

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-29

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	67
9.2	enumitem package	67

*This file describes v0.1.0-alpha, released 2021-09-29.

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

10	Dictionaries	67
10.1	English	67
10.2	German	71
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} {2021-09-29} {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 counterresettters .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 .default:n = zc@thecnt ,
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 }

```

abbrev and noabbrevfirst options

```

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

```

C option

```

718 \keys_define:nn { zref-clever / reference }
719 {
720     C .meta:n =
721     { capfirst = true , noabbrevfirst = true },
722     C .value_forbidden:n = true ,
723 }

```

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`.

```

724 \tl_new:N \l__zrefclever_ref_language_tl
725 \tl_new:N \l__zrefclever_main_language_tl
726 \tl_new:N \l__zrefclever_current_language_tl
727 \AddToHook { begindocument }
728 {
729   \@ifpackageloaded { babel }
730   {
731     \tl_set:Nn \l__zrefclever_current_language_tl { \language }
732     \tl_set:Nn \l__zrefclever_main_language_tl { \bbl@main@language }
733   }
734   {
735     \@ifpackageloaded { polyglossia }
736     {
737       \tl_set:Nn \l__zrefclever_current_language_tl { \babelname }
738       \tl_set:Nn \l__zrefclever_main_language_tl { \mainbabelname }
739     }
740     {
741       \tl_set:Nn \l__zrefclever_current_language_tl { english }
742       \tl_set:Nn \l__zrefclever_main_language_tl { english }
743     }
744   }

```

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.

```

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

```

```

769         { \l__zrefclever_ref_language_tl }
770     }
771 }
772 {
773     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
774     \__zrefclever_provide_dictionary_verbos:x
775     { \l__zrefclever_ref_language_tl }
776 }
777 }
778 } ,
779 lang .value_required:n = true ,
780 }
781 \AddToHook { begindocument / before }
782 {
783     \AddToHook { begindocument }
784     {

```

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.

```

785     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

786     \keys_define:nn { zref-clever / reference }
787     {
788         lang .code:n =
789         {
790             \str_case:nnF {#1}
791             {
792                 { main }
793                 {
794                     \tl_set:Nn \l__zrefclever_ref_language_tl
795                     { \l__zrefclever_main_language_tl }
796                     \__zrefclever_provide_dictionary_verbos:x
797                     { \l__zrefclever_ref_language_tl }
798                 }
799
800                 { current }
801                 {
802                     \tl_set:Nn \l__zrefclever_ref_language_tl
803                     { \l__zrefclever_current_language_tl }
804                     \__zrefclever_provide_dictionary_verbos:x
805                     { \l__zrefclever_ref_language_tl }
806                 }
807             }
808         {
809             \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
810             \__zrefclever_provide_dictionary_verbos:x
811             { \l__zrefclever_ref_language_tl }
812         }
813     } ,
814     lang .value_required:n = true ,
815 }
816 }
817 }

```

font option

```
818 \tl_new:N \l__zrefclever_ref_typeset_font_tl
819 \keys_define:nn { zref-clever / reference }
820 { font .tl_set:N = \l__zrefclever_ref_typeset_font_tl }
```

note option

```
821 \tl_new:N \l__zrefclever_zcref_note_tl
822 \keys_define:nn { zref-clever / reference }
823 {
824     note .tl_set:N = \l__zrefclever_zcref_note_tl ,
825     note .value_required:n = true ,
826 }
```

check option

Integration with zref-check.

```
827 \bool_new:N \l__zrefclever_zrefcheck_available_bool
828 \bool_new:N \l__zrefclever_zcref_with_check_bool
829 \keys_define:nn { zref-clever / reference }
830 {
831     check .code:n =
832     { \msg_warning:nn { zref-clever } { check-document-only } } ,
833 }
834 \AddToHook { begindocument }
835 {
836     \@ifpackageloaded { zref-check }
837     {
838         \bool_set_true:N \l__zrefclever_zrefcheck_available_bool
839         \keys_define:nn { zref-clever / reference }
840         {
841             check .code:n =
842             {
843                 \bool_set_true:N \l__zrefclever_zcref_with_check_bool
844                 \keys_set:nn { zref-check / zcheck } {#1}
845             }
846         }
847     }
848     {
849         \bool_set_false:N \l__zrefclever_zrefcheck_available_bool
850         \keys_define:nn { zref-clever / reference }
851         {
852             check .code:n =
853             { \msg_warning:nn { zref-clever } { missing-zref-check } }
854         }
855     }
856 }
```

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.

