265, 2268, 2273, 2274, 2276, 2277, 2280, 2282
\ExplSyntaxOn	270
F	
file commands:	
\file_get:nnNTF	268
\fmtversion	3

G		L	
group commands:		\labelformat	3
\group_begin:	93, 259, 302, 924, 1029, 1042, 1754, 1780, 1999, 2002, 2022, 2025, 2178, 2183, 2186, 2201, 2208, 2223, 2238, 2241, 2272, 2275	\languagename	21, 727
\group_end:	96, 296, 305, 932, 1045, 1057, 1777, 1783, 2017, 2019, 2030, 2032, 2181, 2192, 2199, 2205, 2211, 2226, 2267, 2269, 2281, 2283	M	
H		\mainbabelname	21, 734
\hyperlink	51	\MessageBreak	10
I		msg commands:	
\IfBooleanTF	1063	\msg_info:nnn	344, 374
\IfFormatAtLeastTF	3, 4	\msg_line_context:	102, 108, 139, 153, 157, 159, 161, 163
int commands:		\msg_new:nnn	100, 106, 111, 113, 115, 117, 122, 128, 130, 132, 137, 142, 147, 149, 151, 156, 158, 160, 162
\int_case:nnTF	1589, 1615, 1646, 1810, 1907, 1943	\msg_note:nnn	280
\int_compare:nNnTF	1192, 1247, 1314, 1330, 1360, 1362, 1407, 1557, 1602, 1636, 1799, 1801, 1864, 1888, 1930, 2361, 2367, 2387, 2393	\msg_warning:nn	535, 560, 664, 670, 827, 848
\int_compare_p:nNn	1376, 1384, 2050, 2061, 2142	\msg_warning:nnn	231, 250, 286, 294, 503, 889, 931, 974, 1013, 1550, 1707, 2084, 2113
\int_eval:n	90	\msg_warning:nnnn	233, 449, 1255, 1278, 1301, 1339
\int_incr:N	1837, 1877, 1879, 1893, 1895, 1899, 1901, 1990	N	
\int_new:N	1075, 1076, 1421, 1422, 1423, 1424	\newcounter	4
\int_set:Nn	1361, 1363, 1367, 1370	\NewDocumentCommand	223, 243, 873, 875, 922, 1025, 1061
\int_use:N	33, 35, 50	\newtheorem	59
\int_zero:N	1351, 1352, 1458, 1459, 1460, 1461, 1836, 1838, 1839, 1985, 1986	\nobreakspace	426, 2416, 2417, 2419, 2420, 2422, 2424, 2606, 2607, 2609, 2610, 2612, 2614, 2784, 2785, 2787, 2788, 2790, 2792, 2957, 2958, 2960, 2961, 2963, 2965, 3141, 3142, 3144, 3145, 3147, 3149
iow commands:		P	
\iow_char:N	103, 109, 120, 125, 126, 135, 145	\PackageError	7
\iow_newline:	144, 148	\pagenumbering	6
K		prg commands:	
keys commands:		\prg_generate_conditional_	394, 409
\keys_define:nn	26, 324, 336, 353, 367, 443, 471, 497, 521, 549, 556, 568, 593, 602, 617, 626, 634, 667, 674, 690, 708, 743, 781, 814, 817, 824, 834, 845, 856, 867, 885, 897, 935, 962, 983, 1006	\prg_new_protected_conditional:Npnn	382, 397, 412
\keys_set:nn	26, 30, 277, 839, 874, 880, 929, 1030	\prg_return_false:	390, 392, 405, 407, 417
keyval commands:		\prg_return_true:	389, 404, 416
\keyval_parse:nnn	447, 501	\ProcessKeysOptions	872
		prop commands:	
		\prop_get:NnN	2310
		\prop_get:NnNTF	260, 384, 387, 399, 402, 414, 925, 2072, 2090, 2095, 2294, 2336, 2346
		\prop_gput:Nnn	237, 247, 947, 955
		\prop_gput_if_new:Nnn	312, 320

`\prop_gset_from_keyval:Nn` 420
`\prop_if_exist:NTF` 273, 877
`\prop_if_exist_p:N` 2301, 2342
`\prop_if_in:NnTF` 25, 227, 245
`\prop_if_in_p:Nn` 60, 2305
`\prop_item:Nn` ... 27, 61, 230, 234, 248
`\prop_new:N`
..... 220, 274, 419, 442, 496, 852, 878
`\prop_put:Nnn` 440, 863, 912
`\prop_remove:Nn` 439, 862, 904
`\providecommand` 3
`\ProvidesExplPackage` 14

