

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	17
5	Configuration	28
5.1	\zcsetup	28
5.2	\zcRefTypeSetup	29
5.3	\zcDeclareTranslations	30
6	User interface	32
6.1	\zceref	32
6.2	\zcpageref	34
7	Sorting	34
8	Typesetting	42
9	Special handling	66
9.1	\appendix	66
9.2	enumitem package	67

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

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

10	Dictionaries	67
10.1	English	67
10.2	German	70
10.3	French	74
10.4	Portuguese	77
10.5	Spanish	81
Index		84

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 ‘ltcounts.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'
104   ~switch-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~\msg_line_context:. }
113 \msg_new:nnn { zref-clever } { language-declared }
114 { Language~'#1'~is~already~declared.~Nothing~to~do. }
115 \msg_new:nnn { zref-clever } { unknown-language-alias }
116 {
117   Language~'#1'~is~unknown,~cannot~alias~to~it.~See~documentation~for~
118   '\iow_char:N\zcDeclareLanguage'~and~
119   '\iow_char:N\zcDeclareLanguageAlias'.
120 }
121 \msg_new:nnn { zref-clever } { unknown-language-transl }
122 {
123   Language~'#1'~is~unknown,~cannot~declare~translations~to~it.~
124   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
125   '\iow_char:N\zcDeclareLanguageAlias'.
126 }
127 \msg_new:nnn { zref-clever } { dict-loaded }
128 { Loaded~'#1'~dictionary. }
129 \msg_new:nnn { zref-clever } { dict-not-available }
130 { Dictionary~for~'#1'~not~available~\msg_line_context:. }
131 \msg_new:nnn { zref-clever } { unknown-language-load }
132 {
133   Language~'#1'~is~unknown~\msg_line_context:~Unable~to~load~dictionary.~
134   See~documentation~for~'\iow_char:N\zcDeclareLanguage'~and~
135   '\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.~
145   Use~the~starred~version~of~'\iow_char:N\zcref'~instead.
146 }
147 \msg_new:nnn { zref-clever } { missing-hyperref }
148 { Missing~'hyperref'~package.~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 } { missing-string }
163 {
164   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:~
165   But~we~should~have:~throw~a~rock~at~the~maintainer.
166 }
167 \msg_new:nnn { zref-clever } { single-element-range }
168 { 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). Besides those, there’s actually a fifth *internal* scope, with the least priority of all, a “fallback”, for the cases where it is meaningful to provide some value, even for an unknown language. These precedence rules are handled / enforced in `__zrefclever_get_ref_string:nN`, `__zrefclever_get_ref_font:nN`, and `__zrefclever_type_name_setup`: which are the basic functions to retrieve proper values for reference format settings.

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. The “fallback” settings are stored in `\g__zrefclever_fallback_dict_prop`.

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`.

```
169 \tl_new:N \l__zrefclever_setup_type_tl
170 \tl_new:N \l__zrefclever_dict_language_tl
```

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

`f_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
171 \seq_const_from_clist:Nn
172 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
173 {
174     tpairsep ,
175     tlistsep ,
176     tlastsep ,
177     notesep ,
178 }
179 \seq_const_from_clist:Nn
180 \c__zrefclever_ref_options_possibly_type_specific_seq
181 {
182     namesep ,
183     pairsep ,
184     listsep ,
185     lastsep ,
186     rangesep ,
187     refpre ,
188     refpos ,
189     refpre-in ,
190     refpos-in ,
191 }
```

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:.`

```
192 \seq_const_from_clist:Nn
193 \c__zrefclever_ref_options_necessarily_type_specific_seq
194 {
```

```

195     Name-sg ,
196     name-sg ,
197     Name-pl ,
198     name-pl ,
199     Name-sg-ab ,
200     name-sg-ab ,
201     Name-pl-ab ,
202     name-pl-ab ,
203 }

```

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

```

204 \seq_const_from_clist:Nn
205   \c__zrefclever_ref_options_font_seq
206   {
207     namefont ,
208     reffont ,
209     reffont-in ,
210   }
211 \seq_new:N \c__zrefclever_ref_options_typesetup_seq
212 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
213   \c__zrefclever_ref_options_possibly_type_specific_seq
214   \c__zrefclever_ref_options_necessarily_type_specific_seq
215 \seq_gconcat:NNN \c__zrefclever_ref_options_typesetup_seq
216   \c__zrefclever_ref_options_typesetup_seq
217   \c__zrefclever_ref_options_font_seq
218 \seq_new:N \c__zrefclever_ref_options_reference_seq
219 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
220   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
221   \c__zrefclever_ref_options_possibly_type_specific_seq
222 \seq_gconcat:NNN \c__zrefclever_ref_options_reference_seq
223   \c__zrefclever_ref_options_reference_seq
224   \c__zrefclever_ref_options_font_seq

```

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

4.3 Languages

`\g__zrefclever_languages_prop` Stores the names of known languages and the mapping from “language name” to “dictionary name”. Whether or not a language or alias is known to `zref-clever` is decided by its presence in this property list. A “base language” (loose concept here, meaning just “the name we gave for the dictionary in that particular language”) is just like any other one, the only difference is that the “language name” happens to be the same as the “dictionary name”, in other words, it is an “alias to itself”.

```

225 \prop_new:N \g__zrefclever_languages_prop

```

(End definition for `\g__zrefclever_languages_prop`.)

`\zcDeclareLanguage` Declare a new language for use with `zref-clever`. $\langle language \rangle$ is taken to be both the “language name” and the “dictionary name”. If $\langle language \rangle$ is already known, just warn. `\zcDeclareLanguage` is preamble only.

```

\zcDeclareLanguage {\langle language \rangle}

```

```

226 \NewDocumentCommand \zcDeclareLanguage { m }
227 {
228   \tl_if_empty:nF {#1}
229   {
230     \prop_if_in:NnTF \g__zrefclever_languages_prop {#1}
231     { \msg_warning:nnn { zref-clever } { language-declared } {#1} }
232     { \prop_gput:Nnn \g__zrefclever_languages_prop {#1} {#1} }
233   }
234 }
235 \@onlypreamble \zcDeclareLanguage

```

(End definition for \zcDeclareLanguage.)

`\zcDeclareLanguageAlias` Declare $\langle language\ alias \rangle$ to be an alias of $\langle aliased\ language \rangle$. $\langle aliased\ language \rangle$ must be already known to `zref-clever`, as stored in `\g__zrefclever_languages_prop`. `\zcDeclareLanguageAlias` is preamble only.

```

\zcDeclareLanguageAlias {\langle language alias \rangle} {\langle aliased language \rangle}

236 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
237 {
238   \tl_if_empty:nF {#1}
239   {
240     \prop_if_in:NnTF \g__zrefclever_languages_prop {#2}
241     {
242       \exp_args:Nnxx
243       \prop_gput:Nnn \g__zrefclever_languages_prop {#1}
244       { \prop_item:Nn \g__zrefclever_languages_prop {#2} }
245     }
246     { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
247   }
248 }
249 \@onlypreamble \zcDeclareLanguageAlias

```

(End definition for \zcDeclareLanguageAlias.)

4.4 Dictionaries

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

The built-in dictionaries and their related infrastructure are designed to perform “on the fly” loading of dictionaries, “lazily” as needed. Much like `babel` does for languages not declared in the preamble, but used in the document. This offers some convenience, of course, and that’s one reason to do it. But it also has the purpose of parsimony, of “loading the least possible”. My expectation is that for most use cases, users will require a single language of the functionality of `zref-clever` – the main language of the document –, even in multilingual documents. Hence, even the set of `babel` or `polyglossia` “loaded languages”, which would be the most tenable set if loading were restricted to the preamble, is bound to be an overshoot in typical cases. Therefore, we load at `\begin{document}` one single language (see [lang option](#)), as specified by the user in the preamble with the `lang` option or, failing any specification, the main language of the document, which is the default. Anything else is lazily loaded, on the fly, along the document.

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

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

Provide

<code>\g__zrefclever_loaded_dictionaries_seq</code>	Used to keep track of whether a dictionary has already been loaded or not.
250	<code>\seq_new:N \g__zrefclever_loaded_dictionaries_seq</code>
	(End definition for <code>\g__zrefclever_loaded_dictionaries_seq</code> .)
<code>\l__zrefclever_load_dict_verbose_bool</code>	Controls whether <code>__zrefclever_provide_dictionary:n</code> fails silently or verbosely in case of unknown languages or dictionaries not found.
251	<code>\bool_new:N \l__zrefclever_load_dict_verbose_bool</code>
	(End definition for <code>\l__zrefclever_load_dict_verbose_bool</code> .)
<code>__zrefclever_provide_dictionary:n</code>	Load dictionary for known <code>⟨language⟩</code> if it is available and if it has not already been loaded.
	<code>__zrefclever_provide_dictionary:n {⟨language⟩}</code>
252	<code>\cs_new_protected:Npn __zrefclever_provide_dictionary:n #1</code>
253	<code>{</code>
254	<code>\group_begin:</code>
255	<code>\prop_get:NnNTF \g__zrefclever_languages_prop {#1}</code>
256	<code>\l__zrefclever_dict_language_tl</code>
257	<code>{</code>
258	<code>\seq_if_in:NVF</code>

```

259     \g__zrefclever_loaded_dictionaries_seq
260     \l__zrefclever_dict_language_tl
261     {
262         \exp_args:Nx \file_get:nnNTF
263         { zref-clever- \l__zrefclever_dict_language_tl .dict }
264         { \ExplSyntaxOn }
265         \l_tmpa_tl
266         {
267             \prop_if_exist:cF
268             {
269                 g__zrefclever_dict_
270                 \l__zrefclever_dict_language_tl _prop
271             }
272             {
273                 \prop_new:c
274                 {
275                     g__zrefclever_dict_
276                     \l__zrefclever_dict_language_tl _prop
277                 }
278             }
279             \tl_clear:N \l__zrefclever_setup_type_tl
280             \exp_args:NnV
281             \keys_set:nn { zref-clever / dictionary } \l_tmpa_tl
282             \seq_gput_right:NV \g__zrefclever_loaded_dictionaries_seq
283             \l__zrefclever_dict_language_tl
284             \msg_note:nnx { zref-clever } { dict-loaded }
285             { \l__zrefclever_dict_language_tl }
286         }
287         {
288             \bool_if:NT \l__zrefclever_load_dict_verbose_bool
289             {
290                 \msg_warning:nnx { zref-clever } { dict-not-available }
291                 { \l__zrefclever_dict_language_tl }
292             }
293         }
294     }
295 }
296 {
297     \bool_if:NT \l__zrefclever_load_dict_verbose_bool
298     { \msg_warning:nnn { zref-clever } { unknown-language-load } {#1} }
299 }
300 \group_end:
301 }
302 \cs_generate_variant:Nn \__zrefclever_provide_dictionary:n { x }

```

(End definition for __zrefclever_provide_dictionary:n.)

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

```

    \__zrefclever_provide_dictionary_verbose:n {<language>}
303 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
304 {
305     \group_begin:

```

```

306     \bool_set_true:N \l__zrefclever_load_dict_verbose_bool
307     \__zrefclever_provide_dictionary:n {#1}
308     \group_end:
309 }
310 \cs_generate_variant:Nn \__zrefclever_provide_dictionary_verbose:n { x }

```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn
__zrefclever_provide_dict_default_transl:nn

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

```

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

311 \cs_new_protected:Npn \__zrefclever_provide_dict_type_transl:nn #1#2
312 {
313     \exp_args:Nnx \prop_gput_if_new:cnn
314     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
315     { type- \l__zrefclever_setup_type_tl - #1 } {#2}
316 }
317 \cs_new_protected:Npn \__zrefclever_provide_dict_default_transl:nn #1#2
318 {
319     \prop_gput_if_new:cnn
320     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
321     { default- #1 } {#2}
322 }

```

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

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

```

323 \keys_define:nn { zref-clever / dictionary }
324 {
325     type .code:n =
326     {
327         \tl_if_empty:nTF {#1}
328         { \tl_clear:N \l__zrefclever_setup_type_tl }
329         { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
330     } ,
331 }
332 \seq_map_inline:Nn
333 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
334 {
335     \keys_define:nn { zref-clever / dictionary }
336     {
337         #1 .value_required:n = true ,
338         #1 .code:n =

```

```

339         {
340             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
341             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
342             {
343                 \msg_info:nnn { zref-clever }
344                 { option-not-type-specific } {#1}
345             }
346         } ,
347     }
348 }
349 \seq_map_inline:Nn
350 \c__zrefclever_ref_options_possibly_type_specific_seq
351 {
352     \keys_define:nn { zref-clever / dictionary }
353     {
354         #1 .value_required:n = true ,
355         #1 .code:n =
356         {
357             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
358             { \__zrefclever_provide_dict_default_transl:nn {#1} {##1} }
359             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
360         } ,
361     }
362 }
363 \seq_map_inline:Nn
364 \c__zrefclever_ref_options_necessarily_type_specific_seq
365 {
366     \keys_define:nn { zref-clever / dictionary }
367     {
368         #1 .value_required:n = true ,
369         #1 .code:n =
370         {
371             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
372             {
373                 \msg_info:nnn { zref-clever }
374                 { option-only-type-specific } {#1}
375             }
376             { \__zrefclever_provide_dict_type_transl:nn {#1} {##1} }
377         } ,
378     }
379 }

```

Fallback

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

need to be provided here, since the later function sets an empty value if the option is not found.

TODO Add regression test to ensure all fallback “translations” are indeed present.

```

380 \prop_new:N \g__zrefclever_fallback_dict_prop
381 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
382 {
383   tpairsep = {,~} ,
384   tlistsep = {,~} ,
385   tlastsep = {,~} ,
386   notesep  = {~} ,
387   namesep  = {\nobreakspace} ,
388   pairsep  = {,~} ,
389   listsep  = {,~} ,
390   lastsep  = {,~} ,
391   rangesep = {\textendash} ,
392   refpre   = {} ,
393   refpos   = {} ,
394   refpre-in = {} ,
395   refpos-in = {} ,
396 }

```

Get translations

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

```

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

397 \prg_new_protected_conditional:Npnn
398   \__zrefclever_get_type_transl:nnnN #1#2#3#4 { F }
399 {
400   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
401   \l__zrefclever_dict_language_tl
402   {
403     \prop_get:cnNTF
404       { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
405       { type- #2 - #3 } #4
406       { \prg_return_true: }
407       { \prg_return_false: }
408   }
409   { \prg_return_false: }
410 }
411 \prg_generate_conditional_variant:Nnn
412   \__zrefclever_get_type_transl:nnnN { xxxN , xxnN } { F }

```

(End definition for `_zrefclever_get_type_transl:nnnNF`.)

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

```

\__zrefclever_get_default_transl:nnNF {<language>} {<key>}
  <tl variable> {<false code>}

```



```

413 \prg_new_protected_conditional:Npnn
414 \__zrefclever_get_default_transl:nnN #1#2#3 { F }
415 {
416   \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
417   \l__zrefclever_dict_language_tl
418   {
419     \prop_get:cnNTF
420     { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
421     { default- #2 } #3
422     { \prg_return_true: }
423     { \prg_return_false: }
424   }
425   { \prg_return_false: }
426 }
427 \prg_generate_conditional_variant:Nnn
428 \__zrefclever_get_default_transl:nnN { xnN } { F }

```

(End definition for __zrefclever_get_default_transl:nnNF.)

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

```

\__zrefclever_get_fallback_transl:nNF {<key>}
  <tl variable> {<false code>}

429 % {<key>}<tl var to set>
430 \prg_new_protected_conditional:Npnn
431 \__zrefclever_get_fallback_transl:nN #1#2 { F }
432 {
433   \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
434   { #1 } #2
435   { \prg_return_true: }
436   { \prg_return_false: }
437 }

```

(End definition for __zrefclever_get_fallback_transl:nNF.)

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 <property list> {<key>} {<value>}

438 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
439 {
440   \tl_if_empty:nTF {#3}
441   { \prop_remove:Nn #1 {#2} }
442   { \prop_put:Nnn #1 {#2} {#3} }
443 }

```

(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`.

```
444 \prop_new:N \l__zrefclever_counter_type_prop
445 \keys_define:nn { zref-clever / label }
446 {
447   countertype .code:n =
448   {
449     \keyval_parse:nnn
450     {
451       \msg_warning:nnnn { zref-clever }
452       { key-requires-value } { countertype }
453     }
454     {
455       \__zrefclever_prop_put_non_empty:Nnn
456       \l__zrefclever_counter_type_prop
457     }
458     {#1}
459   } ,
460   countertype .value_required:n = true ,
461   countertype .initial:n =
462   {
463     subsection      = section ,
464     subsubsection   = section ,
465     subparagraph    = paragraph ,
466     enumi            = item ,
467     enumii           = item ,
468     enumiii          = item ,
469     enumiv           = item ,
470   } ,
471 }
```

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.

```
472 \seq_new:N \l__zrefclever_counter_resetters_seq
473 \keys_define:nn { zref-clever / label }
474 {
475   counterresetters .code:n =
476   {
477     \clist_map_inline:nn {#1}
478     {
479       \seq_if_in:NnF \l__zrefclever_counter_resetters_seq {##1}
```

```

480         {
481             \seq_put_right:Nn
482             \l__zrefclever_counter_resettters_seq {##1}
483         }
484     }
485 },
486 counterresettters .initial:n =
487 {
488     part ,
489     chapter ,
490     section ,
491     subsection ,
492     subsubsection ,
493     paragraph ,
494     subparagraph ,
495 },
496 typesort .value_required:n = true ,
497 }

```

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`.

```

498 \prop_new:N \l__zrefclever_counter_resetby_prop
499 \keys_define:nn { zref-clever / label }
500 {
501     counterresetby .code:n =
502     {
503         \keyval_parse:nnn
504         {
505             \msg_warning:nnn { zref-clever }
506             { key-requires-value } { counterresetby }
507         }
508         {
509             \__zrefclever_prop_put_non_empty:Nnn
510             \l__zrefclever_counter_resetby_prop
511             {##1}
512         }
513     } ,
514     counterresetby .value_required:n = true ,
515     counterresetby .initial:n =
516     {