```

857 \prop_new:N \l__zrefclever_ref_options_prop
858 \seq_map_inline:Nn
859   \c__zrefclever_ref_options_reference_seq
860   {
861     \keys_define:nn { zref-clever / reference }
862     {
863       #1 .default:V = \c_novalue_tl ,
864       #1 .code:n =
865       {
866         \tl_if_novalue:nTF {##1}
867         { \prop_remove:Nn \l__zrefclever_ref_options_prop {#1} }
868         { \prop_put:Nnn \l__zrefclever_ref_options_prop {#1} {##1} }
869       } ,
870     }
871   }

```

Package options

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

```

872 \keys_define:nn { }
873 {
874   zref-clever / zcsetup .inherit:n = zref-clever / label ,
875   zref-clever / zcsetup .inherit:n = zref-clever / reference ,
876 }

```

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

```

877 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

878 \NewDocumentCommand \zcsetup { m }
879 { \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}

880 \NewDocumentCommand \zcRefTypeSetup { m m }
881 {
882   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
883   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
884   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
885   \keys_set:nn { zref-clever / typesetup } {#2}
886 }
```

(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>.

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

899 \seq_map_inline:Nn
900   \c__zrefclever_ref_options_typesetup_seq
901   {
902     \keys_define:nn { zref-clever / typesetup }
903     {
904       #1 .default:V = \c_novaluel_tl ,
```

```

905     #1 .code:n =
906     {
907         \tl_if_novalue:nTF {##1}
908         {
909             \prop_remove:cn
910             {
911                 l__zrefclever_type_
912                 \l__zrefclever_setup_type_tl _options_prop
913             }
914             {#1}
915         }
916         {
917             \prop_put:cnn
918             {
919                 l__zrefclever_type_
920                 \l__zrefclever_setup_type_tl _options_prop
921             }
922             {#1} {##1}
923         }
924     } ,
925 }
926 }

```

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>}

927 \NewDocumentCommand \zcDeclareTranslations { m m }
928 {
929     \group_begin:
930     \prop_get:NnNTF \g__zrefclever_languages_prop {#1}
931     \l__zrefclever_dict_language_tl
932     {
933         \tl_clear:N \l__zrefclever_setup_type_tl
934         \keys_set:nn { zref-clever / translations } {#2}
935     }
936     { \msg_warning:nnn { zref-clever } { unknown-language-transl } {#1} }
937     \group_end:
938 }
939 \@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>}}

940 \cs_new_protected:Npn \_zrefclever_declare_type_transl:nnnn #1#2#3#4
941 {
942   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
943     { type- #2 - #3 } {#4}
944 }
945 \cs_generate_variant:Nn \_zrefclever_declare_type_transl:nnnn { VVnn }
946 \cs_new_protected:Npn \_zrefclever_declare_default_transl:nnn #1#2#3
947 {
948   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
949     { default- #2 } {#3}
950 }
951 \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.

```

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

```

```

984 {
985   \keys_define:nn { zref-clever / translations }
986   {
987     #1 .value_required:n = true ,
988     #1 .code:n =
989     {
990       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
991       {
992         \__zrefclever_declare_default_transl:Vnn
993         \l__zrefclever_dict_language_tl
994         {#1} {##1}
995       }
996       {
997         \__zrefclever_declare_type_transl:Vnn
998         \l__zrefclever_dict_language_tl
999         \l__zrefclever_setup_type_tl
1000         {#1} {##1}
1001       }
1002     } ,
1003   }
1004 }
1005 \seq_map_inline:Nn
1006 \c__zrefclever_ref_options_necessarily_type_specific_seq
1007 {
1008   \keys_define:nn { zref-clever / translations }
1009   {
1010     #1 .value_required:n = true ,
1011     #1 .code:n =
1012     {
1013       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1014       {
1015         \msg_warning:nnn { zref-clever }
1016         { option-only-type-specific } {#1}
1017       }
1018       {
1019         \__zrefclever_declare_type_transl:Vnn
1020         \l__zrefclever_dict_language_tl
1021         \l__zrefclever_setup_type_tl
1022         {#1} {##1}
1023       }
1024     } ,
1025   }
1026 }

```

6 User interface

6.1 \zcref

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

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

```

1027 \NewDocumentCommand \zcref { s O { } m }
1028 { \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}
```

```
1029 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1030 {
1031   \group_begin:
```

Set options.

```
1032   \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

```
1033   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1034   \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.

```
1035   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Integration with zref-check.