R

`\refstepcounter` 3
`\RequirePackage` ... 16, 17, 18, 19, 20, 660

S

seq commands:

`\seq_clear:N` 613, 1097
`\seq_const_from_clist:Nn`
..... 166, 174, 187, 199
`\seq_gconcat:NNN` ... 207, 210, 214, 217
`\seq_get_left:NN` 1478
`\seq_gput_right:Nn` 278
`\seq_if_empty:NTF` 1473
`\seq_if_in:NnTF` 263, 477, 1085
`\seq_map_break:n` 81, 1396, 1399
`\seq_map_function:NN` 1100
`\seq_map_indexed_inline:Nn` . 18, 1356
`\seq_map_inline:Nn` 333, 350,
364, 853, 882, 894, 959, 980, 1003, 1393
`\seq_map_tokens:Nn` 63
`\seq_new:N` 206,
213, 253, 470, 601, 1059, 1094, 1416
`\seq_pop_left:NN` 1472
`\seq_put_right:Nn` 479, 1089
`\seq_reverse:N` 607
`\seq_set_eq:NN` 1452
`\seq_set_from_clist:Nn` 606, 1031
`\seq_sort:Nn` 1103

sort commands:

`\sort_return_same:`
..... 33, 38, 1110, 1115, 1152,
1197, 1199, 1252, 1258, 1275, 1281,
1304, 1335, 1342, 1380, 1396, 1412
`\sort_return_swapped:`
..... 33, 38, 1123, 1161, 1196,
1251, 1298, 1334, 1388, 1399, 1411

str commands:

`\str_case:nnTF` 749, 785
`\str_if_eq:nnTF` 80, 229
`\str_if_eq_p:nn` 2127, 2133, 2135, 2139
`\str_new:N` 673

`\str_set:Nn` 678, 680, 682, 684

T

TeX and L^AT_EX 2_ε commands:

`\@Alph` 59
`\@addtoreset` 4
`\@chapapp` 59
`\@currentcounter`
..... 4, 21, 25, 28, 30, 33, 84, 86
`\@currentlabel` 3
`\@ifl@t@r` 3
`\@ifpackageloaded`
..... 532, 547, 657, 725, 731, 831
`\@onlypreamble` 240, 252, 934
`\bbl@loaded` 21
`\bbl@main@language` 21, 728
`\c@` 3
`\c@page` 6, 94
`\cl@` 4
`\hyper@link` .. 1757, 2005, 2160, 2244
`\p@...` 3
`\zref@addprop` 22, 32, 34, 36, 87, 88, 99
`\zref@default`
..... 51, 2035, 2152, 2213, 2219, 2286
`\zref@extractdefault`
..... 51, 1082, 1141, 1143, 1189,
1190, 1193, 1195, 1207, 1211, 1215,
1219, 1243, 1244, 1248, 1250, 1270,
1293, 1408, 1410, 1488, 1493, 1762,
1767, 1773, 2008, 2009, 2011, 2014,
2028, 2165, 2169, 2174, 2189, 2249,
2253, 2258, 2263, 2278, 2358, 2359,
2362, 2364, 2368, 2370, 2380, 2381,
2384, 2385, 2388, 2390, 2394, 2396
`\zref@ifpropundefined` 17
`\zref@ifrefcontainsprop` . 17, 1759,
1994, 2007, 2156, 2162, 2229, 2246
`\zref@ifrefundefined`
1105, 1107, 1119, 1505, 1507, 1512,
1545, 1704, 1713, 1855, 2040, 2151
`\ZREF@mainlist` 22, 32, 34, 36, 87, 88, 99
`\zref@newprop` 4, 21, 23, 33, 35, 83, 85, 98
`\zref@refused` 1544
`\zref@wrapper@babel` 29, 1026
`\textendash` 430
`\the` 3
`\thechapter` 59
`\thepage` 6, 95
`\thesection` 59
tl commands:
`\c_empty_tl` 1082, 1141, 1143,
1207, 1211, 1215, 1219, 1489, 1494
`\c_novalue_tl` 858, 899