```

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.

```

517         enumii = enumi ,
518         enumiii = enumii ,
519         enumiv = enumiii ,
520     } ,
521 }

```

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.

```
522 \tl_new:N \l__zrefclever_ref_property_tl
523 \keys_define:nn { zref-clever / reference }
524 {
525   ref .choice: ,
526   ref / zc@thecnt .code:n =
527     { \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt } } ,
528   ref / page .code:n =
529     { \tl_set:Nn \l__zrefclever_ref_property_tl { page } } ,
530   ref / title .code:n =
531     {
532       \AddToHook { begindocument }
533       {
534         \@ifpackageloaded { zref-titleref }
535         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
536         {
537           \msg_warning:nn { zref-clever } { missing-zref-titleref }
538           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
539         }
540       }
541     } ,
542   ref .initial:n = zc@thecnt ,
543   ref .value_required:n = true ,
544   page .meta:n = { ref = page },
545   page .value_forbidden:n = true ,
546 }
547 \AddToHook { begindocument }
548 {
549   \@ifpackageloaded { zref-titleref }
550   {
551     \keys_define:nn { zref-clever / reference }
552     {
553       ref / title .code:n =
554       { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
555     }
556   }
557   {
558     \keys_define:nn { zref-clever / reference }
559     {
560       ref / title .code:n =
561       {
```

```

562             \msg_warning:nn { zref-clever } { missing-zref-titleref }
563             \tl_set:Nn \l__zrefclever_ref_property_tl { zc@thecnt }
564         }
565     }
566 }
567 }

```

typeset option

```

568 \bool_new:N \l__zrefclever_typeset_ref_bool
569 \bool_new:N \l__zrefclever_typeset_name_bool
570 \keys_define:nn { zref-clever / reference }
571 {
572     typeset .choice: ,
573     typeset / both .code:n =
574     {
575         \bool_set_true:N \l__zrefclever_typeset_ref_bool
576         \bool_set_true:N \l__zrefclever_typeset_name_bool
577     } ,
578     typeset / ref .code:n =
579     {
580         \bool_set_true:N \l__zrefclever_typeset_ref_bool
581         \bool_set_false:N \l__zrefclever_typeset_name_bool
582     } ,
583     typeset / name .code:n =
584     {
585         \bool_set_false:N \l__zrefclever_typeset_ref_bool
586         \bool_set_true:N \l__zrefclever_typeset_name_bool
587     } ,
588     typeset .initial:n = both ,
589     typeset .value_required:n = true ,
590
591     noname .meta:n = { typeset = ref },
592     noname .value_forbidden:n = true ,
593 }

```

sort option

```

594 \bool_new:N \l__zrefclever_typeset_sort_bool
595 \keys_define:nn { zref-clever / reference }
596 {
597     sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
598     sort .initial:n = true ,
599     sort .default:n = true ,
600     nosort .meta:n = { sort = false },
601     nosort .value_forbidden:n = true ,
602 }

```

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.

```

603 \seq_new:N \l__zrefclever_typesort_seq

```

```

604 \keys_define:nn { zref-clever / reference }
605 {
606   typesort .code:n =
607   {
608     \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
609     \seq_reverse:N \l__zrefclever_typesort_seq
610   } ,
611   typesort .initial:n =
612   { part , chapter , section , paragraph } ,
613   typesort .value_required:n = true ,
614   notypesort .code:n =
615   { \seq_clear:N \l__zrefclever_typesort_seq } ,
616   notypesort .value_forbidden:n = true ,
617 }

```

comp option

```

618 \bool_new:N \l__zrefclever_typeset_compress_bool
619 \keys_define:nn { zref-clever / reference }
620 {
621   comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
622   comp .initial:n = true ,
623   comp .default:n = true ,
624   nocomp .meta:n = { comp = false } ,
625   nocomp .value_forbidden:n = true ,
626 }

```

range option

```

627 \bool_new:N \l__zrefclever_typeset_range_bool
628 \keys_define:nn { zref-clever / reference }
629 {
630   range .bool_set:N = \l__zrefclever_typeset_range_bool ,
631   range .initial:n = false ,
632   range .default:n = true ,
633 }

```

hyperref option

```

634 \bool_new:N \l__zrefclever_use_hyperref_bool
635 \bool_new:N \l__zrefclever_warn_hyperref_bool
636 \keys_define:nn { zref-clever / reference }
637 {
638   hyperref .choice: ,
639   hyperref / auto .code:n =
640   {
641     \bool_set_true:N \l__zrefclever_use_hyperref_bool
642     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
643   } ,
644   hyperref / true .code:n =
645   {
646     \bool_set_true:N \l__zrefclever_use_hyperref_bool
647     \bool_set_true:N \l__zrefclever_warn_hyperref_bool
648   } ,
649   hyperref / false .code:n =
650   {

```

```

651     \bool_set_false:N \l__zrefclever_use_hyperref_bool
652     \bool_set_false:N \l__zrefclever_warn_hyperref_bool
653   } ,
654   hyperref .initial:n = auto ,
655   hyperref .default:n = auto
656 }
657 \AddToHook { begindocument }
658 {
659   \@ifpackageloaded { hyperref }
660   {
661     \bool_if:NT \l__zrefclever_use_hyperref_bool
662     { \RequirePackage { zref-hyperref } }
663   }
664   {
665     \bool_if:NT \l__zrefclever_warn_hyperref_bool
666     { \msg_warning:nn { zref-clever } { missing-hyperref } }
667     \bool_set_false:N \l__zrefclever_use_hyperref_bool
668   }
669   \keys_define:nn { zref-clever / reference }
670   {
671     hyperref .code:n =
672     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
673   }
674 }

```

nameinlink option

```

675 \str_new:N \l__zrefclever_nameinlink_str
676 \keys_define:nn { zref-clever / reference }
677 {
678   nameinlink .choice: ,
679   nameinlink / true .code:n =
680   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
681   nameinlink / false .code:n =
682   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
683   nameinlink / single .code:n =
684   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
685   nameinlink / tsingle .code:n =
686   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
687   nameinlink .initial:n = tsingle ,
688   nameinlink .default:n = true ,
689 }

```

cap and capfirst options

```

690 \bool_new:N \l__zrefclever_capitalize_bool
691 \bool_new:N \l__zrefclever_capitalize_first_bool
692 \keys_define:nn { zref-clever / reference }
693 {
694   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
695   cap .initial:n = false ,
696   cap .default:n = true ,
697   nocap .meta:n = { cap = false } ,
698   nocap .value_forbidden:n = true ,
699
700   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,

```

```

701     capfirst .initial:n = false ,
702     capfirst .default:n = true ,
703
704     C .meta:n =
705       { capfirst = true , noabbrevfirst = true },
706     C .value_forbidden:n = true ,
707   }

```

abbrev and noabbrevfirst options

```

708 \bool_new:N \l__zrefclever_abbrev_bool
709 \bool_new:N \l__zrefclever_noabbrev_first_bool
710 \keys_define:nn { zref-clever / reference }
711 {
712   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
713   abbrev .initial:n = false ,
714   abbrev .default:n = true ,
715   noabbrev .meta:n = { abbrev = false },
716   noabbrev .value_forbidden:n = true ,
717
718   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
719   noabbrevfirst .initial:n = false ,
720   noabbrevfirst .default:n = true ,
721 }

```

lang option

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

The overall setup here seems a little roundabout, but this is actually required. In the preamble, we (potentially) don’t yet have values for the “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 have been set when they are executed. Finally, we set a third `begindocument` hook, at `begindocument/before`, so that it runs after any options set in the preamble. This hook redefines the `lang` option for immediate execution in the document body, and ensures the 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`.

```

722 \tl_new:N \l__zrefclever_ref_language_tl

```



```

723 \tl_new:N \l__zrefclever_main_language_tl
724 \tl_new:N \l__zrefclever_current_language_tl
725 \AddToHook { begindocument }
726 {
727   \ifpackageloaded { babel }
728   {
729     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
730     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
731   }
732   {
733     \ifpackageloaded { polyglossia }
734     {
735       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
736       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
737     }
738     {
739       \tl_set:Nn \l__zrefclever_current_language_tl { english }
740       \tl_set:Nn \l__zrefclever_main_language_tl { english }
741     }
742   }

```

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

```

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

```

```

771         \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
772         \__zrefclever_provide_dictionary_verbosely:x
773         { \l__zrefclever_ref_language_tl }
774     }
775 }
776 },
777 lang .value_required:n = true ,
778 }
779 \AddToHook { begindocument / before }
780 {
781     \AddToHook { begindocument }
782     {

```

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.

```

783         \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

784     \keys_define:nn { zref-clever / reference }
785     {
786         lang .code:n =
787         {
788             \str_case:nnF {#1}
789             {
790                 { main }
791                 {
792                     \tl_set:Nn \l__zrefclever_ref_language_tl
793                     { \l__zrefclever_main_language_tl }
794                     \__zrefclever_provide_dictionary_verbosely:x
795                     { \l__zrefclever_ref_language_tl }
796                 }
797                 { current }
798                 {
799                     \tl_set:Nn \l__zrefclever_ref_language_tl
800                     { \l__zrefclever_current_language_tl }
801                     \__zrefclever_provide_dictionary_verbosely:x
802                     { \l__zrefclever_ref_language_tl }
803                 }
804             }
805         }
806         {
807             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
808             \__zrefclever_provide_dictionary_verbosely:x
809             { \l__zrefclever_ref_language_tl }
810         }
811     },
812     lang .value_required:n = true ,
813 }
814 }
815 }

```

font option

```

816 \tl_new:N \l__zrefclever_ref_typeset_font_tl

```

```

817 \keys_define:nn { zref-clever / reference }
818 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }

```

note option

```

819 \tl_new:N \l__zrefclever_zcref_note_tl
820 \keys_define:nn { zref-clever / reference }
821 {
822   note .tl_set:N = \l__zrefclever_zcref_note_tl ,
823   note .value_required:n = true ,
824 }

```

check option

Integration with zref-check.

```

825 \bool_new:N \l__zrefclever_zrefcheck_available_bool
826 \bool_new:N \l__zrefclever_zcref_with_check_bool
827 \keys_define:nn { zref-clever / reference }
828 {
829   check .code:n =
830     { \msg_warning:nn { zref-clever } { check-document-only } } ,
831 }
832 \AddToHook { begindocument }
833 {
834   \@ifpackageloaded { zref-check }
835   {
836     \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
837     \keys_define:nn { zref-clever / reference }
838     {
839       check .code:n =
840       {
841         \bool_set_true:N \l__zrefclever_zcref_with_check_bool
842         \keys_set:nn { zref-check / zcheck } {#1}
843       }
844     }
845   }
846   {
847     \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
848     \keys_define:nn { zref-clever / reference }
849     {
850       check .code:n =
851       { \msg_warning:nn { zref-clever } { missing-zref-check } }
852     }
853   }
854 }

```

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.

```

855 \prop_new:N \l__zrefclever_ref_options_prop
856 \seq_map_inline:Nn
857   \c__zrefclever_ref_options_reference_seq
858   {
859     \keys_define:nn { zref-clever / reference }
860     {
861       #1 .default:V = \c_novalue_tl ,
862       #1 .code:n =
863       {
864         \tl_if_novalue:nTF {##1}
865         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
866         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
867       } ,
868     }
869   }

```

Package options

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

```

870 \keys_define:nn { }
871 {
872   zref-clever / zcsetup .inherit:n = zref-clever / label ,
873   zref-clever / zcsetup .inherit:n = zref-clever / reference ,
874 }

```

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

```

875 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

```

```

876 \NewDocumentCommand \zcsetup { m }
877 { \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 $\langle options \rangle$ 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}
878 \NewDocumentCommand \zcRefTypeSetup { m m }
879 {
880   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
881   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
882   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
883   \keys_set:nn { zref-clever / typesetup } {#2}
884 }
```

(End definition for `\zcRefTypeSetup`.)

Inside `\zcRefTypeSetup` any of the options *can* receive empty values, and those values, if they exist in the property list, will override translations, regardless of their emptiness. In principle, we could live with the situation of, once a setting has been made in `\l__zrefclever_type_<type>_options_prop` or in `\l__zrefclever_ref_options_prop` it stays there forever, and can only be overridden by a new value at the same precedence level or a higher one. But it would be nice if an user can “unset” an option at either of those scopes to go back to the lower precedence level of the translations at any given point. So both in `\zcRefTypeSetup` and in setting reference options (see Section 4.5), we leverage the distinction of an “empty valued key” (`key=` or `key={}`) from a “key with no value” (`key`). This distinction is captured internally by the lower-level key parsing, but must be made explicit at `\keys_set:nn` by means of the `.default:V` property of the key in `\keys_define:nn`. For the technique 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>.

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

```

903     #1 .code:n =
904     {
905         \tl_if_novalue:nTF {##1}
906         {
907             \prop_remove:cn
908             {
909                 l__zrefclever_type_
910                 \l__zrefclever_setup_type_tl _options_prop
911             }
912             {#1}
913         }
914         {
915             \prop_put:cnn
916             {
917                 l__zrefclever_type_
918                 \l__zrefclever_setup_type_tl _options_prop
919             }
920             {#1} {##1}
921         }
922     } ,
923 }
924 }

```

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)}

925 \NewDocumentCommand \zcDeclareTranslations { m m }
926 {
927     \group_begin:
928     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
929     \l__zrefclever_dict_language_tl
930     {
931         \tl_clear:N \l__zrefclever_setup_type_tl
932         \keys_set:nn { zref-clever / translations } {#2}
933     }
934     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
935     \group_end:
936 }
937 \@onlypreamble \zcDeclareTranslations

```

(End definition for \zcDeclareTranslations.)

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

```

    \_zrefclever_declare_type_transl:nnnn {<language>} {<type>}
      {<key>} {<translation>}}
    \_zrefclever_declare_default_transl:nnn {<language>}
      {<key>} {<translation>}}

938 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
939 {
940   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
941     { type- #2 - #3 } {#4}
942 }
943 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
944 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
945 {
946   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
947     { default- #2 } {#3}
948 }
949 \cs_generate_variant:Nn \_zrefclever_declare_default_transl:nnn { Vnn }

(End definition for \_zrefclever_declare_type_transl:nnnn and \_zrefclever_declare_default_
transl:nnn.)

```

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

```

950 \keys_define:nn { zref-clever / translations }
951 {
952   type .code:n =
953   {
954     \tl_if_empty:NTF {#1}
955       { \tl_clear:N \l__zrefclever_setup_type_tl }
956       { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
957   } ,
958 }
959 \seq_map_inline:Nn
960 \c__zrefclever_ref_options_necessarily_not_type_specific_seq
961 {
962   \keys_define:nn { zref-clever / translations }
963   {
964     #1 .value_required:n = true ,
965     #1 .code:n =
966     {
967       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
968       {
969         \_zrefclever_declare_default_transl:Vnn
970         \l__zrefclever_dict_language_tl
971         {#1} {##1}
972       }
973       {
974         \msg_warning:nnn { zref-clever }
975           { option-not-type-specific } {#1}
976       }
977     } ,
978   }
979 }
980 \seq_map_inline:Nn
981 \c__zrefclever_ref_options_possibly_type_specific_seq

```

```

982 {
983   \keys_define:nn { zref-clever / translations }
984   {
985     #1 .value_required:n = true ,
986     #1 .code:n =
987     {
988       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
989       {
990         \__zrefclever_declare_default_transl:Vnn
991         \l__zrefclever_dict_language_tl
992         {#1} {##1}
993       }
994       {
995         \__zrefclever_declare_type_transl:Vnn
996         \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:Vnn
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` The main user command of the package.

`\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 $\{\langle labels \rangle\}$ as first argument, so that it can be protected by \zref@wrapper@babel in \zcref.

```
\__zrefclever_zcref:nnnn {\langle labels \rangle} {\langle * \rangle} {\langle options \rangle}
```

```
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_zcref_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_zcref_with_check_bool }
1037     { \zrefcheck_zcref_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_tmpa_tl
1047   \l_tmpa_tl
1048   \l__zrefclever_zcref_note_tl
```

Integration with zref-check.

```
1049   \bool_lazy_and:nnT
1050     { \l__zrefclever_zrefcheck_available_bool }
1051     { \l__zrefclever_zcref_with_check_bool }
1052     {
1053       \zrefcheck_zcref_end_label_maybe:
1054       \zrefcheck_zcref_run_checks_on_labels:n
1055       { \l__zrefclever_zcref_labels_seq }
```

```

1056     }
1057     \group_end:
1058 }
(End definition for \_zrefclever_zcref:nnnn.)

```

```

\_l_zrefclever_zcref_labels_seq
\_l_zrefclever_link_star_bool
1059 \seq_new:N \l__zrefclever_zcref_labels_seq
1060 \bool_new:N \l__zrefclever_link_star_bool
(End definition for \l__zrefclever_zcref_labels_seq and \l__zrefclever_link_star_bool.)