```
1036   \bool_lazy_and:nnT
1037   { \l__zrefclever_zrefcheck_available_bool }
1038   { \l__zrefclever_zcref_with_check_bool }
1039   { \zrefcheck_zcref_beg_label: }
```

Sort the labels.

```
1040   \bool_lazy_or:nnT
1041   { \l__zrefclever_typeset_sort_bool }
1042   { \l__zrefclever_typeset_range_bool }
1043   { \__zrefclever_sort_labels: }
```

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

```
1044   \group_begin:
1045   \l__zrefclever_ref_typeset_font_tl
1046   \__zrefclever_typeset_refs:
1047   \group_end:
```

Typeset note.

```
1048   \tl_if_empty:NF \l__zrefclever_zcref_note_tl
1049   {
1050     \__zrefclever_get_ref_string:nN { notesep } \l_tmpa_tl
1051     \l_tmpa_tl
1052     \l__zrefclever_zcref_note_tl
1053   }
```

Integration with zref-check.

```
1054   \bool_lazy_and:nnT
1055   { \l__zrefclever_zrefcheck_available_bool }
1056   { \l__zrefclever_zcref_with_check_bool }
1057   {
```

```

1058         \zrefcheck_zcref_end_label_maybe:
1059         \zrefcheck_zcref_run_checks_on_labels:n
1060         { \l__zrefclever_zcref_labels_seq }
1061     }
1062     \group_end:
1063 }

```

(End definition for `__zrefclever_zcref:nnnn`.)

```

\l__zrefclever_zcref_labels_seq
\l__zrefclever_link_star_bool

```

```

1064 \seq_new:N \l__zrefclever_zcref_labels_seq
1065 \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>}

```

```

1066 \NewDocumentCommand \zcpageref { s O { } m }
1067 {
1068     \IfBooleanTF {#1}
1069     { \zcref*[#2, ref = page] {#3} }
1070     { \zcref [ #2, ref = page] {#3} }
1071 }

```

(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
1072 \tl_new:N \l__zrefclever_label_type_a_tl
1073 \tl_new:N \l__zrefclever_label_type_b_tl
1074 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
1075 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
1076 \tl_new:N \l__zrefclever_label_enclval_a_tl
1077 \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 `__zrefclever_sort_default_same_type:nn`, signals if the sorting between two labels has been decided or not.

```
1078 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for `\l_zrefclever_sort_decided_bool`.)

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

`\l_zrefclever_sort_prior_b_int`

```
1079 \int_new:N \l__zrefclever_sort_prior_a_int
```

```
1080 \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`.

```
1081 \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.

```
1082 \cs_new_protected:Npn \__zrefclever_sort_labels:
```

```
1083 {
```

Store label types sequence.

```
1084   \seq_clear:N \l__zrefclever_label_types_seq
```

```
1085   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
```

```
1086   {
```

```
1087     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
```

```
1088     \__zrefclever_label_type_put_new_right:n
```

```
1089   }
```

Sort.

```
1090   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
```

```
1091   {
```

```
1092     \zref@ifrefundefined {##1}
```

```
1093     {
```

```
1094       \zref@ifrefundefined {##2}
```

```
1095       {
```

```
1096         % Neither label is defined.
```

```
1097         \sort_return_same:
```

```
1098       }
```

```
1099     {
```

```
1100       % The second label is defined, but the first isn't, leave the
```

```
1101       % undefined first (to be more visible).
```

```
1102       \sort_return_same:
```

```
1103     }
```

```
1104   }
```

```

1105         {
1106             \zref@ifrefundefined {##2}
1107             {
1108                 % The first label is defined, but the second isn't, bring the
1109                 % second forward.
1110                 \sort_return_swapped:
1111             }
1112             {
1113                 % The interesting case: both labels are defined. References
1114                 % to the "default" property or to the "page" are quite
1115                 % different with regard to sorting, so we branch them here to
1116                 % specialized functions.
1117                 \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1118                 { \__zrefclever_sort_page:nn {##1} {##2} }
1119                 { \__zrefclever_sort_default:nn {##1} {##2} }
1120             }
1121         }
1122     }
1123 }