274, 279, 281, 287, 313, 321, 385,
 387, 400, 402, 926, 970, 991, 996, 1018
 \g_zrefclever_fallback_dict_
 prop 414, 419, 420
 _zrefclever_get_default_
 transl:nnN 9, 397, 409
 _zrefclever_get_default_
 transl:nnNTF 2321
 _zrefclever_get_enclosing_
 counters:n 5, 37, 42, 84
 _zrefclever_get_enclosing_
 counters_value:n ... 5, 37, 51, 86
 _zrefclever_get_fallback_
 transl:nN 412
 _zrefclever_get_fallback_
 transl:nNTF 2326
 _zrefclever_get_ref:n 1607, 1623,
 1635, 1639, 1659, 1672, 1675, 1687,
 1690, 1724, 1744, 1915, 1928, 1933,
 1953, 1964, 1967, 1977, 1980, 1992
 _zrefclever_get_ref_first: ...
 40, 51, 1737, 1793, 2149
 _zrefclever_get_ref_font:nN ...
 8, 24, 1559, 1560, 1561, 2333
 _zrefclever_get_ref_string:nN .
 8, 9, 14, 24, 1046, 1464,
 1465, 1466, 1562, 1563, 1564, 1565,
 1566, 1567, 1568, 1569, 1570, 2290
 _zrefclever_get_type_transl:nnnN
 9, 382, 394
 _zrefclever_get_type_transl:nnnNTF
 2077, 2100, 2106, 2315
 \l_zrefclever_label_a_tl
 ... 1067, 1472, 1489, 1505, 1544,
 1545, 1551, 1595, 1607, 1623, 1639,
 1675, 1690, 1717, 1724, 1855, 1859,
 1868, 1892, 1915, 1933, 1967, 1980
 \l_zrefclever_label_b_tl . 1067,
 1475, 1478, 1494, 1507, 1512, 1859
 \l_zrefclever_label_count_int ..
 39, 1421,
 1458, 1557, 1589, 1836, 1864, 1990
 \l_zrefclever_label_enclcnt_a_
 tl 1067, 1206,
 1208, 1209, 1230, 1295, 1319, 1320
 \l_zrefclever_label_enclcnt_b_
 tl 1067, 1210,
 1212, 1213, 1232, 1272, 1321, 1322
 \l_zrefclever_label_enclval_a_
 tl 1067, 1214,
 1216, 1217, 1315, 1323, 1324, 1331
 \l_zrefclever_label_enclval_b_
 tl 1067, 1218,
 1220, 1221, 1317, 1325, 1326, 1333
 \l_zrefclever_label_type_a_tl ..
 1067, 1081, 1083, 1087,
 1090, 1140, 1149, 1158, 1166, 1174,
 1366, 1395, 1482, 1486, 1519, 1527,
 1532, 1548, 1596, 1869, 2299, 2302,
 2306, 2311, 2317, 2340, 2343, 2347
 \l_zrefclever_label_type_b_tl ..
 1067,
 1142, 1150, 1159, 1167, 1175, 1369,
 1398, 1483, 1491, 1520, 1528, 1532
 _zrefclever_label_type_put_
 new_right:n 31, 1079, 1101
 \l_zrefclever_label_types_seq ..
 31, 1086, 1089, 1094, 1097, 1393
 _zrefclever_labels_in_sequence:nn
 1716, 1858, 2353
 \g_zrefclever_language_aliases_
 prop 220, 227, 230, 234,
 237, 245, 247, 248, 260, 384, 399, 925
 \l_zrefclever_last_of_type_bool
 39, 1414, 1503, 1508, 1509,
 1513, 1522, 1533, 1534, 1537, 1575
 \l_zrefclever_lastsep_tl . 1437,
 1566, 1622, 1638, 1658, 1674, 1686
 \l_zrefclever_link_star_bool ...
 1032, 1059, 1997, 2125, 2235
 \l_zrefclever_listsep_tl
 ... 1436, 1565, 1634, 1671, 1914,
 1927, 1932, 1952, 1963, 1966, 1976
 \l_zrefclever_load_dict_
 verbose_bool ... 254, 284, 293, 303
 \g_zrefclever_loaded_dictionaries_
 seq 253, 264, 278
 \l_zrefclever_main_language_tl .
 . 21, 721, 728, 734, 738, 741, 754, 790
 \l_zrefclever_name_format_
 fallback_tl
 .. 1446, 2065, 2068, 2070, 2097, 2109
 \l_zrefclever_name_format_tl ...
 ... 1446, 2052, 2053, 2056, 2057,
 2065, 2066, 2074, 2080, 2092, 2103
 \l_zrefclever_name_in_link_bool
 .. 1446, 1752, 2129, 2145, 2146, 2154
 \l_zrefclever_namefont_tl 1429,
 1559, 1755, 1781, 2179, 2209, 2224
 \l_zrefclever_nameinlink_str ...
 673, 678,
 680, 682, 684, 2127, 2133, 2135, 2139
 \l_zrefclever_namesep_tl
 .. 1433, 1562, 2182, 2212, 2220, 2227
 \l_zrefclever_next_is_same_bool
 39, 58, 1423,
 1848, 1878, 1894, 1900, 2373, 2399