```

6.2 \zcpageref

\zcpageref A \pageref equivalent of \zcref.

```

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

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

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

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

```

\_l_zrefclever_label_type_a_tl
\_l_zrefclever_label_type_b_tl
\_l_zrefclever_label_enclcnt_a_tl
\_l_zrefclever_label_enclcnt_b_tl
\_l_zrefclever_label_enclval_a_tl
\_l_zrefclever_label_enclval_b_tl
1067 \tl_new:N \l__zrefclever_label_type_a_tl
1068 \tl_new:N \l__zrefclever_label_type_b_tl
1069 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
1070 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1071 \tl_new:N \l__zrefclever_label_enclval_a_tl
1072 \tl_new:N \l__zrefclever_label_enclval_b_tl
(End definition for \l__zrefclever_label_type_a_tl and others.)

```

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

```

1073 \bool_new:N \l__zrefclever_sort_decided_bool

```

(End definition for \l__zrefclever_sort_decided_bool.)

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

```
1074 \int_new:N \l__zrefclever_sort_prior_a_int
1075 \int_new:N \l__zrefclever_sort_prior_b_int
```

(End definition for \l__zrefclever_sort_prior_a_int and \l__zrefclever_sort_prior_b_int.)

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

```
1076 \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.

```
1077 \cs_new_protected:Npn \__zrefclever_sort_labels:
1078 {
```

Store label types sequence.

```
1079   \seq_clear:N \l__zrefclever_label_types_seq
1080   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1081   {
1082     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1083     \__zrefclever_label_type_put_new_right:n
1084   }
```

Sort.

```
1085   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1086   {
1087     \zref@ifrefundefined {##1}
1088     {
1089       \zref@ifrefundefined {##2}
1090       {
1091         % Neither label is defined.
1092         \sort_return_same:
1093       }
1094       {
1095         % The second label is defined, but the first isn't, leave the
1096         % undefined first (to be more visible).
1097         \sort_return_same:
1098       }
1099     }
1100     {
1101       \zref@ifrefundefined {##2}
1102       {
1103         % The first label is defined, but the second isn't, bring the
```

```

1104         % second forward.
1105         \sort_return_swapped:
1106     }
1107     {
1108         % The interesting case: both labels are defined.  References
1109         % to the "default" property or to the "page" are quite
1110         % different with regard to sorting, so we branch them here to
1111         % specialized functions.
1112         \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1113             { \__zrefclever_sort_page:nn {##1} {##2} }
1114             { \__zrefclever_sort_default:nn {##1} {##2} }
1115     }
1116 }
1117 }
1118 }

```

(End definition for __zrefclever_sort_labels:.)

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

```

\__zrefclever_label_type_put_new_right:n {\label}

1119 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1120 {
1121     \tl_set:Nx \l__zrefclever_label_type_a_tl
1122     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1123     \seq_if_in:NVF \l__zrefclever_label_types_seq
1124     \l__zrefclever_label_type_a_tl
1125     {
1126         \seq_put_right:NV \l__zrefclever_label_types_seq
1127         \l__zrefclever_label_type_a_tl
1128     }
1129 }

```

(End definition for __zrefclever_label_type_put_new_right:n.)

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

```

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

1130 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1131 {
1132     \tl_set:Nx \l__zrefclever_label_type_a_tl
1133     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }

```

```

1134 \tl_set:Nx \l__zrefclever_label_type_b_tl
1135 { \zref@extractdefault {#2} { zc@type } { \c_empty_tl } }
1136
1137 \bool_if:nTF
1138 {
1139   % The second label has a type, but the first doesn't, leave the
1140   % undefined first (to be more visible).
1141   \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1142   ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1143 }
1144 { \sort_return_same: }
1145 {
1146   \bool_if:nTF
1147   {
1148     % The first label has a type, but the second doesn't, bring the
1149     % second forward.
1150     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1151     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1152   }
1153   { \sort_return_swapped: }
1154   {
1155     \bool_if:nTF
1156     {
1157       % The interesting case: both labels have a type...
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     {
1162       \tl_if_eq:NNTF
1163       \l__zrefclever_label_type_a_tl
1164       \l__zrefclever_label_type_b_tl
1165       % ...and it's the same type.
1166       { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1167       % ...and they are different types.
1168       { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1169     }
1170     {
1171       % Neither label has a type. We can't do much of meaningful
1172       % here, but if it's the same counter, compare it.
1173       \exp_args:Nxx \tl_if_eq:nnTF
1174       { \zref@extractdefault {#1} { counter } { } }
1175       { \zref@extractdefault {#2} { counter } { } }
1176       {
1177         \int_compare:nNnTF
1178         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1179         >
1180         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1181         { \sort_return_swapped: }
1182         { \sort_return_same: }
1183       }
1184       { \sort_return_same: }
1185     }
1186   }
1187 }

```

```

1188 }

(End definition for \_zrefclever_sort_default:nn.)
Variant not provided by the kernel, for use in \_zrefclever_sort_default_-
same_type:nn.
1189 \cs_generate_variant:Nn \tl_reverse_items:n { V }

\_zrefclever_sort_default_same_type:nn      \_zrefclever_sort_default_same_type:nn {\label a}\{\label b}\}
1190 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1191 {
1192   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1193     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1194   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1195     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1196   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1197     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1198   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1199     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1200   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1201     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1202   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1203     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1204   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1205     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1206   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1207     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1208
1209   \bool_set_false:N \l__zrefclever_sort_decided_bool
1210   \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1211     {
1212       \bool_if:nTF
1213         {
1214           % Both are empty: neither label has any (further) "enclosing
1215           % counters" (left).
1216           \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1217           \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1218         }
1219         {
1220           \exp_args:Nxx \tl_if_eq:nnTF
1221             { \zref@extractdefault {#1} { counter } { } }
1222             { \zref@extractdefault {#2} { counter } { } }
1223             {
1224               \bool_set_true:N \l__zrefclever_sort_decided_bool
1225               \int_compare:nNnTF
1226                 { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1227                 >
1228                 { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1229                 { \sort_return_swapped: }
1230                 { \sort_return_same: }
1231             }
1232         }
1233       \msg_warning:nnnn { zref-clever }
1234         { counters-not-nested } {#1} {#2}
1235       \bool_set_true:N \l__zrefclever_sort_decided_bool

```

```

1236         \sort_return_same:
1237     }
1238 }
1239 {
1240     \bool_if:nTF
1241     {
1242         % 'a' is empty (and 'b' is not): 'b' may be nested in 'a'.
1243         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl
1244     }
1245     {
1246         \exp_args:NNx \tl_if_in:NnTF
1247         \l__zrefclever_label_enclcnt_b_tl
1248         { {\zref@extractdefault {#1} { counter } { }} }
1249         {
1250             \bool_set_true:N \l__zrefclever_sort_decided_bool
1251             \sort_return_same:
1252         }
1253         {
1254             \msg_warning:nnnn { zref-clever }
1255             { counters-not-nested } {#1} {#2}
1256             \bool_set_true:N \l__zrefclever_sort_decided_bool
1257             \sort_return_same:
1258         }
1259     }
1260 }
1261     \bool_if:nTF
1262     {
1263         % 'b' is empty (and 'a' is not): 'a' may be nested in 'b'.
1264         \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1265     }
1266     {
1267         \exp_args:NNx \tl_if_in:NnTF
1268         \l__zrefclever_label_enclcnt_a_tl
1269         { {\zref@extractdefault {#2} { counter } { }} }
1270         {
1271             \bool_set_true:N \l__zrefclever_sort_decided_bool
1272             \sort_return_swapped:
1273         }
1274         {
1275             \msg_warning:nnnn { zref-clever }
1276             { counters-not-nested } {#1} {#2}
1277             \bool_set_true:N \l__zrefclever_sort_decided_bool
1278             \sort_return_same:
1279         }
1280     }
1281 }
1282     {
1283         % Neither is empty: we can (possibly) compare the values
1284         % of the current enclosing counter in the loop, if they
1285         % are equal, we are still in the loop, if they are not, a
1286         % sorting decision can be made directly.
1287         \exp_args:Nxx \tl_if_eq:nnTF
1288         { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1289         { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1290     }

```

```

1290 \int_compare:nNnTF
1291 { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1292 =
1293 { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1294 {
1295   \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1296     { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1297   \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1298     { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1299   \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1300     { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1301   \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1302     { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1303 }
1304 {
1305   \bool_set_true:N \l__zrefclever_sort_decided_bool
1306   \int_compare:nNnTF
1307     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1308     >
1309     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1310     { \sort_return_swapped: }
1311     { \sort_return_same: }
1312 }
1313 }
1314 {
1315   \msg_warning:nnnn { zref-clever }
1316     { counters-not-nested } {#1} {#2}
1317   \bool_set_true:N \l__zrefclever_sort_decided_bool
1318   \sort_return_same:
1319 }
1320 }
1321 }
1322 }
1323 }
1324 }

```

(End definition for `_zrefclever_sort_default_same_type:nn`.)

```

\_zrefclever_sort_default_different_types:nn
\__zrefclever_sort_default_different_types:nn {<label a>} {<label b>}
1325 \cs_new_protected:Npn \_zrefclever_sort_default_different_types:nn #1#2
1326 {

```

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

```

1327 \int_zero:N \l__zrefclever_sort_prior_a_int
1328 \int_zero:N \l__zrefclever_sort_prior_b_int
1329 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1330 {
1331   \tl_if_eq:nnTF {##2} {{othertypes}}
1332   {
1333     \int_compare:nNnT { \l__zrefclever_sort_prior_a_int } = { 0 }
1334       { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1335     \int_compare:nNnT { \l__zrefclever_sort_prior_b_int } = { 0 }

```



```

1336         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1337     }
1338     {
1339         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1340         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1341         {
1342             \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1343             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1344         }
1345     }
1346 }

```

Then do the actual sorting.

```

1347 \bool_if:nTF
1348 {
1349     \int_compare_p:nNn
1350     { \l__zrefclever_sort_prior_a_int } <
1351     { \l__zrefclever_sort_prior_b_int }
1352 }
1353 { \sort_return_same: }
1354 {
1355     \bool_if:nTF
1356     {
1357         \int_compare_p:nNn
1358         { \l__zrefclever_sort_prior_a_int } >
1359         { \l__zrefclever_sort_prior_b_int }
1360     }
1361     { \sort_return_swapped: }
1362     {
1363         % Sort priorities are equal: the type that occurs first in
1364         % ‘labels’, as given by the user, is kept (or brought) forward.
1365         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1366         {
1367             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1368             { \seq_map_break:n { \sort_return_same: } }
1369             {
1370                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1371                 { \seq_map_break:n { \sort_return_swapped: } }
1372             }
1373         }
1374     }
1375 }
1376 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```
\__zrefclever_sort_page:nn {<label a>} {<label b>}
```

```

1377 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1378 {
1379     \int_compare:nNnTF
1380     { \zref@extractdefault {#1} { abspage } {-1} }
1381     >
1382     { \zref@extractdefault {#2} { abspage } {-1} }
1383     { \sort_return_swapped: }
1384     { \sort_return_same: }
1385 }

```

(End definition for `__zrefclever_sort_page:nn`.)

8 Typesetting

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

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

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

tl being its type). And, since we may need up to two type blocks in storage before typesetting, we have two of these “queues”: `\l__zrefclever_typeset_queue_curr_tl` and `\l__zrefclever_typeset_queue_prev_tl`.

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

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

One further thing to discuss here – to keep this “on record” – is inhibition of compression for individual labels. It is not difficult to handle it at the infrastructure side, what gets sloppy is the user facing syntax to signal such inhibition. For some possible alternatives for this (and good ones at that) see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be an option receiving the label(s) not to be compressed, this would be a repetition, but would keep the syntax clean. All in all, probably the best is simply not to allow individual inhibition of compression. We can already control compression of each `\zcref` call with existing options, this should be enough. I don’t think the small extra flexibility individual label control for this would grant is worth the syntax disruption it would entail. Anyway, it would be easy to deal with this in case the need arose, by just adding another condition (coming from whatever the chosen syntax was) when we check for `__zrefclever_labels_in_sequence:nn` in `__zrefclever_typeset_refs_not_last_of_type:.` But I remain unconvinced of the pertinence of doing so.

Variables

Auxiliary variables for `__zrefclever_typeset_refs`: main stack control.

```
\l__zrefclever_typeset_labels_seq
\l__zrefclever_typeset_last_bool
\l__zrefclever_last_of_type_bool
1386 \seq_new:N \l__zrefclever_typeset_labels_seq
1387 \bool_new:N \l__zrefclever_typeset_last_bool
1388 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

Auxiliary variables for `__zrefclever_typeset_refs`: main counters.

```
\l__zrefclever_type_count_int
\l__zrefclever_label_count_int
1389 \int_new:N \l__zrefclever_type_count_int
1390 \int_new:N \l__zrefclever_label_count_int
```

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

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

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_typeset_queue_curr_tl
\l__zrefclever_type_first_label_tl
\l__zrefclever_type_first_label_type_tl
1391 \tl_new:N \l__zrefclever_label_a_tl
```

```

1392 \tl_new:N \l__zrefclever_label_b_tl
1393 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1394 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1395 \tl_new:N \l__zrefclever_type_first_label_tl
1396 \tl_new:N \l__zrefclever_type_first_label_type_tl

```

(End definition for \l__zrefclever_label_a_tl and others.)

\l__zrefclever_type_name_tl Auxiliary variables for __zrefclever_typeset_refs: type name handling.

```

1397 \tl_new:N \l__zrefclever_type_name_tl
1398 \bool_new:N \l__zrefclever_name_in_link_bool
1399 \tl_new:N \l__zrefclever_name_format_tl
1400 \tl_new:N \l__zrefclever_name_format_fallback_tl

```

(End definition for \l__zrefclever_type_name_tl and others.)

\l__zrefclever_range_count_int Auxiliary variables for __zrefclever_typeset_refs: range handling.

```

\l__zrefclever_range_count_int
\l__zrefclever_range_same_count_int
\l__zrefclever_range_beg_label_tl
\l__zrefclever_next_maybe_range_bool
\l__zrefclever_next_is_same_bool
1401 \int_new:N \l__zrefclever_range_count_int
1402 \int_new:N \l__zrefclever_range_same_count_int
1403 \tl_new:N \l__zrefclever_range_beg_label_tl
1404 \bool_new:N \l__zrefclever_next_maybe_range_bool
1405 \bool_new:N \l__zrefclever_next_is_same_bool

```

(End definition for \l__zrefclever_range_count_int and others.)

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

```

1406 \tl_new:N \l__zrefclever_tpairsep_tl
1407 \tl_new:N \l__zrefclever_tlistsep_tl
1408 \tl_new:N \l__zrefclever_tlastsep_tl
1409 \tl_new:N \l__zrefclever_namesep_tl
1410 \tl_new:N \l__zrefclever_pairsep_tl
1411 \tl_new:N \l__zrefclever_listsep_tl
1412 \tl_new:N \l__zrefclever_lastsep_tl
1413 \tl_new:N \l__zrefclever_rangesep_tl
1414 \tl_new:N \l__zrefclever_refpre_out_tl
1415 \tl_new:N \l__zrefclever_refpos_out_tl
1416 \tl_new:N \l__zrefclever_refpre_in_tl
1417 \tl_new:N \l__zrefclever_refpos_in_tl
1418 \tl_new:N \l__zrefclever_namefont_tl
1419 \tl_new:N \l__zrefclever_reffont_out_tl
1420 \tl_new:N \l__zrefclever_reffont_in_tl

```

(End definition for \l__zrefclever_tpairsep_tl and others.)

Main functions

__zrefclever_typeset_refs: Main typesetting function for \zceref.

```

1421 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1422 {
1423   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1424   \l__zrefclever_zceref_labels_seq
1425   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1426   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl

```

```

1427 \tl_clear:N \l__zrefclever_type_first_label_tl
1428 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1429 \tl_clear:N \l__zrefclever_range_beg_label_tl
1430 \int_zero:N \l__zrefclever_label_count_int
1431 \int_zero:N \l__zrefclever_type_count_int
1432 \int_zero:N \l__zrefclever_range_count_int
1433 \int_zero:N \l__zrefclever_range_same_count_int
1434
1435 % Get type block options (not type-specific).
1436 \__zrefclever_get_ref_string:nN { tpairsep }
1437   \l__zrefclever_tpairsep_tl
1438 \__zrefclever_get_ref_string:nN { tlistsep }
1439   \l__zrefclever_tlistsep_tl
1440 \__zrefclever_get_ref_string:nN { tlastsep }
1441   \l__zrefclever_tlastsep_tl
1442
1443 % Process label stack.
1444 \bool_set_false:N \l__zrefclever_typeset_last_bool
1445 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1446 {
1447   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1448   \l__zrefclever_label_a_tl
1449   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1450   {
1451     \tl_clear:N \l__zrefclever_label_b_tl
1452     \bool_set_true:N \l__zrefclever_typeset_last_bool
1453   }
1454   {
1455     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1456     \l__zrefclever_label_b_tl
1457   }
1458
1459   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1460   {
1461     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1462     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1463   }
1464   {
1465     \tl_set:Nx \l__zrefclever_label_type_a_tl
1466     {
1467       \zref@extractdefault
1468       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1469     }
1470     \tl_set:Nx \l__zrefclever_label_type_b_tl
1471     {
1472       \zref@extractdefault
1473       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1474     }
1475   }
1476
1477   % First, we establish whether the "current label" (i.e. 'a') is the
1478   % last one of its type. This can happen because the "next label"
1479   % (i.e. 'b') is of a different type (or different definition status),
1480   % or because we are at the end of the list.