```

(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}

1124 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1125 {
1126     \tl_set:Nx \l__zrefclever_label_type_a_tl
1127     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1128     \seq_if_in:NVF \l__zrefclever_label_types_seq
1129     \l__zrefclever_label_type_a_tl
1130     {
1131         \seq_put_right:NV \l__zrefclever_label_types_seq
1132         \l__zrefclever_label_type_a_tl
1133     }
1134 }

```

(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}

```

```

1135 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1136 {
1137   \tl_set:Nx \l__zrefclever_label_type_a_tl
1138     { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1139   \tl_set:Nx \l__zrefclever_label_type_b_tl
1140     { \zref@extractdefault {#2} {zc@type} { \c_empty_tl } }
1141
1142   \bool_if:nTF
1143     {
1144       % The second label has a type, but the first doesn't, leave the
1145       % undefined first (to be more visible).
1146       \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1147       ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1148     }
1149     { \sort_return_same: }
1150     {
1151       \bool_if:nTF
1152         {
1153           % The first label has a type, but the second doesn't, bring the
1154           % second forward.
1155           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1156           \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1157         }
1158         { \sort_return_swapped: }
1159         {
1160           \bool_if:nTF
1161             {
1162               % The interesting case: both labels have a type...
1163               ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1164               ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1165             }
1166             {
1167               \tl_if_eq:NNTF
1168                 \l__zrefclever_label_type_a_tl
1169                 \l__zrefclever_label_type_b_tl
1170               % ...and it's the same type.
1171               { \__zrefclever_sort_default_same_type:nn {#1} {#2} }
1172               % ...and they are different types.
1173               { \__zrefclever_sort_default_different_types:nn {#1} {#2} }
1174             }
1175             {
1176               % Neither label has a type. We can't do much of meaningful
1177               % here, but if it's the same counter, compare it.
1178               \exp_args:Nxx \tl_if_eq:nnTF
1179                 { \zref@extractdefault {#1} { counter } { } }
1180                 { \zref@extractdefault {#2} { counter } { } }
1181                 {
1182                   \int_compare:nNnTF
1183                     { \zref@extractdefault {#1} {zc@cntval} { -1 } }
1184                     >
1185                     { \zref@extractdefault {#2} {zc@cntval} { -1 } }
1186                     { \sort_return_swapped: }
1187                     { \sort_return_same: }
1188                 }

```

```

1189         { \sort_return_same: }
1190     }
1191 }
1192 }
1193 }

```

(End definition for _zrefclever_sort_default:nn.)

Variant not provided by the kernel, for use in _zrefclever_sort_default_same_type:nn.

```

1194 \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>}
1195 \cs_new_protected:Npn \_zrefclever_sort_default_same_type:nn #1#2
1196 {
1197     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1198     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1199     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1200     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }
1201     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1202     { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1203     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1204     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1205     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1206     { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1207     \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1208     { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1209     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1210     { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1211     \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1212     { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1213
1214     \bool_set_false:N \l__zrefclever_sort_decided_bool
1215     \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1216     {
1217         \bool_if:nTF
1218         {
1219             % Both are empty: neither label has any (further) "enclosing
1220             % counters" (left).
1221             \tl_if_empty_p:V \l__zrefclever_label_enclcnt_a_tl &&
1222             \tl_if_empty_p:V \l__zrefclever_label_enclcnt_b_tl
1223         }
1224         {
1225             \exp_args:Nxx \tl_if_eq:nnTF
1226             { \zref@extractdefault {#1} { counter } { } }
1227             { \zref@extractdefault {#2} { counter } { } }
1228             {
1229                 \bool_set_true:N \l__zrefclever_sort_decided_bool
1230                 \int_compare:nNnTF
1231                 { \zref@extractdefault {#1} { zc@cntval } { -1 } }
1232                 >
1233                 { \zref@extractdefault {#2} { zc@cntval } { -1 } }
1234                 { \sort_return_swapped: }
1235                 { \sort_return_same: }
1236             }
1237         }
1238     }