\l_zrefclever_next_maybe_range_- bool .. 39, 58, 1423, 1712, 1721, 1847, 1874, 1885, 2365, 2372, 2391, 2398 166, 365, 1004
\l_zrefclever_noabbrev_first_- bool 707, 716, 2062	\c_zrefclever_ref_options_- possibly_type_specific_seq 14, 166, 351, 981
\l_zrefclever_noteseq_tl 1046, 1047, 1441	\l_zrefclever_ref_options_prop 24, 26, 852, 862, 863, 2294, 2336
_zrefclever_page_format_aux: 90, 94	\c_zrefclever_ref_options_- reference_seq 166, 854
\g_zrefclever_page_format_tl 6, 89, 95, 98	\c_zrefclever_ref_options_- typesetup_seq 166, 895
\l_zrefclever_pairsep_tl 1435, 1564, 1606, 1722	\l_zrefclever_ref_property_tl 17, 520, 525, 527, 533, 536, 552, 561, 1098, 1131, 1480, 1994, 2014, 2028, 2157, 2190, 2230, 2264, 2279, 2355
_zrefclever_prop_put_non_- empty:Nnn 14, 436, 453, 507	\l_zrefclever_ref_typeset_font_- tl 813, 815, 1043
_zrefclever_provide_dict_- default_transl:nn .. 318, 342, 359	\l_zrefclever_reffont_in_tl 1431, 1561, 2003, 2026, 2187, 2242, 2276
_zrefclever_provide_dict_type_- transl:nn 310, 360, 377	\l_zrefclever_reffont_out_tl 1430, 1560, 2000, 2023, 2184, 2203, 2239, 2273
_zrefclever_provide_dictionary:n 9, 30, 257, 298, 304, 780, 1033	\l_zrefclever_refpos_in_tl 1445, 1570, 2015, 2029, 2191, 2265, 2280
_zrefclever_provide_dictionary_- verbose:n 300, 307, 755, 763, 769, 791, 799, 805	\l_zrefclever_refpos_out_tl 1443, 1568, 2018, 2031, 2204, 2268, 2282
\l_zrefclever_range_beg_label_- tl 39, 1423, 1457, 1635, 1654, 1659, 1669, 1672, 1684, 1687, 1835, 1876, 1892, 1925, 1928, 1950, 1953, 1961, 1964, 1974, 1977	\l_zrefclever_refpre_in_tl 1444, 1569, 2013, 2027, 2188, 2262, 2277
\l_zrefclever_range_count_int 39, 1423, 1460, 1615, 1647, 1838, 1877, 1889, 1893, 1899, 1907, 1944, 1985	\l_zrefclever_refpre_out_tl 1442, 1567, 2001, 2024, 2185, 2240, 2274
\l_zrefclever_range_inhibit_- next_bool 39, 1423, 1853	\l_zrefclever_setup_type_tl 164, 275, 314, 329, 330, 341, 358, 372, 879, 907, 915, 928, 940, 941, 967, 988, 997, 1011, 1019
\l_zrefclever_range_same_count_- int 39, 1423, 1461, 1602, 1636, 1647, 1839, 1879, 1895, 1901, 1930, 1944, 1986	\l_zrefclever_sort_decided_bool ... 1077, 1223, 1227, 1246, 1257, 1274, 1280, 1297, 1303, 1329, 1341
\l_zrefclever_rangesep_tl 1434, 1563, 1689, 1723, 1979	_zrefclever_sort_default:nn 31–33, 1133, 1138
\l_zrefclever_ref_language_tl 21, 22, 720, 741, 753, 756, 761, 764, 768, 770, 780, 789, 792, 797, 800, 804, 806, 1033, 2078, 2101, 2107, 2316, 2322	_zrefclever_sort_default_- different_types:nn . 18, 1182, 1349
\c_zrefclever_ref_options_font_- seq 10, 166	_zrefclever_sort_default_same_- type:nn 1178, 1204
\c_zrefclever_ref_options_- necessarily_not_type_specific_- seq 14, 166, 334, 883, 960	_zrefclever_sort_labels: 31, 33, 38, 1041, 1095
\c_zrefclever_ref_options_- necessarily_type_specific_seq	_zrefclever_sort_page:nn 38, 1132, 1405
	\l_zrefclever_sort_prior_a_int 1075, 1351, 1360, 1361, 1367, 1377, 1385
	\l_zrefclever_sort_prior_b_int 1076, 1352, 1362, 1363, 1370, 1378, 1386