```

```

1481 \bool_if:NTF \l__zrefclever_typeset_last_bool
1482 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1483 {
1484   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1485   {
1486     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1487     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1488     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1489   }
1490   {
1491     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1492     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1493     {
1494       % Neither is undefined, we must check the types.
1495       \bool_if:nTF
1496       {
1497         % Both empty: same "type".
1498         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1499         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1500       }
1501       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1502       {
1503         \bool_if:nTF
1504         {
1505           % Neither empty: compare types.
1506           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1507           &&
1508           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1509         }
1510         {
1511           \tl_if_eq:NNTF
1512           \l__zrefclever_label_type_a_tl
1513           \l__zrefclever_label_type_b_tl
1514           {
1515             \bool_set_false:N
1516             \l__zrefclever_last_of_type_bool
1517           }
1518           {
1519             \bool_set_true:N
1520             \l__zrefclever_last_of_type_bool
1521           }
1522         }
1523         % One empty, the other not: different "types".
1524         {
1525           \bool_set_true:N
1526           \l__zrefclever_last_of_type_bool
1527         }
1528       }
1529     }
1530   }
1531 }
1532
1533 % Handle warnings in case of reference or type undefined.
1534 \zref@refused { \l__zrefclever_label_a_tl }

```

```

1535 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1536 {}
1537 {
1538   \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1539   {
1540     \msg_warning:nxx { zref-clever } { missing-type }
1541     { \l__zrefclever_label_a_tl }
1542   }
1543 }
1544
1545 % Get type-specific separators, refpre/pos and font options, once per
1546 % type.
1547 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1548 {
1549   \__zrefclever_get_ref_string:nN { namesep      }
1550   \l__zrefclever_namesep_tl
1551   \__zrefclever_get_ref_string:nN { rangesep     }
1552   \l__zrefclever_rangesep_tl
1553   \__zrefclever_get_ref_string:nN { pairsep      }
1554   \l__zrefclever_pairsep_tl
1555   \__zrefclever_get_ref_string:nN { listsep      }
1556   \l__zrefclever_listsep_tl
1557   \__zrefclever_get_ref_string:nN { lastsep      }
1558   \l__zrefclever_lastsep_tl
1559   \__zrefclever_get_ref_string:nN { refpre       }
1560   \l__zrefclever_refpre_out_tl
1561   \__zrefclever_get_ref_string:nN { refpos       }
1562   \l__zrefclever_refpos_out_tl
1563   \__zrefclever_get_ref_string:nN { refpre-in    }
1564   \l__zrefclever_refpre_in_tl
1565   \__zrefclever_get_ref_string:nN { refpos-in    }
1566   \l__zrefclever_refpos_in_tl
1567   \__zrefclever_get_ref_font:nN   { namefont     }
1568   \l__zrefclever_namefont_tl
1569   \__zrefclever_get_ref_font:nN   { reffont      }
1570   \l__zrefclever_reffont_out_tl
1571   \__zrefclever_get_ref_font:nN   { reffont-in   }
1572   \l__zrefclever_reffont_in_tl
1573 }
1574
1575 % Here we send this to a couple of auxiliary functions.
1576 \bool_if:NTF \l__zrefclever_last_of_type_bool
1577   % There exists no next label of the same type as the current.
1578   { \__zrefclever_typeset_refs_last_of_type: }
1579   % There exists a next label of the same type as the current.
1580   { \__zrefclever_typeset_refs_not_last_of_type: }
1581 }
1582 }

```

(End definition for `__zrefclever_typeset_refs:.`)

This is actually the one meaningful “big branching” we can do while processing the label stack: i) the “current” label is the last of its type block; or ii) the “current” label is *not* the last of its type block. Indeed, as mentioned above, quite a number of things can only be decided when the type block ends, and we only know this when we look at the

“next” label and find something of a different “type” (loose here, maybe different definition status, maybe end of stack). So, though this is not very strict, `__zrefclever_typeset_refs_last_of_type:` is more of a “wrapping up” function, and it is indeed the one which does the actual typesetting, while `__zrefclever_typeset_refs_not_last_of_type:` is more of an “accumulation” function.

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

```

1583 \cs_new_protected:Npn \__zrefclever_typeset_refs_last_of_type:
1584 {
1585   % Process the current label to the current queue.
1586   \int_case:nnF { \l__zrefclever_label_count_int }
1587   {
1588     % It is the last label of its type, but also the first one, and that's
1589     % what matters here: just store it.
1590     { 0 }
1591     {
1592       \tl_set:NV \l__zrefclever_type_first_label_tl
1593       \l__zrefclever_label_a_tl
1594       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1595       \l__zrefclever_label_type_a_tl
1596     }
1597
1598     % The last is the second: we have a pair (if not repeated).
1599     { 1 }
1600     {
1601       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1602       {
1603         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1604         {
1605           \exp_not:V \l__zrefclever_pairsep_tl
1606           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1607         }
1608       }
1609     }
1610   }
1611   % Last is third or more of its type: without repetition, we'd have the
1612   % last element on a list, but control for possible repetition.
1613   {
1614     \int_case:nnF { \l__zrefclever_range_count_int }
1615     {
1616       % There was no range going on.
1617       { 0 }
1618       {
1619         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1620         {
1621           \exp_not:V \l__zrefclever_lastsep_tl
1622           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1623         }
1624       }
1625       % Last in the range is also the second in it.
1626       { 1 }
1627       {
1628         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1629         {

```



```

1630         % We know 'range_beg_label' is not empty, since this is the
1631         % second element in the range, but the third or more in the
1632         % type list.
1633         \exp_not:V \l__zrefclever_listsep_tl
1634         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1635         \int_compare:nNnF
1636             { \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     {
1648         \l__zrefclever_range_count_int -
1649         \l__zrefclever_range_same_count_int
1650     }
1651     {
1652         % Repetition, not a range.
1653         { 0 }
1654         {
1655             % If 'range_beg_label' is empty, it means it was also the
1656             % first of the type, and hence was already handled.
1657             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1658             {
1659                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1660                 {
1661                     \exp_not:V \l__zrefclever_lastsep_tl
1662                     \__zrefclever_get_ref:V
1663                     \l__zrefclever_range_beg_label_tl
1664                 }
1665             }
1666         }
1667         % A 'range', but with no skipped value, treat as list.
1668         { 1 }
1669         {
1670             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1671             {
1672                 % Ditto.
1673                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1674                 {
1675                     \exp_not:V \l__zrefclever_listsep_tl
1676                     \__zrefclever_get_ref:V
1677                     \l__zrefclever_range_beg_label_tl
1678                 }
1679                 \exp_not:V \l__zrefclever_lastsep_tl
1680                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1681             }
1682         }
1683     }

```

```

1684         {
1685             % An actual range.
1686             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1687             {
1688                 % Ditto.
1689                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1690                 {
1691                     \exp_not:V \l__zrefclever_lastsep_tl
1692                     \__zrefclever_get_ref:V
1693                     \l__zrefclever_range_beg_label_tl
1694                 }
1695                 \exp_not:V \l__zrefclever_rangesep_tl
1696                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1697             }
1698         }
1699     }
1700 }
1701
1702 % Handle "range" option. The idea is simple: if the queue is not empty,
1703 % we replace it with the end of the range (or pair). We can still
1704 % retrieve the end of the range from 'label_a' since we know to be
1705 % processing the last label of its type at this point.
1706 \bool_if:NT \l__zrefclever_typeset_range_bool
1707 {
1708     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1709     {
1710         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1711         { }
1712         {
1713             \msg_warning:nxx { zref-clever } { single-element-range }
1714             { \l__zrefclever_type_first_label_type_tl }
1715         }
1716     }
1717     {
1718         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1719         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1720         { }
1721         {
1722             \__zrefclever_labels_in_sequence:nn
1723             { \l__zrefclever_type_first_label_tl }
1724             { \l__zrefclever_label_a_tl }
1725         }
1726         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1727         {
1728             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1729             { \exp_not:V \l__zrefclever_pairsep_tl }
1730             { \exp_not:V \l__zrefclever_rangesep_tl }
1731             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1732         }
1733     }
1734 }
1735
1736 % Now that the type block is finished, we can add the name and the first
1737 % ref to the queue. Also, if "typeset" option is not "both", handle it

```

```

1738 % here as well.
1739 \__zrefclever_type_name_setup:
1740 \bool_if:nTF
1741 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1742 {
1743   \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1744     { \__zrefclever_get_ref_first: }
1745 }
1746 {
1747   \bool_if:nTF
1748   { \l__zrefclever_typeset_ref_bool }
1749   {
1750     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1751       { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1752   }
1753   {
1754     \bool_if:nTF
1755     { \l__zrefclever_typeset_name_bool }
1756     {
1757       \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1758         {
1759           \bool_if:NTF \l__zrefclever_name_in_link_bool
1760           {
1761             \exp_not:N \group_begin:
1762             \exp_not:V \l__zrefclever_namefont_tl
1763             % It's two '@s', but escaped for DocStrip.
1764             \exp_not:N \hyper@@link
1765             {
1766               \zref@ifrefcontainsprop
1767               { \l__zrefclever_type_first_label_tl }
1768               { urluse }
1769               {
1770                 \zref@extractdefault
1771                 { \l__zrefclever_type_first_label_tl }
1772                 { urluse } {}
1773               }
1774               {
1775                 \zref@extractdefault
1776                 { \l__zrefclever_type_first_label_tl }
1777                 { url } {}
1778               }
1779             }
1780             {
1781               \zref@extractdefault
1782               { \l__zrefclever_type_first_label_tl }
1783               { anchor } {}
1784             }
1785             { \exp_not:V \l__zrefclever_type_name_tl }
1786           \exp_not:N \group_end:
1787         }
1788         {
1789           \exp_not:N \group_begin:
1790           \exp_not:V \l__zrefclever_namefont_tl
1791           \exp_not:V \l__zrefclever_type_name_tl

```

```

1792         \exp_not:N \group_end:
1793     }
1794 }
1795 }
1796 {
1797     % Logically, this case would correspond to "typeset=none", but
1798     % it should not occur, given that the options are set up to
1799     % typeset either "ref" or "name". Still, leave here a
1800     % sensible fallback, equal to the behavior of "both".
1801     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1802         { \__zrefclever_get_ref_first: }
1803 }
1804 }
1805 }
1806
1807 % Typeset the previous type, if there is one.
1808 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1809 {
1810     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1811     { \l__zrefclever_tlistsep_tl }
1812     \l__zrefclever_typeset_queue_prev_tl
1813 }
1814
1815 % Wrap up loop, or prepare for next iteration.
1816 \bool_if:NTF \l__zrefclever_typeset_last_bool
1817 {
1818     % We are finishing, typeset the current queue.
1819     \int_case:nnF { \l__zrefclever_type_count_int }
1820     {
1821         % Single type.
1822         { 0 }
1823         { \l__zrefclever_typeset_queue_curr_tl }
1824         % Pair of types.
1825         { 1 }
1826         {
1827             \l__zrefclever_tpairsep_tl
1828             \l__zrefclever_typeset_queue_curr_tl
1829         }
1830     }
1831     {
1832         % Last in list of types.
1833         \l__zrefclever_tlastsep_tl
1834         \l__zrefclever_typeset_queue_curr_tl
1835     }
1836 }
1837 {
1838     % There are further labels, set variables for next iteration.
1839     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1840         \l__zrefclever_typeset_queue_curr_tl
1841     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1842     \tl_clear:N \l__zrefclever_type_first_label_tl
1843     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1844     \tl_clear:N \l__zrefclever_range_beg_label_tl
1845     \int_zero:N \l__zrefclever_label_count_int

```

```

1846         \int_incr:N \l__zrefclever_type_count_int
1847         \int_zero:N \l__zrefclever_range_count_int
1848         \int_zero:N \l__zrefclever_range_same_count_int
1849     }
1850 }

```

(End definition for __zrefclever_typeset_refs_last_of_type:.)

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

```

1851 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1852 {
1853     % Signal if next label may form a range with the current one (only
1854     % considered if compression is enabled in the first place).
1855     \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1856     \bool_set_false:N \l__zrefclever_next_is_same_bool
1857     \bool_if:NT \l__zrefclever_typeset_compress_bool
1858     {
1859         \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1860         { }
1861         {
1862             \__zrefclever_labels_in_sequence:nn
1863             { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1864         }
1865     }
1866
1867     % Process the current label to the current queue.
1868     \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1869     {
1870         % Current label is the first of its type (also not the last, but it
1871         % doesn't matter here): just store the label.
1872         \tl_set:NV \l__zrefclever_type_first_label_tl
1873         \l__zrefclever_label_a_tl
1874         \tl_set:NV \l__zrefclever_type_first_label_type_tl
1875         \l__zrefclever_label_type_a_tl
1876
1877         % If the next label may be part of a range, we set 'range_beg_label'
1878         % to "empty" (we deal with it as the "first", and must do it there, to
1879         % handle hyperlinking), but also step the range counters.
1880         \bool_if:NT \l__zrefclever_next_maybe_range_bool
1881         {
1882             \tl_clear:N \l__zrefclever_range_beg_label_tl
1883             \int_incr:N \l__zrefclever_range_count_int
1884             \bool_if:NT \l__zrefclever_next_is_same_bool
1885             { \int_incr:N \l__zrefclever_range_same_count_int }
1886         }
1887     }
1888     {
1889         % Current label is neither the first (nor the last) of its type.
1890         \bool_if:NNTF \l__zrefclever_next_maybe_range_bool
1891         {
1892             % Starting, or continuing a range.
1893             \int_compare:nNnTF
1894             { \l__zrefclever_range_count_int } = { 0 }
1895             {

```

```

1896 % There was no range going, we are starting one.
1897 \tl_set:NV \l__zrefclever_range_beg_label_tl
1898 \l__zrefclever_label_a_tl
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 % Second or more in the range, but not the last.
1905 \int_incr:N \l__zrefclever_range_count_int
1906 \bool_if:NT \l__zrefclever_next_is_same_bool
1907 { \int_incr:N \l__zrefclever_range_same_count_int }
1908 }
1909 }
1910 {
1911 % Next element is not in sequence: there was no range, or we are
1912 % closing one.
1913 \int_case:nnF { \l__zrefclever_range_count_int }
1914 {
1915 % There was no range going on.
1916 { 0 }
1917 {
1918 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1919 {
1920 \exp_not:V \l__zrefclever_listsep_tl
1921 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1922 }
1923 }
1924 % Last is second in the range: if 'range_same_count' is also
1925 % '1', it's a repetition (drop it), otherwise, it's a "pair
1926 % within a list", treat as list.
1927 { 1 }
1928 {
1929 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1930 {
1931 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1932 {
1933 \exp_not:V \l__zrefclever_listsep_tl
1934 \__zrefclever_get_ref:V
1935 \l__zrefclever_range_beg_label_tl
1936 }
1937 \int_compare:nNnF
1938 { \l__zrefclever_range_same_count_int } = { 1 }
1939 {
1940 \exp_not:V \l__zrefclever_listsep_tl
1941 \__zrefclever_get_ref:V
1942 \l__zrefclever_label_a_tl
1943 }
1944 }
1945 }
1946 }
1947 {
1948 % Last is third or more in the range: if 'range_count' and
1949 % 'range_same_count' are the same, its a repetition (drop it),

```

```

1950 % if they differ by '1', its a list, if they differ by more,
1951 % it is a real range.
1952 \int_case:nnF
1953 {
1954   \l__zrefclever_range_count_int -
1955   \l__zrefclever_range_same_count_int
1956 }
1957 {
1958   { 0 }
1959   {
1960     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1961     {
1962       \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1963       {
1964         \exp_not:V \l__zrefclever_listsep_tl
1965         \__zrefclever_get_ref:V
1966         \l__zrefclever_range_beg_label_tl
1967       }
1968     }
1969   }
1970   { 1 }
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
1978         \l__zrefclever_range_beg_label_tl
1979       }
1980       \exp_not:V \l__zrefclever_listsep_tl
1981       \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1982     }
1983   }
1984 }
1985 {
1986   \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1987   {
1988     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1989     {
1990       \exp_not:V \l__zrefclever_listsep_tl
1991       \__zrefclever_get_ref:V
1992       \l__zrefclever_range_beg_label_tl
1993     }
1994     \exp_not:V \l__zrefclever_rangesep_tl
1995     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1996   }
1997 }
1998 }
1999 % Reset counters.
2000 \int_zero:N \l__zrefclever_range_count_int
2001 \int_zero:N \l__zrefclever_range_same_count_int
2002 }
2003 }

```

```

2004      % Step label counter for next iteration.
2005      \int_incr:N \l__zrefclever_label_count_int
2006    }

```

(End definition for `__zrefclever_typeset_refs_not_last_of_type:`)

Aux functions

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

`__zrefclever_ref_default:` Default values for undefined references and undefined type names, respectively. We are ultimately using `\zref@default`, but calls to it should be made through these internal functions, according to the case.

```

2007 \cs_new_protected:Npn \__zrefclever_ref_default:
2008 { \zref@default }
2009 \cs_new_protected:Npn \__zrefclever_name_default:
2010 { \zref@default }

```

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

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

```

\__zrefclever_get_ref:n {\label}

2011 \cs_new:Npn \__zrefclever_get_ref:n #1
2012 {
2013   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2014   {
2015     \bool_if:nTF
2016     {
2017       \l__zrefclever_use_hyperref_bool &&
2018       ! \l__zrefclever_link_star_bool

```



```

2019     }
2020     {
2021         \exp_not:N \group_begin:
2022         \exp_not:V \l__zrefclever_reffont_out_tl
2023         \exp_not:V \l__zrefclever_refpre_out_tl
2024         \exp_not:N \group_begin:
2025         \exp_not:V \l__zrefclever_reffont_in_tl
2026         % It's two '@s', but escaped for DocStrip.
2027         \exp_not:N \hyper@@link
2028         {
2029             \zref@ifrefcontainsprop {#1} { urluse }
2030             { \zref@extractdefault {#1} { urluse } { } }
2031             { \zref@extractdefault {#1} { url } { } }
2032         }
2033         { \zref@extractdefault {#1} { anchor } { } }
2034         {
2035             \exp_not:V \l__zrefclever_refpre_in_tl
2036             \zref@extractdefault {#1}
2037             { \l__zrefclever_ref_property_tl } { }
2038             \exp_not:V \l__zrefclever_refpos_in_tl
2039         }
2040         \exp_not:N \group_end:
2041         \exp_not:V \l__zrefclever_refpos_out_tl
2042         \exp_not:N \group_end:
2043     }
2044     {
2045         \exp_not:N \group_begin:
2046         \exp_not:V \l__zrefclever_reffont_out_tl
2047         \exp_not:V \l__zrefclever_refpre_out_tl
2048         \exp_not:N \group_begin:
2049         \exp_not:V \l__zrefclever_reffont_in_tl
2050         \exp_not:V \l__zrefclever_refpre_in_tl
2051         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2052         \exp_not:V \l__zrefclever_refpos_in_tl
2053         \exp_not:N \group_end:
2054         \exp_not:V \l__zrefclever_refpos_out_tl
2055         \exp_not:N \group_end:
2056     }
2057 }
2058 { \exp_not:N \__zrefclever_ref_default: }
2059 }
2060 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

__zrefclever_get_ref_first: Handles a complete reference block for the first label of its type to be accumulated in the “queue”, including “pre” and “pos” elements, hyperlinking, and the reference type “name”. It does not receive arguments, but relies on being called in the appropriate place in `__zrefclever_typeset_refs_last_of_type:` where a number of variables are expected to be appropriately set for it to consume. Prominently among those is `\l__zrefclever_type_first_label_tl`, but it also expected to be called right after `__zrefclever_type_name_setup:` which sets `\l__zrefclever_type_name_tl` and `\l__zrefclever_name_in_link_bool` which it uses.