```

```

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

```

```

1291 \exp_args:Nxx \tl_if_eq:nnTF
1292 { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1293 { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1294 {
1295   \int_compare:nNnTF
1296     { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1297     =
1298     { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1299     {
1300       \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1301         { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1302       \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1303         { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1304       \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1305         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1306       \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1307         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1308     }
1309     {
1310       \bool_set_true:N \l__zrefclever_sort_decided_bool
1311       \int_compare:nNnTF
1312         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1313         >
1314         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1315         { \sort_return_swapped: }
1316         { \sort_return_same: }
1317     }
1318   }
1319   {
1320     \msg_warning:nnnn { zref-clever }
1321     { counters-not-nested } {#1} {#2}
1322     \bool_set_true:N \l__zrefclever_sort_decided_bool
1323     \sort_return_same:
1324   }
1325 }
1326 }
1327 }
1328 }
1329 }

```

(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>}
1330 \cs_new_protected:Npn \__zrefclever_sort_default_different_types:nn #1#2
1331 {

```

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

```

1332 \int_zero:N \l__zrefclever_sort_prior_a_int
1333 \int_zero:N \l__zrefclever_sort_prior_b_int
1334 \seq_map_indexed_inline:Nn \l__zrefclever_typesort_seq
1335 {
1336   \tl_if_eq:nnTF {##2} {othertypes}}

```



```

1337     {
1338         \int_compare:nNt { \l__zrefclever_sort_prior_a_int } = { 0 }
1339         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1340         \int_compare:nNt { \l__zrefclever_sort_prior_b_int } = { 0 }
1341         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1342     }
1343     {
1344         \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##2}
1345         { \int_set:Nn \l__zrefclever_sort_prior_a_int { - ##1 } }
1346         {
1347             \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##2}
1348             { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1349         }
1350     }
1351 }

```

Then do the actual sorting.

```

1352     \bool_if:nTF
1353     {
1354         \int_compare_p:nNn
1355         { \l__zrefclever_sort_prior_a_int } <
1356         { \l__zrefclever_sort_prior_b_int }
1357     }
1358     { \sort_return_same: }
1359     {
1360         \bool_if:nTF
1361         {
1362             \int_compare_p:nNn
1363             { \l__zrefclever_sort_prior_a_int } >
1364             { \l__zrefclever_sort_prior_b_int }
1365         }
1366         { \sort_return_swapped: }
1367         {
1368             % Sort priorities are equal: the type that occurs first in
1369             % ‘labels’, as given by the user, is kept (or brought) forward.
1370             \seq_map_inline:Nn \l__zrefclever_label_types_seq
1371             {
1372                 \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1373                 { \seq_map_break:n { \sort_return_same: } }
1374                 {
1375                     \tl_if_eq:NnTF \l__zrefclever_label_type_b_tl {##1}
1376                     { \seq_map_break:n { \sort_return_swapped: } }
1377                 }
1378             }
1379         }
1380     }
1381 }

```

(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}}
1382 \cs_new_protected:Npn \_zrefclever_sort_page:nn #1#2
1383 {
1384   \int_compare:nNnTF
1385     { \zref@extractdefault {#1} { abspage } {-1} }
1386     >
1387     { \zref@extractdefault {#2} { abspage } {-1} }
1388     { \sort_return_swapped: }
1389     { \sort_return_same: }
1390 }

```

(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

<code>\l__zrefclever_typeset_labels_seq</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main stack control.
<code>\l__zrefclever_typeset_last_bool</code>	1391 <code>\seq_new:N \l__zrefclever_typeset_labels_seq</code>
<code>\l__zrefclever_last_of_type_bool</code>	1392 <code>\bool_new:N \l__zrefclever_typeset_last_bool</code>
	1393 <code>\bool_new:N \l__zrefclever_last_of_type_bool</code>
	(End definition for <code>\l__zrefclever_typeset_labels_seq</code> , <code>\l__zrefclever_typeset_last_bool</code> , and <code>\l__zrefclever_last_of_type_bool</code> .)
<code>\l__zrefclever_type_count_int</code>	Auxiliary variables for <code>__zrefclever_typeset_refs</code> : main counters.
<code>\l__zrefclever_label_count_int</code>	1394 <code>\int_new:N \l__zrefclever_type_count_int</code>
	1395 <code>\int_new:N \l__zrefclever_label_count_int</code>
	(End definition for <code>\l__zrefclever_type_count_int</code> and <code>\l__zrefclever_label_count_int</code> .)