\l__zrefclever_tlastsep_tl	curr_tl
1440, 1466, 1824	1416, 1454, 1604,
\l__zrefclever_tlistsep_tl	1620, 1629, 1656, 1666, 1681, 1702,
1439, 1465, 1802	1719, 1736, 1743, 1750, 1793, 1814,
\l__zrefclever_tpairsep_tl	1819, 1825, 1831, 1832, 1912, 1923,
1438, 1464, 1818	1948, 1959, 1972, 2055, 2136, 2140
\l__zrefclever_type_<type>-	\l__zrefclever_typeset_queue-
options_prop 26	prev_tl 1416, 1453, 1803, 1831
\l__zrefclever_type_count_int . . .	\l__zrefclever_typeset_range-
39, 1421, 1459, 1799,	bool 625, 628, 1040, 1700
1801, 1810, 1837, 2050, 2061, 2142	\l__zrefclever_typeset_ref_bool .
\l__zrefclever_type_first_label-	566, 573, 578, 583, 1734, 1741
tl 1416, 1455, 1595, 1704,	__zrefclever_typeset_refs:
1713, 1717, 1744, 1760, 1763, 1768,	39, 40, 51, 52, 54, 1044, 1450
1774, 1833, 1868, 2040, 2151, 2157,	__zrefclever_typeset_refs_aux-
2163, 2165, 2169, 2174, 2189, 2230,	last_of_type: 1578, 1586
2247, 2249, 2253, 2258, 2263, 2278	__zrefclever_typeset_refs_aux-
\l__zrefclever_type_first_label-	not_last_of_type: 1582, 1842
type_tl 1416, 1456, 1596,	\l__zrefclever_typeset_sort_bool
1708, 1834, 1869, 2043, 2073, 2079,	592, 595, 1039
2085, 2091, 2096, 2102, 2108, 2114	\l__zrefclever_typesort_seq
__zrefclever_type_name_setup: . .	18, 601, 606, 607, 613, 1356
9, 40, 1732, 2038	\l__zrefclever_use_hyperref_bool
\l__zrefclever_type_name_tl . 52,	632, 639,
1446, 1776, 1782, 2041, 2044, 2075,	644, 649, 659, 665, 1997, 2124, 2234
2081, 2083, 2093, 2098, 2104, 2110,	\l__zrefclever_warn_hyperref-
2112, 2126, 2180, 2210, 2217, 2225	bool 633, 640, 645, 650, 663
\l__zrefclever_typeset_compress-	__zrefclever_zcref:nnn . . 1026, 1027
bool 616, 619, 1850	__zrefclever_zcref:nnnn 29, 32, 1027
\l__zrefclever_typeset_labels-	\l__zrefclever_zcref_labels_seq .
seq . . . 1416, 1452, 1472, 1473, 1478	32, 1031, 1055, 1059, 1100, 1103, 1452
\l__zrefclever_typeset_last_bool	\l__zrefclever_zcref_note_tl . . .
39, 1414,	816, 819, 1048
1469, 1470, 1476, 1502, 1807, 2141	\l__zrefclever_zcref_with_check-
\l__zrefclever_typeset_name_bool	bool 823, 838, 1036, 1051
567, 574, 579, 584, 1734, 1748	\l__zrefclever_zrefcheck-
\l__zrefclever_typeset_queue-	available_bool
	822, 833, 844, 1035, 1050