```

2061 \cs_new:Npn \__zrefclever_get_ref_first:

```

```

2062 {
2063   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2064   { \exp_not:N \__zrefclever_ref_default: }
2065   {
2066     \bool_if:NTF \l__zrefclever_name_in_link_bool
2067     {
2068       \zref@ifrefcontainsprop
2069       { \l__zrefclever_type_first_label_tl }
2070       { \l__zrefclever_ref_property_tl }
2071       {
2072         % It's two '@s', but escaped for DocStrip.
2073         \exp_not:N \hyper@@link
2074         {
2075           \zref@ifrefcontainsprop
2076           { \l__zrefclever_type_first_label_tl } { urluse }
2077           {
2078             \zref@extractdefault
2079             { \l__zrefclever_type_first_label_tl }
2080             { urluse } { }
2081           }
2082           {
2083             \zref@extractdefault
2084             { \l__zrefclever_type_first_label_tl }
2085             { url } { }
2086           }
2087         }
2088       }
2089       \zref@extractdefault
2090       { \l__zrefclever_type_first_label_tl }
2091       { anchor } { }
2092     }
2093     {
2094       \exp_not:N \group_begin:
2095       \exp_not:N \l__zrefclever_namefont_tl
2096       \exp_not:N \l__zrefclever_type_name_tl
2097       \exp_not:N \group_end:
2098       \exp_not:N \l__zrefclever_namesep_tl
2099       \exp_not:N \group_begin:
2100       \exp_not:N \l__zrefclever_reffont_out_tl
2101       \exp_not:N \l__zrefclever_refpre_out_tl
2102       \exp_not:N \group_begin:
2103       \exp_not:N \l__zrefclever_reffont_in_tl
2104       \exp_not:N \l__zrefclever_refpre_in_tl
2105       \zref@extractdefault
2106       { \l__zrefclever_type_first_label_tl }
2107       { \l__zrefclever_ref_property_tl } { }
2108       \exp_not:N \l__zrefclever_refpos_in_tl
2109       \exp_not:N \group_end:
2110       % hyperlink makes it's own group, we'd like to close the
2111       % 'refpre-out' group after 'refpos-out', but... we close
2112       % it here, and give the trailing 'refpos-out' its own
2113       % group. This will result that formatting given to
2114       % 'refpre-out' will not reach 'refpos-out', but I see no
2115       % alternative, and this has to be handled specially.

```

```

2116         \exp_not:N \group_end:
2117     }
2118     \exp_not:N \group_begin:
2119     % Ditto: special treatment.
2120     \exp_not:V \l__zrefclever_reffont_out_tl
2121     \exp_not:V \l__zrefclever_refpos_out_tl
2122     \exp_not:N \group_end:
2123 }
2124 {
2125     \exp_not:N \group_begin:
2126     \exp_not:V \l__zrefclever_namefont_tl
2127     \exp_not:V \l__zrefclever_type_name_tl
2128     \exp_not:N \group_end:
2129     \exp_not:V \l__zrefclever_namesep_tl
2130     \exp_not:N \__zrefclever_ref_default:
2131 }
2132 }
2133 {
2134     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2135     {
2136         \exp_not:N \__zrefclever_name_default:
2137         \exp_not:V \l__zrefclever_namesep_tl
2138     }
2139     {
2140         \exp_not:N \group_begin:
2141         \exp_not:V \l__zrefclever_namefont_tl
2142         \exp_not:V \l__zrefclever_type_name_tl
2143         \exp_not:N \group_end:
2144         \exp_not:V \l__zrefclever_namesep_tl
2145     }
2146     \zref@ifrefcontainsprop
2147     { \l__zrefclever_type_first_label_tl }
2148     { \l__zrefclever_ref_property_tl }
2149     {
2150         \bool_if:nTF
2151         {
2152             \l__zrefclever_use_hyperref_bool &&
2153             ! \l__zrefclever_link_star_bool
2154         }
2155         {
2156             \exp_not:N \group_begin:
2157             \exp_not:V \l__zrefclever_reffont_out_tl
2158             \exp_not:V \l__zrefclever_refpre_out_tl
2159             \exp_not:N \group_begin:
2160             \exp_not:V \l__zrefclever_reffont_in_tl
2161             % It's two '@s', but escaped for DocStrip.
2162             \exp_not:N \hyper@link
2163             {
2164                 \zref@ifrefcontainsprop
2165                 { \l__zrefclever_type_first_label_tl } { urluse }
2166                 {
2167                     \zref@extractdefault
2168                     { \l__zrefclever_type_first_label_tl }
2169                     { urluse } { }

```

```

2170         }
2171         {
2172             \zref@extractdefault
2173             { \l__zrefclever_type_first_label_tl }
2174             { url } { }
2175         }
2176     }
2177     {
2178         \zref@extractdefault
2179         { \l__zrefclever_type_first_label_tl }
2180         { anchor } { }
2181     }
2182     {
2183         \exp_not:N \l__zrefclever_refpre_in_tl
2184         \zref@extractdefault
2185         { \l__zrefclever_type_first_label_tl }
2186         { \l__zrefclever_ref_property_tl } { }
2187         \exp_not:N \l__zrefclever_refpos_in_tl
2188     }
2189     \exp_not:N \group_end:
2190     \exp_not:N \l__zrefclever_refpos_out_tl
2191     \exp_not:N \group_end:
2192 }
2193 {
2194     \exp_not:N \group_begin:
2195     \exp_not:N \l__zrefclever_reffont_out_tl
2196     \exp_not:N \l__zrefclever_refpre_out_tl
2197     \exp_not:N \group_begin:
2198     \exp_not:N \l__zrefclever_reffont_in_tl
2199     \exp_not:N \l__zrefclever_refpre_in_tl
2200     \zref@extractdefault
2201     { \l__zrefclever_type_first_label_tl }
2202     { \l__zrefclever_ref_property_tl } { }
2203     \exp_not:N \l__zrefclever_refpos_in_tl
2204     \exp_not:N \group_end:
2205     \exp_not:N \l__zrefclever_refpos_out_tl
2206     \exp_not:N \group_end:
2207 }
2208 }
2209 { \exp_not:N \__zrefclever_ref_default: }
2210 }
2211 }
2212 }

```

(End definition for __zrefclever_get_ref_first:.)

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

prominently `\l__zrefclever_type_first_label_type_tl`, but also the queue itself in `\l__zrefclever_typeset_queue_curr_tl`, which should be “ready except for the first label”, and the type counter `\l__zrefclever_type_count_int`.

```

2213 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2214 {
2215   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2216   { \tl_clear:N \l__zrefclever_type_name_tl }
2217   {
2218     \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2219     { \tl_clear:N \l__zrefclever_type_name_tl }
2220     {
2221       % Determine whether we should use capitalization, abbreviation,
2222       % and plural.
2223       \bool_lazy_or:nnTF
2224       { \l__zrefclever_capitalize_bool }
2225       {
2226         \l__zrefclever_capitalize_first_bool &&
2227         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2228       }
2229       { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2230       { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2231       % If the queue is empty, we have a singular, otherwise, plural.
2232       \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2233       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2234       { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2235       \bool_lazy_and:nnTF
2236       { \l__zrefclever_abbrev_bool }
2237       {
2238         ! \int_compare_p:nNn
2239         { \l__zrefclever_type_count_int } = { 0 } ||
2240         ! \l__zrefclever_noabbrev_first_bool
2241       }
2242       {
2243         \tl_set:NV \l__zrefclever_name_format_fallback_tl
2244         \l__zrefclever_name_format_tl
2245         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2246       }
2247       { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2248
2249       \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2250       {
2251         \prop_get:cVNF
2252         {
2253           \l__zrefclever_type_
2254           \l__zrefclever_type_first_label_type_tl _options_prop
2255         }
2256         \l__zrefclever_name_format_tl
2257         \l__zrefclever_type_name_tl
2258         {
2259           \__zrefclever_get_type_transl:xxxNF
2260           { \l__zrefclever_ref_language_tl }
2261           { \l__zrefclever_type_first_label_type_tl }
2262           { \l__zrefclever_name_format_tl }
2263           \l__zrefclever_type_name_tl

```

```

2264         {
2265             \tl_clear:N \l__zrefclever_type_name_tl
2266             \msg_warning:nnx { zref-clever } { missing-name }
2267             { \l__zrefclever_type_first_label_type_tl }
2268         }
2269     }
2270 }
2271 {
2272     \prop_get:cVNF
2273     {
2274         l__zrefclever_type_
2275         \l__zrefclever_type_first_label_type_tl _options_prop
2276     }
2277     \l__zrefclever_name_format_tl
2278     \l__zrefclever_type_name_tl
2279     {
2280         \prop_get:cVNF
2281         {
2282             l__zrefclever_type_
2283             \l__zrefclever_type_first_label_type_tl _options_prop
2284         }
2285         \l__zrefclever_name_format_fallback_tl
2286         \l__zrefclever_type_name_tl
2287         {
2288             \__zrefclever_get_type_transl:xxxNF
2289             { \l__zrefclever_ref_language_tl }
2290             { \l__zrefclever_type_first_label_type_tl }
2291             { \l__zrefclever_name_format_tl }
2292             \l__zrefclever_type_name_tl
2293             {
2294                 \__zrefclever_get_type_transl:xxxNF
2295                 { \l__zrefclever_ref_language_tl }
2296                 { \l__zrefclever_type_first_label_type_tl }
2297                 { \l__zrefclever_name_format_fallback_tl }
2298                 \l__zrefclever_type_name_tl
2299                 {
2300                     \tl_clear:N \l__zrefclever_type_name_tl
2301                     \msg_warning:nnx { zref-clever }
2302                     { missing-name }
2303                     { \l__zrefclever_type_first_label_type_tl }
2304                 }
2305             }
2306         }
2307     }
2308 }
2309 }
2310 }
2311
2312 % Signal whether the type name is to be included in the hyperlink or not.
2313 \bool_lazy_any:nTF
2314 {
2315     { ! \l__zrefclever_use_hyperref_bool }
2316     { \l__zrefclever_link_star_bool }
2317     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }

```

```

2318     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2319   }
2320   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2321   {
2322     \bool_lazy_any:nTF
2323     {
2324       { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2325       {
2326         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2327         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2328       }
2329       {
2330         \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2331         \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2332         \l__zrefclever_typeset_last_bool &&
2333         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2334       }
2335     }
2336     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2337     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2338   }
2339 }

```

(End definition for __zrefclever_type_name_setup:.)

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

```

\__zrefclever_labels_in_sequence:nn {(label a)} {(label b)}

2340 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2341 {
2342   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2343   {
2344     \exp_args:Nxx \tl_if_eq:nnT
2345     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2346     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2347     {
2348       \int_compare:nNnTF
2349       { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2350       =
2351       { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2352       { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2353       {
2354         \int_compare:nNnT
2355         { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2356         =
2357         { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2358         {
2359           \bool_set_true:N \l__zrefclever_next_maybe_range_bool

```

```

2360         \bool_set_true:N \l__zrefclever_next_is_same_bool
2361     }
2362 }
2363 }
2364 }
2365 {
2366     \exp_args:Nxx \tl_if_eq:nnT
2367     { \zref@extractdefault {#1} { counter } { } }
2368     { \zref@extractdefault {#2} { counter } { } }
2369     {
2370         \exp_args:Nxx \tl_if_eq:nnT
2371         { \zref@extractdefault {#1} { zc@enclval } { } }
2372         { \zref@extractdefault {#2} { zc@enclval } { } }
2373         {
2374             \int_compare:nNnTF
2375             { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2376             =
2377             { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2378             { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2379             {
2380                 \int_compare:nNnT
2381                 { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2382                 =
2383                 { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2384                 {
2385                     \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2386                     \bool_set_true:N \l__zrefclever_next_is_same_bool
2387                 }
2388             }
2389         }
2390     }
2391 }
2392 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

Finally, a couple of functions for retrieving options values, according to the relevant precedence rules (see Section 4.2). They both receive an *option* as argument, and store the retrieved value in *tl variable*. Though these are mostly general functions (for a change...), they are not completely so, they rely on the current state of `\l__zrefclever_label_type_a_tl`, as set during the processing of the label stack. This could be easily generalized, of course, but I don't think it is worth it, `\l__zrefclever_label_type_a_tl` is indeed what we want in all practical cases. The difference between `__zrefclever_get_ref_string:nN` and `__zrefclever_get_ref_font:nN` is the kind of option each should be used for. `__zrefclever_get_ref_string:nN` is meant for the general options, and attempts to find values for them in all precedence levels (four plus “fallback”). `__zrefclever_get_ref_font:nN` is intended for “font” options, which cannot be “language-specific”, thus for these we just search general options and type options.

```

\__zrefclever_get_ref_string:nN      \__zrefclever_get_ref_string:nN {<option>} {<tl variable>}
2393 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2394 {
2395     % First attempt: general options.

```



```

2396 \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2397 {
2398   % If not found, try type specific options.
2399   \bool_lazy_all:nTF
2400   {
2401     { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2402     {
2403       \prop_if_exist_p:c
2404       {
2405         l__zrefclever_type_
2406         \l__zrefclever_label_type_a_tl _options_prop
2407       }
2408     }
2409     {
2410       \prop_if_in_p:cn
2411       {
2412         l__zrefclever_type_
2413         \l__zrefclever_label_type_a_tl _options_prop
2414       }
2415       {#1}
2416     }
2417   }
2418   {
2419     \prop_get:cnN
2420     {
2421       l__zrefclever_type_
2422       \l__zrefclever_label_type_a_tl _options_prop
2423     }
2424     {#1} #2
2425   }
2426   {
2427     % If not found, try type specific translations.
2428     \__zrefclever_get_type_transl:xxnNF
2429     { \l__zrefclever_ref_language_tl }
2430     { \l__zrefclever_label_type_a_tl }
2431     {#1} #2
2432     {
2433       % If not found, try default translations.
2434       \__zrefclever_get_default_transl:xxnNF
2435       { \l__zrefclever_ref_language_tl }
2436       {#1} #2
2437       {
2438         % If not found, try fallback.
2439         \__zrefclever_get_fallback_transl:nNF {#1} #2
2440         {
2441           \tl_clear:N #2
2442           \msg_warning:nnn { zref-clever }
2443             { missing-string } {#1}
2444         }
2445       }
2446     }
2447   }
2448 }
2449 }

```

(End definition for `_zrefclever_get_ref_string:nN`.)

```

\_zrefclever_get_ref_font:nN      \_zrefclever_get_ref_font:nN {\langle option \rangle} {\langle tl variable \rangle}
2450 \cs_new_protected:Npn \_zrefclever_get_ref_font:nN #1#2
2451 {
2452   % First attempt: general options.
2453   \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2454   {
2455     % If not found, try type specific options.
2456     \bool_lazy_and:nnTF
2457       { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2458       {
2459         \prop_if_exist_p:c
2460         {
2461           l__zrefclever_type_
2462           \l__zrefclever_label_type_a_tl _options_prop
2463         }
2464       }
2465       {
2466         \prop_get:cnNF
2467         {
2468           l__zrefclever_type_
2469           \l__zrefclever_label_type_a_tl _options_prop
2470         }
2471         {#1} #2
2472         { \tl_clear:N #2 }
2473       }
2474       { \tl_clear:N #2 }
2475   }
2476 }

```

(End definition for `_zrefclever_get_ref_font: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 enumitem package

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

```
2477 \endpackage
```

10 Dictionaries

10.1 English

```
2478 \zcdDeclareLanguage { english }
2479 \zcdDeclareLanguageAlias { american } { english }
2480 \zcdDeclareLanguageAlias { australian } { english }
2481 \zcdDeclareLanguageAlias { british } { english }
2482 \zcdDeclareLanguageAlias { canadian } { english }
2483 \zcdDeclareLanguageAlias { newzealand } { english }
2484 \zcdDeclareLanguageAlias { UKenglish } { english }
2485 \zcdDeclareLanguageAlias { USenglish } { english }

2486 \dict-english

2487 namesep = {\nobreakspace} ,
2488 pairsep = {\and\nobreakspace} ,
2489 listsep = {,~} ,
2490 lastsep = {\and\nobreakspace} ,
2491 tpairsep = {\and\nobreakspace} ,
2492 tlistsep = {,~} ,
2493 tlastsep = {,~\and\nobreakspace} ,
2494 notesep = {~} ,
2495 rangesep = {\to\nobreakspace} ,

2496
2497 type = part ,
2498   Name-sg = Part ,
2499   name-sg = part ,
2500   Name-pl = Parts ,
2501   name-pl = parts ,

2502
2503 type = chapter ,
2504   Name-sg = Chapter ,
2505   name-sg = chapter ,
2506   Name-pl = Chapters ,
2507   name-pl = chapters ,

2508
2509 type = section ,
2510   Name-sg = Section ,
2511   name-sg = section ,
2512   Name-pl = Sections ,
2513   name-pl = sections ,

2514
2515 type = paragraph ,
2516   Name-sg = Paragraph ,
```

```

2517     name-sg = paragraph ,
2518     Name-pl = Paragraphs ,
2519     name-pl = paragraphs ,
2520     Name-sg-ab = Par. ,
2521     name-sg-ab = par. ,
2522     Name-pl-ab = Par. ,
2523     name-pl-ab = par. ,
2524
2525     type = appendix ,
2526     Name-sg = Appendix ,
2527     name-sg = appendix ,
2528     Name-pl = Appendices ,
2529     name-pl = appendices ,
2530
2531     type = page ,
2532     Name-sg = Page ,
2533     name-sg = page ,
2534     Name-pl = Pages ,
2535     name-pl = pages ,
2536     name-sg-ab = p. ,
2537     name-pl-ab = pp. ,
2538
2539     type = line ,
2540     Name-sg = Line ,
2541     name-sg = line ,
2542     Name-pl = Lines ,
2543     name-pl = lines ,
2544
2545     type = figure ,
2546     Name-sg = Figure ,
2547     name-sg = figure ,
2548     Name-pl = Figures ,
2549     name-pl = figures ,
2550     Name-sg-ab = Fig. ,
2551     name-sg-ab = fig. ,
2552     Name-pl-ab = Figs. ,
2553     name-pl-ab = figs. ,
2554
2555     type = table ,
2556     Name-sg = Table ,
2557     name-sg = table ,
2558     Name-pl = Tables ,
2559     name-pl = tables ,
2560
2561     type = item ,
2562     Name-sg = Item ,
2563     name-sg = item ,
2564     Name-pl = Items ,
2565     name-pl = items ,
2566
2567     type = footnote ,
2568     Name-sg = Footnote ,
2569     name-sg = footnote ,
2570     Name-pl = Footnotes ,