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

```

\l__zrefclever_typeset_queue_prev_tl 1396 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_typeset_queue_curr_tl 1397 \tl_new:N \l__zrefclever_label_b_tl
\l__zrefclever_type_first_label_tl 1398 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
\l__zrefclever_type_first_label_type_tl 1399 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1400 \tl_new:N \l__zrefclever_type_first_label_tl
1401 \tl_new:N \l__zrefclever_type_first_label_type_tl

```

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

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

```

\l__zrefclever_name_in_link_bool 1402 \tl_new:N \l__zrefclever_type_name_tl
\l__zrefclever_name_format_tl 1403 \bool_new:N \l__zrefclever_name_in_link_bool
\l__zrefclever_name_format_fallback_tl 1404 \tl_new:N \l__zrefclever_name_format_tl
1405 \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_same_count_int 1406 \int_new:N \l__zrefclever_range_count_int
\l__zrefclever_range_beg_label_tl 1407 \int_new:N \l__zrefclever_range_same_count_int
\l__zrefclever_next_maybe_range_bool 1408 \tl_new:N \l__zrefclever_range_beg_label_tl
\l__zrefclever_next_is_same_bool 1409 \bool_new:N \l__zrefclever_next_maybe_range_bool
1410 \bool_new:N \l__zrefclever_next_is_same_bool

```

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

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

```

\l__zrefclever_tlastsep_tl 1411 \tl_new:N \l__zrefclever_tpairsep_tl
\l__zrefclever_namesep_tl 1412 \tl_new:N \l__zrefclever_tlistsep_tl
\l__zrefclever_pairsep_tl 1413 \tl_new:N \l__zrefclever_tlastsep_tl
\l__zrefclever_listsep_tl 1414 \tl_new:N \l__zrefclever_namesep_tl
\l__zrefclever_lastsep_tl 1415 \tl_new:N \l__zrefclever_pairsep_tl
\l__zrefclever_rangesep_tl 1416 \tl_new:N \l__zrefclever_listsep_tl
\l__zrefclever_refpre_out_tl 1417 \tl_new:N \l__zrefclever_lastsep_tl
\l__zrefclever_refpos_out_tl 1418 \tl_new:N \l__zrefclever_rangesep_tl
\l__zrefclever_refpre_in_tl 1419 \tl_new:N \l__zrefclever_refpre_out_tl
\l__zrefclever_refpos_in_tl 1420 \tl_new:N \l__zrefclever_refpos_out_tl
\l__zrefclever_namefont_tl 1421 \tl_new:N \l__zrefclever_refpre_in_tl
\l__zrefclever_reffont_out_tl 1422 \tl_new:N \l__zrefclever_refpos_in_tl
\l__zrefclever_reffont_in_tl 1423 \tl_new:N \l__zrefclever_namefont_tl
1424 \tl_new:N \l__zrefclever_reffont_out_tl
1425 \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 `\zcref`.

```

1426 \cs_new_protected:Npn \_zrefclever_typeset_refs:
1427 {
1428   \seq_set_eq:NN \l__zrefclever_typeset_labels_seq
1429   \l__zrefclever_zcref_labels_seq
1430   \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1431   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1432   \tl_clear:N \l__zrefclever_type_first_label_tl
1433   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1434   \tl_clear:N \l__zrefclever_range_beg_label_tl
1435   \int_zero:N \l__zrefclever_label_count_int
1436   \int_zero:N \l__zrefclever_type_count_int
1437   \int_zero:N \l__zrefclever_range_count_int
1438   \int_zero:N \l__zrefclever_range_same_count_int
1439
1440   % Get type block options (not type-specific).
1441   \_zrefclever_get_ref_string:nN { tpairsep }
1442   \l__zrefclever_tpairsep_tl
1443   \_zrefclever_get_ref_string:nN { tlistsep }
1444   \l__zrefclever_tlistsep_tl
1445   \_zrefclever_get_ref_string:nN { tlastsep }
1446   \l__zrefclever_tlastsep_tl
1447
1448   % Process label stack.
1449   \bool_set_false:N \l__zrefclever_typeset_last_bool
1450   \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1451   {
1452     \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1453     \l__zrefclever_label_a_tl
1454     \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1455     {
1456       \tl_clear:N \l__zrefclever_label_b_tl
1457       \bool_set_true:N \l__zrefclever_typeset_last_bool
1458     }
1459     {
1460       \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1461       \l__zrefclever_label_b_tl
1462     }
1463
1464     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1465     {
1466       \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1467       \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1468     }
1469     {
1470       \tl_set:Nx \l__zrefclever_label_type_a_tl
1471       {
1472         \zref@extractdefault
1473         { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1474       }
1475       \tl_set:Nx \l__zrefclever_label_type_b_tl
1476       {

```

```

1477         \zref@extractdefault
1478         { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1479     }
1480 }
1481
1482 % First, we establish whether the "current label" (i.e. 'a') is the
1483 % last one of its type. This can happen because the "next label"
1484 % (i.e. 'b') is of a different type (or different definition status),
1485 % or because we are at the end of the list.
1486 \bool_if:NTF \l__zrefclever_typeset_last_bool
1487 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1488 {
1489     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1490     {
1491         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1492         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1493         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1494     }
1495     {
1496         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1497         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1498         {
1499             % Neither is undefined, we must check the types.
1500             \bool_if:nTF
1501             {
1502                 % Both empty: same "type".
1503                 \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1504                 \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1505             }
1506             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1507             {
1508                 \bool_if:nTF
1509                 {
1510                     % Neither empty: compare types.
1511                     ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1512                     &&
1513                     ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1514                 }
1515                 {
1516                     \tl_if_eq:NNTF
1517                     \l__zrefclever_label_type_a_tl
1518                     \l__zrefclever_label_type_b_tl
1519                     {
1520                         \bool_set_false:N
1521                         \l__zrefclever_last_of_type_bool
1522                     }
1523                     {
1524                         \bool_set_true:N
1525                         \l__zrefclever_last_of_type_bool
1526                     }
1527                 }
1528                 % One empty, the other not: different "types".
1529                 {
1530                     \bool_set_true:N

```

```

1531         \l__zrefclever_last_of_type_bool
1532     }
1533 }
1534 }
1535 }
1536 }
1537
1538 % Handle warnings in case of reference or type undefined.
1539 \zref@refused { \l__zrefclever_label_a_tl }
1540 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1541 {}
1542 {
1543     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1544     {
1545         \msg_warning:nxx { zref-clever } { missing-type }
1546         { \l__zrefclever_label_a_tl }
1547     }
1548 }
1549
1550 % Get type-specific separators, refpre/pos and font options, once per
1551 % type.
1552 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1553 {
1554     \__zrefclever_get_ref_string:nN { namesep      }
1555     \l__zrefclever_namesep_tl
1556     \__zrefclever_get_ref_string:nN { rangesep     }
1557     \l__zrefclever_rangesep_tl
1558     \__zrefclever_get_ref_string:nN { pairsep      }
1559     \l__zrefclever_pairsep_tl
1560     \__zrefclever_get_ref_string:nN { listsep      }
1561     \l__zrefclever_listsep_tl
1562     \__zrefclever_get_ref_string:nN { lastsep      }
1563     \l__zrefclever_lastsep_tl
1564     \__zrefclever_get_ref_string:nN { refpre       }
1565     \l__zrefclever_refpre_out_tl
1566     \__zrefclever_get_ref_string:nN { refpos       }
1567     \l__zrefclever_refpos_out_tl
1568     \__zrefclever_get_ref_string:nN { refpre-in    }
1569     \l__zrefclever_refpre_in_tl
1570     \__zrefclever_get_ref_string:nN { refpos-in    }
1571     \l__zrefclever_refpos_in_tl
1572     \__zrefclever_get_ref_font:nN   { namefont    }
1573     \l__zrefclever_namefont_tl
1574     \__zrefclever_get_ref_font:nN   { reffont     }
1575     \l__zrefclever_reffont_out_tl
1576     \__zrefclever_get_ref_font:nN   { reffont-in  }
1577     \l__zrefclever_reffont_in_tl
1578 }
1579
1580 % Here we send this to a couple of auxiliary functions.
1581 \bool_if:NTF \l__zrefclever_last_of_type_bool
1582 % There exists no next label of the same type as the current.
1583 { \__zrefclever_typeset_refs_last_of_type: }
1584 % There exists a next label of the same type as the current.

```