```

```

2571     name-pl = footnotes ,
2572
2573 type = note ,
2574     Name-sg = Note ,
2575     name-sg = note ,
2576     Name-pl = Notes ,
2577     name-pl = notes ,
2578
2579 type = equation ,
2580     Name-sg = Equation ,
2581     name-sg = equation ,
2582     Name-pl = Equations ,
2583     name-pl = equations ,
2584     Name-sg-ab = Eq. ,
2585     name-sg-ab = eq. ,
2586     Name-pl-ab = Eqs. ,
2587     name-pl-ab = eqs. ,
2588     refpre-in = {() ,
2589     refpos-in = {} } ,
2590
2591 type = theorem ,
2592     Name-sg = Theorem ,
2593     name-sg = theorem ,
2594     Name-pl = Theorems ,
2595     name-pl = theorems ,
2596
2597 type = lemma ,
2598     Name-sg = Lemma ,
2599     name-sg = lemma ,
2600     Name-pl = Lemmas ,
2601     name-pl = lemmas ,
2602
2603 type = corollary ,
2604     Name-sg = Corollary ,
2605     name-sg = corollary ,
2606     Name-pl = Corollaries ,
2607     name-pl = corollaries ,
2608
2609 type = proposition ,
2610     Name-sg = Proposition ,
2611     name-sg = proposition ,
2612     Name-pl = Propositions ,
2613     name-pl = propositions ,
2614
2615 type = definition ,
2616     Name-sg = Definition ,
2617     name-sg = definition ,
2618     Name-pl = Definitions ,
2619     name-pl = definitions ,
2620
2621 type = proof ,
2622     Name-sg = Proof ,
2623     name-sg = proof ,
2624     Name-pl = Proofs ,

```

```

2625     name-pl = proofs ,
2626
2627 type = result ,
2628     Name-sg = Result ,
2629     name-sg = result ,
2630     Name-pl = Results ,
2631     name-pl = results ,
2632
2633 type = example ,
2634     Name-sg = Example ,
2635     name-sg = example ,
2636     Name-pl = Examples ,
2637     name-pl = examples ,
2638
2639 type = remark ,
2640     Name-sg = Remark ,
2641     name-sg = remark ,
2642     Name-pl = Remarks ,
2643     name-pl = remarks ,
2644
2645 type = algorithm ,
2646     Name-sg = Algorithm ,
2647     name-sg = algorithm ,
2648     Name-pl = Algorithms ,
2649     name-pl = algorithms ,
2650
2651 type = listing ,
2652     Name-sg = Listing ,
2653     name-sg = listing ,
2654     Name-pl = Listings ,
2655     name-pl = listings ,
2656
2657 type = exercise ,
2658     Name-sg = Exercise ,
2659     name-sg = exercise ,
2660     Name-pl = Exercises ,
2661     name-pl = exercises ,
2662
2663 type = solution ,
2664     Name-sg = Solution ,
2665     name-sg = solution ,
2666     Name-pl = Solutions ,
2667     name-pl = solutions ,
2668 </dict-english>

```

10.2 German

```

2669 <package>\zcDeclareLanguage { german }
2670 <package>\zcDeclareLanguageAlias { austrian      } { german }
2671 <package>\zcDeclareLanguageAlias { germanb       } { german }
2672 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2673 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2674 <package>\zcDeclareLanguageAlias { nswissgerman } { german }
2675 <package>\zcDeclareLanguageAlias { swissgerman  } { german }

```

```

2676 (*dict-german)
2677 namesep = {\nobreakspace} ,
2678 pairsep = {\~und\nobreakspace} ,
2679 listsep = {,~} ,
2680 lastsep = {\~und\nobreakspace} ,
2681 tpairsep = {\~und\nobreakspace} ,
2682 tlistsep = {,~} ,
2683 tlastsep = {\~und\nobreakspace} ,
2684 notesep = {~} ,
2685 rangesep = {\~bis\nobreakspace} ,
2686
2687 type = part ,
2688   Name-sg = Teil ,
2689   name-sg = Teil ,
2690   Name-pl = Teile ,
2691   name-pl = Teile ,
2692
2693 type = chapter ,
2694   Name-sg = Kapitel ,
2695   name-sg = Kapitel ,
2696   Name-pl = Kapitel ,
2697   name-pl = Kapitel ,
2698
2699 type = section ,
2700   Name-sg = Abschnitt ,
2701   name-sg = Abschnitt ,
2702   Name-pl = Abschnitte ,
2703   name-pl = Abschnitte ,
2704
2705 type = paragraph ,
2706   Name-sg = Absatz ,
2707   name-sg = Absatz ,
2708   Name-pl = Absätze ,
2709   name-pl = Absätze ,
2710
2711 type = appendix ,
2712   Name-sg = Anhang ,
2713   name-sg = Anhang ,
2714   Name-pl = Anhänge ,
2715   name-pl = Anhänge ,
2716
2717 type = page ,
2718   Name-sg = Seite ,
2719   name-sg = Seite ,
2720   Name-pl = Seiten ,
2721   name-pl = Seiten ,
2722
2723 type = line ,
2724   Name-sg = Zeile ,
2725   name-sg = Zeile ,
2726   Name-pl = Zeilen ,
2727   name-pl = Zeilen ,
2728
2729 type = figure ,

```

```

2730 Name-sg = Abbildung ,
2731 name-sg = Abbildung ,
2732 Name-pl = Abbildungen ,
2733 name-pl = Abbildungen ,
2734 Name-sg-ab = Abb. ,
2735 name-sg-ab = Abb. ,
2736 Name-pl-ab = Abb. ,
2737 name-pl-ab = Abb. ,
2738
2739 type = table ,
2740 Name-sg = Tabelle ,
2741 name-sg = Tabelle ,
2742 Name-pl = Tabellen ,
2743 name-pl = Tabellen ,
2744
2745 type = item ,
2746 Name-sg = Punkt ,
2747 name-sg = Punkt ,
2748 Name-pl = Punkte ,
2749 name-pl = Punkte ,
2750
2751 type = footnote ,
2752 Name-sg = Fußnote ,
2753 name-sg = Fußnote ,
2754 Name-pl = Fußnoten ,
2755 name-pl = Fußnoten ,
2756
2757 type = note ,
2758 Name-sg = Anmerkung ,
2759 name-sg = Anmerkung ,
2760 Name-pl = Anmerkungen ,
2761 name-pl = Anmerkungen ,
2762
2763 type = equation ,
2764 Name-sg = Gleichung ,
2765 name-sg = Gleichung ,
2766 Name-pl = Gleichungen ,
2767 name-pl = Gleichungen ,
2768 refpre-in = {() ,
2769 refpos-in = {()} ,
2770
2771 type = theorem ,
2772 Name-sg = Theorem ,
2773 name-sg = Theorem ,
2774 Name-pl = Theoreme ,
2775 name-pl = Theoreme ,
2776
2777 type = lemma ,
2778 Name-sg = Lemma ,
2779 name-sg = Lemma ,
2780 Name-pl = Lemmata ,
2781 name-pl = Lemmata ,
2782
2783 type = corollary ,

```



```

2784   Name-sg = Korollar ,
2785   name-sg = Korollar ,
2786   Name-pl = Korollare ,
2787   name-pl = Korollare ,
2788
2789   type = proposition ,
2790   Name-sg = Satz ,
2791   name-sg = Satz ,
2792   Name-pl = Sätze ,
2793   name-pl = Sätze ,
2794
2795   type = definition ,
2796   Name-sg = Definition ,
2797   name-sg = Definition ,
2798   Name-pl = Definitionen ,
2799   name-pl = Definitionen ,
2800
2801   type = proof ,
2802   Name-sg = Beweis ,
2803   name-sg = Beweis ,
2804   Name-pl = Beweise ,
2805   name-pl = Beweise ,
2806
2807   type = result ,
2808   Name-sg = Ergebnis ,
2809   name-sg = Ergebnis ,
2810   Name-pl = Ergebnisse ,
2811   name-pl = Ergebnisse ,
2812
2813   type = example ,
2814   Name-sg = Beispiel ,
2815   name-sg = Beispiel ,
2816   Name-pl = Beispiele ,
2817   name-pl = Beispiele ,
2818
2819   type = remark ,
2820   Name-sg = Bemerkung ,
2821   name-sg = Bemerkung ,
2822   Name-pl = Bemerkungen ,
2823   name-pl = Bemerkungen ,
2824
2825   type = algorithm ,
2826   Name-sg = Algorithmus ,
2827   name-sg = Algorithmus ,
2828   Name-pl = Algorithmen ,
2829   name-pl = Algorithmen ,
2830
2831   type = listing ,
2832   Name-sg = Listing , % CHECK
2833   name-sg = Listing , % CHECK
2834   Name-pl = Listings , % CHECK
2835   name-pl = Listings , % CHECK
2836
2837   type = exercise ,

```

```

2838   Name-sg = Übungsaufgabe ,
2839   name-sg = Übungsaufgabe ,
2840   Name-pl = Übungsaufgaben ,
2841   name-pl = Übungsaufgaben ,
2842
2843   type = solution ,
2844   Name-sg = Lösung ,
2845   name-sg = Lösung ,
2846   Name-pl = Lösungen ,
2847   name-pl = Lösungen ,
2848   </dict-german>

```

10.3 French

```

2849 <package>\zcDeclareLanguage { french }
2850 <package>\zcDeclareLanguageAlias { acadian } { french }
2851 <package>\zcDeclareLanguageAlias { canadien } { french }
2852 <package>\zcDeclareLanguageAlias { francais } { french }
2853 <package>\zcDeclareLanguageAlias { frenchb } { french }
2854 <*dict-french>
2855 namesep = {\nobreakspace} ,
2856 pairsep = {\~et\nobreakspace} ,
2857 listsep = {\~,~} ,
2858 lastsep = {\~et\nobreakspace} ,
2859 tpairsep = {\~et\nobreakspace} ,
2860 tlistsep = {\~,~} ,
2861 tlastsep = {\~et\nobreakspace} ,
2862 notesep = {\~} ,
2863 rangesep = {\~à\nobreakspace} ,
2864
2865 type = part ,
2866   Name-sg = Partie ,
2867   name-sg = partie ,
2868   Name-pl = Parties ,
2869   name-pl = parties ,
2870
2871 type = chapter ,
2872   Name-sg = Chapitre ,
2873   name-sg = chapitre ,
2874   Name-pl = Chapitres ,
2875   name-pl = chapitres ,
2876
2877 type = section ,
2878   Name-sg = Section ,
2879   name-sg = section ,
2880   Name-pl = Sections ,
2881   name-pl = sections ,
2882
2883 type = paragraph ,
2884   Name-sg = Paragraphe ,
2885   name-sg = paragraphe ,
2886   Name-pl = Paragraphes ,
2887   name-pl = paragraphes ,
2888

```

```

2889 type = appendix ,
2890     Name-sg = Annexe ,
2891     name-sg = annexe ,
2892     Name-pl = Annexes ,
2893     name-pl = annexes ,
2894
2895 type = page ,
2896     Name-sg = Page ,
2897     name-sg = page ,
2898     Name-pl = Pages ,
2899     name-pl = pages ,
2900
2901 type = line ,
2902     Name-sg = Ligne ,
2903     name-sg = ligne ,
2904     Name-pl = Lignes ,
2905     name-pl = lignes ,
2906
2907 type = figure ,
2908     Name-sg = Figure ,
2909     name-sg = figure ,
2910     Name-pl = Figures ,
2911     name-pl = figures ,
2912
2913 type = table ,
2914     Name-sg = Table ,
2915     name-sg = table ,
2916     Name-pl = Tables ,
2917     name-pl = tables ,
2918
2919 type = item ,
2920     Name-sg = Point ,
2921     name-sg = point ,
2922     Name-pl = Points ,
2923     name-pl = points ,
2924
2925 type = footnote ,
2926     Name-sg = Note ,
2927     name-sg = note ,
2928     Name-pl = Notes ,
2929     name-pl = notes ,
2930
2931 type = note ,
2932     Name-sg = Note ,
2933     name-sg = note ,
2934     Name-pl = Notes ,
2935     name-pl = notes ,
2936
2937 type = equation ,
2938     Name-sg = Équation ,
2939     name-sg = équation ,
2940     Name-pl = Équations ,
2941     name-pl = équations ,
2942     refpre-in = {() ,

```

```

2943   refpos-in = {} } ,
2944
2945   type = theorem ,
2946     Name-sg = Théorème ,
2947     name-sg = théorème ,
2948     Name-pl = Théorèmes ,
2949     name-pl = théorèmes ,
2950
2951   type = lemma ,
2952     Name-sg = Lemme ,
2953     name-sg = lemme ,
2954     Name-pl = Lemmes ,
2955     name-pl = lemmes ,
2956
2957   type = corollary ,
2958     Name-sg = Corollaire ,
2959     name-sg = corollaire ,
2960     Name-pl = Corollaires ,
2961     name-pl = corollaires ,
2962
2963   type = proposition ,
2964     Name-sg = Proposition ,
2965     name-sg = proposition ,
2966     Name-pl = Propositions ,
2967     name-pl = propositions ,
2968
2969   type = definition ,
2970     Name-sg = Définition ,
2971     name-sg = définition ,
2972     Name-pl = Définitions ,
2973     name-pl = définitions ,
2974
2975   type = proof ,
2976     Name-sg = Démonstration ,
2977     name-sg = démonstration ,
2978     Name-pl = Démonstrations ,
2979     name-pl = démonstrations ,
2980
2981   type = result ,
2982     Name-sg = Résultat ,
2983     name-sg = résultat ,
2984     Name-pl = Résultats ,
2985     name-pl = résultats ,
2986
2987   type = example ,
2988     Name-sg = Exemple ,
2989     name-sg = exemple ,
2990     Name-pl = Exemples ,
2991     name-pl = exemples ,
2992
2993   type = remark ,
2994     Name-sg = Remarque ,
2995     name-sg = remarque ,
2996     Name-pl = Remarques ,

```

```

2997   name-pl = remarques ,
2998
2999   type = algorithm ,
3000     Name-sg = Algorithme ,
3001     name-sg = algorithme ,
3002     Name-pl = Algorithmes ,
3003     name-pl = algorithmes ,
3004
3005   type = listing ,
3006     Name-sg = Liste ,
3007     name-sg = liste ,
3008     Name-pl = Listes ,
3009     name-pl = listes ,
3010
3011   type = exercise ,
3012     Name-sg = Exercice ,
3013     name-sg = exercice ,
3014     Name-pl = Exercices ,
3015     name-pl = exercices ,
3016
3017   type = solution ,
3018     Name-sg = Solution ,
3019     name-sg = solution ,
3020     Name-pl = Solutions ,
3021     name-pl = solutions ,
3022 </dict-french>

```

10.4 Portuguese

```

3023 <package>\zcDeclareLanguage { portuguese }
3024 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3025 <package>\zcDeclareLanguageAlias { brazil   } { portuguese }
3026 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3027 <*dict-portuguese>

3028 namesep = {\nobreakspace} ,
3029 pairsep  = {\nobreakspace} ,
3030 listsep  = {,~} ,
3031 lastsep  = {\nobreakspace} ,
3032 tpairsep = {\nobreakspace} ,
3033 tlistsep = {,~} ,
3034 tlastsep = {\nobreakspace} ,
3035 notesep  = {~} ,
3036 rangesep = {\nobreakspace} ,
3037
3038 type = part ,
3039   Name-sg = Parte ,
3040   name-sg = parte ,
3041   Name-pl = Partes ,
3042   name-pl = partes ,
3043
3044 type = chapter ,
3045   Name-sg = Capítulo ,
3046   name-sg = capítulo ,
3047   Name-pl = Capítulos ,

```

```

3048     name-pl = capítulos ,
3049
3050 type = section ,
3051     Name-sg = Seção ,
3052     name-sg = seção ,
3053     Name-pl = Seções ,
3054     name-pl = seções ,
3055
3056 type = paragraph ,
3057     Name-sg = Parágrafo ,
3058     name-sg = parágrafo ,
3059     Name-pl = Parágrafos ,
3060     name-pl = parágrafos ,
3061     Name-sg-ab = Par. ,
3062     name-sg-ab = par. ,
3063     Name-pl-ab = Par. ,
3064     name-pl-ab = par. ,
3065
3066 type = appendix ,
3067     Name-sg = Apêndice ,
3068     name-sg = apêndice ,
3069     Name-pl = Apêndices ,
3070     name-pl = apêndices ,
3071
3072 type = page ,
3073     Name-sg = Página ,
3074     name-sg = página ,
3075     Name-pl = Páginas ,
3076     name-pl = páginas ,
3077     name-sg-ab = p. ,
3078     name-pl-ab = pp. ,
3079
3080 type = line ,
3081     Name-sg = Linha ,
3082     name-sg = linha ,
3083     Name-pl = Linhas ,
3084     name-pl = linhas ,
3085
3086 type = figure ,
3087     Name-sg = Figura ,
3088     name-sg = figura ,
3089     Name-pl = Figuras ,
3090     name-pl = figuras ,
3091     Name-sg-ab = Fig. ,
3092     name-sg-ab = fig. ,
3093     Name-pl-ab = Figs. ,
3094     name-pl-ab = figs. ,
3095
3096 type = table ,
3097     Name-sg = Tabela ,
3098     name-sg = tabela ,
3099     Name-pl = Tabelas ,
3100     name-pl = tabelas ,
3101

```

```

3102 type = item ,
3103     Name-sg = Item ,
3104     name-sg = item ,
3105     Name-pl = Itens ,
3106     name-pl = itens ,
3107
3108 type = footnote ,
3109     Name-sg = Nota ,
3110     name-sg = nota ,
3111     Name-pl = Notas ,
3112     name-pl = notas ,
3113
3114 type = note ,
3115     Name-sg = Nota ,
3116     name-sg = nota ,
3117     Name-pl = Notas ,
3118     name-pl = notas ,
3119
3120 type = equation ,
3121     Name-sg = Equação ,
3122     name-sg = equação ,
3123     Name-pl = Equações ,
3124     name-pl = equações ,
3125     Name-sg-ab = Eq. ,
3126     name-sg-ab = eq. ,
3127     Name-pl-ab = Eqs. ,
3128     name-pl-ab = eqs. ,
3129     refpre-in = {(} ,
3130     refpos-in = {)} ,
3131
3132 type = theorem ,
3133     Name-sg = Teorema ,
3134     name-sg = teorema ,
3135     Name-pl = Teoremas ,
3136     name-pl = teoremas ,
3137
3138 type = lemma ,
3139     Name-sg = Lema ,
3140     name-sg = lema ,
3141     Name-pl = Lemas ,
3142     name-pl = lemas ,
3143
3144 type = corollary ,
3145     Name-sg = Corolário ,
3146     name-sg = corolário ,
3147     Name-pl = Corolários ,
3148     name-pl = corolários ,
3149
3150 type = proposition ,
3151     Name-sg = Proposição ,
3152     name-sg = proposição ,
3153     Name-pl = Proposições ,
3154     name-pl = proposições ,
3155

```

```

3156 type = definition ,
3157     Name-sg = Definição ,
3158     name-sg = definição ,
3159     Name-pl = Definições ,
3160     name-pl = definições ,
3161
3162 type = proof ,
3163     Name-sg = Demonstração ,
3164     name-sg = demonstração ,
3165     Name-pl = Demonstrações ,
3166     name-pl = demonstrações ,
3167
3168 type = result ,
3169     Name-sg = Resultado ,
3170     name-sg = resultado ,
3171     Name-pl = Resultados ,
3172     name-pl = resultados ,
3173
3174 type = example ,
3175     Name-sg = Exemplo ,
3176     name-sg = exemplo ,
3177     Name-pl = Exemplos ,
3178     name-pl = exemplos ,
3179
3180 type = remark ,
3181     Name-sg = Observação ,
3182     name-sg = observação ,
3183     Name-pl = Observações ,
3184     name-pl = observações ,
3185
3186 type = algorithm ,
3187     Name-sg = Algoritmo ,
3188     name-sg = algoritmo ,
3189     Name-pl = Algoritmos ,
3190     name-pl = algoritmos ,
3191
3192 type = listing ,
3193     Name-sg = Listagem ,
3194     name-sg = listagem ,
3195     Name-pl = Listagens ,
3196     name-pl = listagens ,
3197
3198 type = exercise ,
3199     Name-sg = Exercício ,
3200     name-sg = exercício ,
3201     Name-pl = Exercícios ,
3202     name-pl = exercícios ,
3203
3204 type = solution ,
3205     Name-sg = Solução ,
3206     name-sg = solução ,
3207     Name-pl = Soluções ,
3208     name-pl = soluções ,
3209 </dict-portuguese>

```


10.5 Spanish

```
3210 \package\zcDeclareLanguage { spanish }
3211 \*dict-spanish)
3212 namesep = {\nobreakspace} ,
3213 pairsep = {\~y\nobreakspace} ,
3214 listsep = {,~} ,
3215 lastsep = {\~y\nobreakspace} ,
3216 tpairsep = {\~y\nobreakspace} ,
3217 tlistsep = {,~} ,
3218 tlastsep = {\~y\nobreakspace} ,
3219 notesep = {\~} ,
3220 rangesep = {\~a\nobreakspace} ,
3221
3222 type = part ,
3223   Name-sg = Parte ,
3224   name-sg = parte ,
3225   Name-pl = Partes ,
3226   name-pl = partes ,
3227
3228 type = chapter ,
3229   Name-sg = Capítulo ,
3230   name-sg = capítulo ,
3231   Name-pl = Capítulos ,
3232   name-pl = capítulos ,
3233
3234 type = section ,
3235   Name-sg = Sección ,
3236   name-sg = sección ,
3237   Name-pl = Secciones ,
3238   name-pl = secciones ,
3239
3240 type = paragraph ,
3241   Name-sg = Párrafo ,
3242   name-sg = párrafo ,
3243   Name-pl = Párrafos ,
3244   name-pl = párrafos ,
3245
3246 type = appendix ,
3247   Name-sg = Apéndice ,
3248   name-sg = apéndice ,
3249   Name-pl = Apéndices ,
3250   name-pl = apéndices ,
3251
3252 type = page ,
3253   Name-sg = Página ,
3254   name-sg = página ,
3255   Name-pl = Páginas ,
3256   name-pl = páginas ,
3257
3258 type = line ,
3259   Name-sg = Línea ,
3260   name-sg = línea ,
3261   Name-pl = Líneas ,
```

```

3262     name-pl = líneas ,
3263
3264 type = figure ,
3265     Name-sg = Figura ,
3266     name-sg = figura ,
3267     Name-pl = Figuras ,
3268     name-pl = figuras ,
3269
3270 type = table ,
3271     Name-sg = Cuadro ,
3272     name-sg = cuadro ,
3273     Name-pl = Cuadros ,
3274     name-pl = cuadros ,
3275
3276 type = item ,
3277     Name-sg = Punto ,
3278     name-sg = punto ,
3279     Name-pl = Puntos ,
3280     name-pl = puntos ,
3281
3282 type = footnote ,
3283     Name-sg = Nota ,
3284     name-sg = nota ,
3285     Name-pl = Notas ,
3286     name-pl = notas ,
3287
3288 type = note ,
3289     Name-sg = Nota ,
3290     name-sg = nota ,
3291     Name-pl = Notas ,
3292     name-pl = notas ,
3293
3294 type = equation ,
3295     Name-sg = Ecuación ,
3296     name-sg = ecuación ,
3297     Name-pl = Ecuaciones ,
3298     name-pl = ecuaciones ,
3299     refpre-in = {(} ,
3300     refpos-in = {)} ,
3301
3302 type = theorem ,
3303     Name-sg = Teorema ,
3304     name-sg = teorema ,
3305     Name-pl = Teoremas ,
3306     name-pl = teoremas ,
3307
3308 type = lemma ,
3309     Name-sg = Lema ,
3310     name-sg = lema ,
3311     Name-pl = Lemas ,
3312     name-pl = lemas ,
3313
3314 type = corollary ,
3315     Name-sg = Corolario ,

```

```

3316     name-sg = corolario ,
3317     Name-pl = Corolarios ,
3318     name-pl = corolarios ,
3319
3320 type = proposition ,
3321     Name-sg = Proposición ,
3322     name-sg = proposición ,
3323     Name-pl = Proposiciones ,
3324     name-pl = proposiciones ,
3325
3326 type = definition ,
3327     Name-sg = Definición ,
3328     name-sg = definición ,
3329     Name-pl = Definiciones ,
3330     name-pl = definiciones ,
3331
3332 type = proof ,
3333     Name-sg = Demostración ,
3334     name-sg = demostración ,
3335     Name-pl = Demostraciones ,
3336     name-pl = demostraciones ,
3337
3338 type = result ,
3339     Name-sg = Resultado ,
3340     name-sg = resultado ,
3341     Name-pl = Resultados ,
3342     name-pl = resultados ,
3343
3344 type = example ,
3345     Name-sg = Ejemplo ,
3346     name-sg = ejemplo ,
3347     Name-pl = Ejemplos ,
3348     name-pl = ejemplos ,
3349
3350 type = remark ,
3351     Name-sg = Observación ,
3352     name-sg = observación ,
3353     Name-pl = Observaciones ,
3354     name-pl = observaciones ,
3355
3356 type = algorithm ,
3357     Name-sg = Algoritmo ,
3358     name-sg = algoritmo ,
3359     Name-pl = Algoritmos ,
3360     name-pl = algoritmos ,
3361
3362 type = listing ,
3363     Name-sg = Listado ,
3364     name-sg = listado ,
3365     Name-pl = Listados ,
3366     name-pl = listados ,
3367
3368 type = exercise ,
3369     Name-sg = Ejercicio ,

```

```

3370 name-sg = ejercicio ,
3371 Name-pl = Ejercicios ,
3372 name-pl = ejercicios ,
3373
3374 type = solution ,
3375 Name-sg = Solución ,
3376 name-sg = solución ,
3377 Name-pl = Soluciones ,
3378 name-pl = soluciones ,
3379 ⟨/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, 118, 119, 124, 125, 134, 135, 145
A	
\AddToHook	91, 532, 547, 657, 725, 750, 779, 781, 832
\appendix	66
\appendixname	66
B	
\babelname	735
\babelprovide	12, 24
bool commands:	
\bool_case_true:	2
\bool_if:NTF	288, 297, 661, 665, 1481, 1576, 1706, 1728, 1759, 1816, 1857, 1880, 1884, 1890, 1900, 1906, 2066
\bool_if:nTF	59, 1137, 1146, 1155, 1212, 1240, 1261, 1347, 1355, 1495, 1503, 1740, 1747, 1754, 2015, 2150
\bool_lazy_all:nTF	2399
\bool_lazy_and:nnTF	1034, 1049, 2235, 2456
\bool_lazy_any:nTF	2313, 2322
\bool_lazy_or:nnTF	1038, 2223
\bool_new:N	251, 568, 569, 594, 618, 627, 634, 635, 690, 691, 708, 709, 825, 826, 1060, 1073, 1387, 1388, 1398, 1404, 1405
\bool_set:Nn	1032
\bool_set_false:N	581, 585, 642, 651, 652, 667, 847, 1209, 1444, 1487, 1501, 1515, 1718, 1855, 1856, 2320, 2337
\bool_set_true:N	306, 575, 576, 580, 586, 641, 646, 647, 836, 841, 1224, 1235, 1250, 1256, 1271, 1277, 1305, 1317, 1452, 1482, 1488, 1492, 1519, 1525, 2336, 2352, 2359, 2360, 2378, 2385, 2386
\bool_until_do:Nn	1210, 1445
C	
clist commands:	
\clist_map_inline:nn	477
\counterwithin	4
cs commands:	
\cs_generate_variant:Nn	55, 56, 302, 310, 943, 949, 1189, 2060
\cs_if_exist:NTF	39, 48, 69
\cs_new:Npn	37, 46, 57, 67, 78, 2011, 2061
\cs_new_protected:Npn	252, 303, 311, 317, 438, 938, 944, 1027, 1077, 1119, 1130, 1190, 1325, 1377, 1421, 1583, 1851, 2007, 2009, 2213, 2340, 2393, 2450
\cs_new_protected:Npx	90
\cs_set_eq:NN	94
E	
\endinput	12
exp commands:	
\exp_args:NNe	27
\exp_args:NNnx	242
\exp_args:NnV	280
\exp_args:NNx	95, 1246, 1267
\exp_args:Nnx	313
\exp_args:Nx	262
\exp_args:Nxx	1173, 1220, 1286, 2344, 2366, 2370
\exp_not:N	1761, 1764, 1786, 1789, 1792,

2021, 2024, 2027, 2040, 2042, 2045, 2048, 2053, 2055, 2058, 2064, 2073, 2094, 2097, 2099, 2102, 2109, 2116, 2118, 2122, 2125, 2128, 2130, 2136, 2140, 2143, 2156, 2159, 2162, 2189, 2191, 2194, 2197, 2204, 2206, 2209	\exp_not:n . 1605, 1621, 1633, 1638, 1661, 1675, 1679, 1691, 1695, 1729, 1730, 1762, 1785, 1790, 1791, 1920, 1933, 1940, 1964, 1976, 1980, 1990, 1994, 2022, 2023, 2025, 2035, 2038, 2041, 2046, 2047, 2049, 2050, 2052, 2054, 2095, 2096, 2098, 2100, 2101, 2103, 2104, 2108, 2120, 2121, 2126, 2127, 2129, 2137, 2141, 2142, 2144, 2157, 2158, 2160, 2183, 2187, 2190, 2195, 2196, 2198, 2199, 2203, 2205	\ExplSyntaxOn 12, 264
F		
file commands:		
\file_get:nnNTF	262	
\fmtversion	3	
G		
group commands:		
\group_begin:	93, 254, 305, 927, 1029, 1042, 1761, 1789, 2021, 2024, 2045, 2048, 2094, 2099, 2102, 2118, 2125, 2140, 2156, 2159, 2194, 2197	
\group_end:	96, 300, 308, 935, 1045, 1057, 1786, 1792, 2040, 2042, 2053, 2055, 2097, 2109, 2116, 2122, 2128, 2143, 2189, 2191, 2204, 2206	
I		
\IfBooleanTF	1063	
\IfFormatAtLeastTF	3, 4	
\input	12	
int commands:		
\int_case:nnTF	1586, 1614, 1646, 1819, 1913, 1952	
\int_compare:nNnTF	1177, 1225, 1290, 1306, 1333, 1335, 1379, 1547, 1601, 1635, 1808, 1810, 1868, 1893, 1937, 2348, 2354, 2374, 2380	
\int_compare_p:nNn	1349, 1357, 2227, 2238, 2333	
\int_eval:n	90	
\int_incr:N	1846, 1883, 1885, 1899, 1901, 1905, 1907, 2005	
\int_new:N	1074, 1075, 1389, 1390, 1401, 1402	
\int_set:Nn	1334, 1336, 1340, 1343	
\int_use:N	33, 35, 50	
\int_zero:N	1327, 1328, 1430, 1431, 1432, 1433, 1845, 1847, 1848, 2000, 2001	
iow commands:		
\iow_char:N	103, 109, 118, 119, 124, 125, 134, 135, 145	
K		
keys commands:		
\keys_define:nn	29, 323, 335, 352, 366, 445, 473, 499, 523, 551, 558, 570, 595, 604, 619, 628, 636, 669, 676, 692, 710, 746, 784, 817, 820, 827, 837, 848, 859, 870, 888, 900, 950, 962, 983, 1006	
\keys_set:nn	29, 33, 281, 842, 877, 883, 932, 1030	
keyval commands:		
\keyval_parse:nnn	449, 503	
L		
\labelformat	3	
\language name	24, 729	
M		
\mainbabelname	24, 736	
\MessageBreak	10	
msg commands:		
\msg_info:nnn	343, 373	
\msg_line_context:	102, 108, 112, 130, 133, 139, 153, 157, 159, 161, 164, 168	
\msg_new:nnn	100, 106, 111, 113, 115, 121, 127, 129, 131, 137, 142, 147, 149, 151, 156, 158, 160, 162, 167	
\msg_note:nnn	284	
\msg_warning:nn	537, 562, 666, 672, 830, 851	
\msg_warning:nnn	231, 246, 290, 298, 505, 892, 934, 974, 1013, 1540, 1713, 2266, 2301, 2442	
\msg_warning:nnnn	451, 1233, 1254, 1275, 1315	
N		
\newcounter	4	
\NewDocumentCommand	226, 236, 876, 878, 925, 1025, 1061	
\nobreakspace	387, 2487, 2488, 2490, 2491, 2493, 2495, 2677, 2678, 2680, 2681, 2683, 2685, 2855, 2856, 2858, 2859, 2861, 2863, 3028, 3029, 3031, 3032, 3034, 3036, 3212, 3213, 3215, 3216, 3218, 3220	

P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	6
<code>\pageref</code>	34
prg commands:	
<code>\prg_generate_conditional_-</code> variant:Nnn	411, 427
<code>\prg_new_protected_conditional:Npnn</code>	397, 413, 430
<code>\prg_return_false:</code>	407, 409, 423, 425, 436
<code>\prg_return_true:</code>	406, 422, 435
<code>\ProcessKeysOptions</code>	875
prop commands:	
<code>\prop_get:NnN</code>	2419
<code>\prop_get:NnNTF</code>	255, 400, 403, 416, 419, 433, 928, 2251, 2272, 2280, 2396, 2453, 2466
<code>\prop_gput:Nnn</code>	232, 243, 940, 946
<code>\prop_gput_if_new:Nnn</code>	313, 319
<code>\prop_gset_from_keyval:Nn</code>	381
<code>\prop_if_exist:NTF</code>	267, 880
<code>\prop_if_exist_p:N</code>	2403, 2459
<code>\prop_if_in:NnTF</code>	25, 230, 240
<code>\prop_if_in_p:Nn</code>	60, 2410
<code>\prop_item:Nn</code>	27, 61, 244
<code>\prop_new:N</code>	225, 273, 380, 444, 498, 855, 881
<code>\prop_put:Nnn</code>	442, 866, 915
<code>\prop_remove:Nn</code>	441, 865, 907
<code>\providecommand</code>	3
<code>\ProvidesExplPackage</code>	14
<code>\ProvidesFile</code>	12
R	
<code>\refstepcounter</code>	3
<code>\RequirePackage</code>	16, 17, 18, 19, 20, 662
S	
seq commands:	
<code>\seq_clear:N</code>	615, 1079
<code>\seq_const_from_clist:Nn</code>	171, 179, 192, 204
<code>\seq_gconcat:NNN</code>	212, 215, 219, 222
<code>\seq_get_left:NN</code>	1455
<code>\seq_gput_right:Nn</code>	282
<code>\seq_if_empty:NTF</code>	1449
<code>\seq_if_in:NnTF</code>	258, 479, 1123
<code>\seq_map_break:n</code>	81, 1368, 1371
<code>\seq_map_function:NN</code>	1082
<code>\seq_map_indexed_inline:Nn</code>	21, 1329
<code>\seq_map_inline:Nn</code>	332, 349, 363, 856, 885, 897, 959, 980, 1003, 1365
<code>\seq_map_tokens:Nn</code>	63
<code>\seq_new:N</code>	211, 218, 250, 472, 603, 1059, 1076, 1386
<code>\seq_pop_left:NN</code>	1447
<code>\seq_put_right:Nn</code>	481, 1126
<code>\seq_reverse:N</code>	609
<code>\seq_set_eq:NN</code>	1423
<code>\seq_set_from_clist:Nn</code>	608, 1031
<code>\seq_sort:Nn</code>	36, 1085
sort commands:	
<code>\sort_return_same:</code>	36, 41, 1092, 1097, 1144, 1182, 1184, 1230, 1236, 1251, 1257, 1278, 1311, 1318, 1353, 1368, 1384
<code>\sort_return_swapped:</code>	36, 41, 1105, 1153, 1181, 1229, 1272, 1310, 1361, 1371, 1383
str commands:	
<code>\str_case:nnTF</code>	752, 788
<code>\str_if_eq:nnTF</code>	80
<code>\str_if_eq_p:nn</code>	2318, 2324, 2326, 2330
<code>\str_new:N</code>	675
<code>\str_set:Nn</code>	680, 682, 684, 686
T	
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@Alph</code>	66
<code>\@addtoreset</code>	4
<code>\@chapapp</code>	66
<code>\@currentcounter</code>	4, 21, 25, 28, 30, 33, 84, 86
<code>\@currentlabel</code>	3
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	534, 549, 659, 727, 733, 834
<code>\@onlypreamble</code>	235, 249, 937
<code>\bbl@loaded</code>	24
<code>\bbl@main@language</code>	24, 730
<code>\c@</code>	3
<code>\c@page</code>	6, 94
<code>\cl@</code>	4
<code>\hyper@link</code>	56, 1764, 2027, 2073, 2162
<code>\p@</code>	3
<code>\zref@addprop</code>	22, 32, 34, 36, 87, 88, 99
<code>\zref@default</code>	56, 2008, 2010
<code>\zref@extractdefault</code>	1122, 1133, 1135, 1174, 1175, 1178, 1180, 1193, 1197, 1201, 1205, 1221, 1222, 1226, 1228, 1248, 1269, 1380, 1382, 1467, 1472, 1770, 1775, 1781, 2030, 2031, 2033, 2036, 2051, 2078, 2083, 2089, 2105, 2167, 2172, 2178, 2184, 2200, 2345, 2346, 2349, 2351, 2355, 2357, 2367, 2368, 2371, 2372, 2375, 2377, 2381, 2383

`__zrefclever_counter_reset_by_`
`auxi:nnn` 74, 78
`\l__zrefclever_counter_resetby_`
`prop` 4, 19, 60, 61, 498, 510
`\l__zrefclever_counter_resettters_`
`seq` 4, 18, 19, 63, 472, 479, 482
`\l__zrefclever_counter_type_prop`
..... 3, 18, 25, 28, 444, 456
`\l__zrefclever_current_language_`
`tl` .. 24, 724, 729, 735, 739, 765, 801
`__zrefclever_declare_default_`
`transl:nnn` 31, 938, 969, 990
`__zrefclever_declare_type_`
`transl:nnnn` 31, 938, 995, 1017
`\g__zrefclever_dict_⟨language⟩_prop`
..... 12
`\l__zrefclever_dict_language_tl` .
..... 169, 256, 260, 263, 270,
276, 283, 285, 291, 314, 320, 401,
404, 417, 420, 929, 970, 991, 996, 1018
`\g__zrefclever_fallback_dict_`
`prop` 9, 380, 381, 433
`__zrefclever_get_default_`
`transl:nnN` 9, 414, 428
`__zrefclever_get_default_`
`transl:nnNTF` 16, 413, 2434
`__zrefclever_get_enclosing_`
`counters:n` 5, 37, 42, 84
`__zrefclever_get_enclosing_`
`counters_value:n` ... 5, 37, 51, 86
`__zrefclever_get_fallback_`
`transl:nN` 431
`__zrefclever_get_fallback_`
`transl:nNTF` 17, 429, 2439
`__zrefclever_get_ref:n`
..... 56, 1606, 1622,
1634, 1639, 1662, 1676, 1680, 1692,
1696, 1731, 1751, 1921, 1934, 1941,
1965, 1977, 1981, 1991, 1995, 2011
`__zrefclever_get_ref_first:` ...
..... 56, 60, 1744, 1802, 2061
`__zrefclever_get_ref_font:nN` . 8,
15, 27, 64, 66, 1567, 1569, 1571, 2450
`__zrefclever_get_ref_string:nN` .
..... 8, 9, 15, 27, 64, 1046, 1436,
1438, 1440, 1549, 1551, 1553, 1555,
1557, 1559, 1561, 1563, 1565, 2393
`__zrefclever_get_type_transl:nnnN`
..... 9, 398, 412
`__zrefclever_get_type_transl:nnnNTF`
..... 16, 397, 2259, 2288, 2294, 2428
`\l__zrefclever_label_a_tl`
. 42, 1391, 1448, 1468, 1484, 1534,
1535, 1541, 1593, 1606, 1622, 1639,
1680, 1696, 1724, 1731, 1859, 1863,
1873, 1898, 1921, 1942, 1981, 1995
`\l__zrefclever_label_b_tl`
..... 42, 1391,
1451, 1456, 1473, 1486, 1491, 1863
`\l__zrefclever_label_count_int` ..
..... 43, 1389,
1430, 1547, 1586, 1845, 1868, 2005
`\l__zrefclever_label_enclcnt_a_`
`tl` 1067, 1192, 1194, 1195,
1216, 1243, 1268, 1287, 1295, 1296
`\l__zrefclever_label_enclcnt_b_`
`tl` 1067, 1196, 1198, 1199,
1217, 1247, 1264, 1288, 1297, 1298
`\l__zrefclever_label_enclval_a_`
`tl` 1067, 1200,
1202, 1203, 1291, 1299, 1300, 1307
`\l__zrefclever_label_enclval_b_`
`tl` 1067, 1204,
1206, 1207, 1293, 1301, 1302, 1309
`\l__zrefclever_label_type_a_tl` ..
..... 64, 1067, 1121, 1124,
1127, 1132, 1141, 1150, 1158, 1163,
1339, 1367, 1461, 1465, 1498, 1506,
1512, 1538, 1595, 1875, 2401, 2406,
2413, 2422, 2430, 2457, 2462, 2469
`\l__zrefclever_label_type_b_tl` ..
..... 1067,
1134, 1142, 1151, 1159, 1164, 1342,
1370, 1462, 1470, 1499, 1508, 1513
`__zrefclever_label_type_put_`
`new_right:n` 35, 36, 1083, 1119
`\l__zrefclever_label_types_seq` ..
..... 36, 1076, 1079, 1123, 1126, 1365
`__zrefclever_labels_in_sequence:nn`
..... 43, 63, 1722, 1862, 2340
`\g__zrefclever_languages_prop` ...
..... 11, 225, 230,
232, 240, 243, 244, 255, 400, 416, 928
`\l__zrefclever_last_of_type_bool`
..... 42, 1386, 1482, 1487, 1488,
1492, 1501, 1516, 1520, 1526, 1576
`\l__zrefclever_lastsep_tl` . 1406,
1558, 1621, 1638, 1661, 1679, 1691
`\l__zrefclever_link_star_bool` ...
..... 1032, 1059, 2018, 2153, 2316
`\l__zrefclever_listsep_tl`
... 1406, 1556, 1633, 1675, 1920,
1933, 1940, 1964, 1976, 1980, 1990
`\l__zrefclever_load_dict_`
`verbose_bool` ... 251, 288, 297, 306
`\g__zrefclever_loaded_dictionaries_`
`seq` 250, 259, 282

`\l_zrefclever_main_language_tl` .
. 24, 723, 730, 736, 740, 744, 757, 793
`_zrefclever_name_default:`
..... 2007, 2136
`\l_zrefclever_name_format_-`
`fallback_tl`
.. 1397, 2243, 2247, 2249, 2285, 2297
`\l_zrefclever_name_format_tl` ...
... 1397, 2229, 2230, 2233, 2234,
2244, 2245, 2256, 2262, 2277, 2291
`\l_zrefclever_name_in_link_bool`
..... 57,
60, 1397, 1759, 2066, 2320, 2336, 2337
`\l_zrefclever_namefont_tl` 1406,
1568, 1762, 1790, 2095, 2126, 2141
`\l_zrefclever_nameinlink_str` ...
..... 675, 680,
682, 684, 686, 2318, 2324, 2326, 2330
`\l_zrefclever_namesep_tl`
.. 1406, 1550, 2098, 2129, 2137, 2144
`\l_zrefclever_next_is_same_bool`
..... 43, 63, 1401,
1856, 1884, 1900, 1906, 2360, 2386
`\l_zrefclever_next_maybe_range_-`
`bool`
.. 43, 63, 1401, 1718, 1728, 1855,
1880, 1890, 2352, 2359, 2378, 2385
`\l_zrefclever_noabbrev_first_-`
`bool` 709, 718, 2240
`_zrefclever_page_format_aux:` ..
..... 90, 94
`\g_zrefclever_page_format_tl` ...
..... 6, 89, 95, 98
`\l_zrefclever_pairsep_tl`
..... 1406, 1554, 1605, 1729
`_zrefclever_prop_put_non_-`
`empty:Nnn` 17, 438, 455, 509
`_zrefclever_provide_dict_-`
`default_transl:nn` 14, 311, 341, 358
`_zrefclever_provide_dict_type_-`
`transl:nn` 14, 311, 359, 376
`_zrefclever_provide_dictionary:n`
.... 9, 12-14, 33, 252, 307, 783, 1033
`_zrefclever_provide_dictionary_-`
`verbose:n` 13,
303, 758, 766, 772, 794, 802, 808
`\l_zrefclever_range_beg_label_-`
`tl` 43, 1401, 1429,
1634, 1657, 1663, 1673, 1677, 1689,
1693, 1844, 1882, 1897, 1931, 1935,
1962, 1966, 1974, 1978, 1988, 1992
`\l_zrefclever_range_count_int` ..
..... 43,

1401, 1432, 1614, 1648, 1847, 1883,
1894, 1899, 1905, 1913, 1954, 2000
`\l_zrefclever_range_same_count_-`
`int` 43,
1401, 1433, 1601, 1636, 1649, 1848,
1885, 1901, 1907, 1938, 1955, 2001
`\l_zrefclever_rangesep_tl`
..... 1406, 1552, 1695, 1730, 1994
`_zrefclever_ref_default:`
..... 2007, 2058, 2064, 2130, 2209
`\l_zrefclever_ref_language_tl` ..
..... 24, 25,
722, 743, 756, 759, 764, 767, 771,
773, 783, 792, 795, 800, 803, 807,
809, 1033, 2260, 2289, 2295, 2429, 2435
`\c_zrefclever_ref_options_font_-`
`seq` 10, 15, 171
`\c_zrefclever_ref_options_-`
`necessarily_not_type_specific_-`
`seq` 15, 171, 333, 886, 960
`\c_zrefclever_ref_options_-`
`necessarily_type_specific_seq`
..... 171, 364, 1004
`\c_zrefclever_ref_options_-`
`possibly_type_specific_seq` ..
..... 15, 171, 350, 981
`\l_zrefclever_ref_options_prop` .
.... 27-29, 855, 865, 866, 2396, 2453
`\c_zrefclever_ref_options_-`
`reference_seq` 171, 857
`\c_zrefclever_ref_options_-`
`typesetup_seq` 171, 898
`\l_zrefclever_ref_property_tl` ..
..... 20,
522, 527, 529, 535, 538, 554, 563,
1080, 1112, 1459, 2013, 2037, 2051,
2070, 2107, 2148, 2186, 2202, 2342
`\l_zrefclever_ref_typeset_font_-`
`tl` 816, 818, 1043
`\l_zrefclever_reffont_in_tl` 1406,
1572, 2025, 2049, 2103, 2160, 2198
`\l_zrefclever_reffont_out_tl` ...
..... 1406, 1570,
2022, 2046, 2100, 2120, 2157, 2195
`\l_zrefclever_refpos_in_tl` 1406,
1566, 2038, 2052, 2108, 2187, 2203
`\l_zrefclever_refpos_out_tl` 1406,
1562, 2041, 2054, 2121, 2190, 2205
`\l_zrefclever_refpre_in_tl` 1406,
1564, 2035, 2050, 2104, 2183, 2199
`\l_zrefclever_refpre_out_tl` 1406,
1560, 2023, 2047, 2101, 2158, 2196
`\l_zrefclever_setup_type_tl` ...
..... 14, 169, 279, 315, 328,

329, 340, 357, 371, 882, 910, 918,
 931, 955, 956, 967, 988, 997, 1011, 1019
 \l_zrefclever_sort_decided_bool
 ... 1073, 1209, 1210, 1224, 1235,
 1250, 1256, 1271, 1277, 1305, 1317
 _zrefclever_sort_default:nn ...
 ... 36, 1114, 1130
 _zrefclever_sort_default_-
 different_types:nn ...
 ... 21, 35, 40, 1168, 1325
 _zrefclever_sort_default_same_-
 type:nn ... 34, 38, 1166, 1190
 _zrefclever_sort_labels: ...
 ... 35, 36, 41, 1041, 1077
 _zrefclever_sort_page:nn ...
 ... 41, 1113, 1377
 \l_zrefclever_sort_prior_a_int .
 ... 1074,
 1327, 1333, 1334, 1340, 1350, 1358
 \l_zrefclever_sort_prior_b_int .
 ... 1074,
 1328, 1335, 1336, 1343, 1351, 1359
 \l_zrefclever_tlastsep_tl
 ... 1406, 1441, 1833
 \l_zrefclever_tlistsep_tl
 ... 1406, 1439, 1811
 \l_zrefclever_tpairsep_tl
 ... 1406, 1437, 1827
 \l_zrefclever_type_<type>-
 options_prop 29
 \l_zrefclever_type_count_int ...
 ... 43, 61, 1389, 1431, 1808,
 1810, 1819, 1846, 2227, 2239, 2333
 \l_zrefclever_type_first_label_-
 tl 42, 57, 1391, 1427, 1592, 1710,
 1719, 1723, 1751, 1767, 1771, 1776,
 1782, 1842, 1872, 2063, 2069, 2076,
 2079, 2084, 2090, 2106, 2147, 2165,
 2168, 2173, 2179, 2185, 2201, 2215
 \l_zrefclever_type_first_label_-
 type_tl 43, 61, 1391, 1428, 1594,
 1714, 1843, 1874, 2218, 2254, 2261,
 2267, 2275, 2283, 2290, 2296, 2303
 _zrefclever_type_name_setup: ..
 ... 8, 9, 57, 1739, 2213
 \l_zrefclever_type_name_tl
 ... 57, 60,
 1397, 1785, 1791, 2096, 2127, 2134,
 2142, 2216, 2219, 2257, 2263, 2265,
 2278, 2286, 2292, 2298, 2300, 2317
 \l_zrefclever_typeset_compress_-
 bool 618, 621, 1857
 \l_zrefclever_typeset_labels_-
 seq 42, 1386, 1423, 1447, 1449, 1455
 \l_zrefclever_typeset_last_bool
 42, 1386,
 1444, 1445, 1452, 1481, 1816, 2332
 \l_zrefclever_typeset_name_bool
 569, 576, 581, 586, 1741, 1755
 \l_zrefclever_typeset_queue_-
 curr_tl 43,
 56, 61, 1391, 1426, 1603, 1619,
 1628, 1659, 1670, 1686, 1708,
 1726, 1743, 1750, 1757, 1801, 1823,
 1828, 1834, 1840, 1841, 1918, 1929,
 1960, 1972, 1986, 2232, 2327, 2331
 \l_zrefclever_typeset_queue_-
 prev_tl . 43, 1391, 1425, 1812, 1839
 \l_zrefclever_typeset_range_-
 bool 627, 630, 1040, 1706
 \l_zrefclever_typeset_ref_bool .
 568, 575, 580, 585, 1741, 1748
 _zrefclever_typeset_refs:
 42-44, 1044, 1421
 _zrefclever_typeset_refs_last_-
 of_type: . 48, 56, 57, 60, 1578, 1583
 _zrefclever_typeset_refs_not_-
 last_of_type:
 43, 48, 56, 63, 1580, 1851
 \l_zrefclever_typeset_sort_bool
 594, 597, 1039
 \l_zrefclever_typesort_seq
 21, 40, 603, 608, 609, 615, 1329
 \l_zrefclever_use_hyperref_bool
 634, 641,
 646, 651, 661, 667, 2017, 2152, 2315
 \l_zrefclever_warn_hyperref_-
 bool 635, 642, 647, 652, 665
 _zrefclever_zcref:nnn .. 1026, 1027
 _zrefclever_zcref:nnnn 33, 35, 1027
 \l_zrefclever_zcref_labels_seq .
 35,
 36, 1031, 1055, 1059, 1082, 1085, 1424
 \l_zrefclever_zcref_note_tl ...
 819, 822, 1048
 \l_zrefclever_zcref_with_check_-
 bool 826, 841, 1036, 1051
 \l_zrefclever_zrefcheck_-
 available_bool
 825, 836, 847, 1035, 1050