```

1585         { \_zrefclever_typeset_refs_not_last_of_type: }
1586     }
1587 }

```

(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.

```

1588 \cs_new_protected:Npn \_zrefclever_typeset_refs_last_of_type:
1589 {
1590     % Process the current label to the current queue.
1591     \int_case:nnF { \l_zrefclever_label_count_int }
1592     {
1593         % It is the last label of its type, but also the first one, and that's
1594         % what matters here: just store it.
1595         { 0 }
1596         {
1597             \tl_set:NV \l_zrefclever_type_first_label_tl
1598             \l_zrefclever_label_a_tl
1599             \tl_set:NV \l_zrefclever_type_first_label_type_tl
1600             \l_zrefclever_label_type_a_tl
1601         }
1602
1603         % The last is the second: we have a pair (if not repeated).
1604         { 1 }
1605         {
1606             \int_compare:nNnF { \l_zrefclever_range_same_count_int } = { 1 }
1607             {
1608                 \tl_put_right:Nx \l_zrefclever_typeset_queue_curr_tl
1609                 {
1610                     \exp_not:V \l_zrefclever_pairsep_tl
1611                     \_zrefclever_get_ref:V \l_zrefclever_label_a_tl
1612                 }
1613             }
1614         }
1615     }
1616     % Last is third or more of its type: without repetition, we'd have the
1617     % last element on a list, but control for possible repetition.
1618     {
1619         \int_case:nnF { \l_zrefclever_range_count_int }
1620         {
1621             % There was no range going on.
1622             { 0 }
1623             {
1624                 \tl_put_right:Nx \l_zrefclever_typeset_queue_curr_tl
1625                 {

```



```

1626         \exp_not:V \l__zrefclever_lastsep_tl
1627         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1628     }
1629 }
1630 % Last in the range is also the second in it.
1631 { 1 }
1632 {
1633     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1634     {
1635         % We know 'range_beg_label' is not empty, since this is the
1636         % second element in the range, but the third or more in the
1637         % type list.
1638         \exp_not:V \l__zrefclever_listsep_tl
1639         \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1640         \int_compare:nNnF
1641             { \l__zrefclever_range_same_count_int } = { 1 }
1642         {
1643             \exp_not:V \l__zrefclever_lastsep_tl
1644             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1645         }
1646     }
1647 }
1648 }
1649 % Last in the range is third or more in it.
1650 {
1651     \int_case:nnF
1652     {
1653         \l__zrefclever_range_count_int -
1654         \l__zrefclever_range_same_count_int
1655     }
1656     {
1657         % Repetition, not a range.
1658         { 0 }
1659     {
1660         % If 'range_beg_label' is empty, it means it was also the
1661         % first of the type, and hence was already handled.
1662         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1663         {
1664             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1665             {
1666                 \exp_not:V \l__zrefclever_lastsep_tl
1667                 \__zrefclever_get_ref:V
1668                 \l__zrefclever_range_beg_label_tl
1669             }
1670         }
1671     }
1672     % A 'range', but with no skipped value, treat as list.
1673     { 1 }
1674     {
1675         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1676         {
1677             % Ditto.
1678             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1679             {

```

```

1680         \exp_not:V \l__zrefclever_listsep_tl
1681         \__zrefclever_get_ref:V
1682         \l__zrefclever_range_beg_label_tl
1683     }
1684     \exp_not:V \l__zrefclever_lastsep_tl
1685     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1686 }
1687 }
1688 }
1689 {
1690     % An actual range.
1691     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1692     {
1693         % Ditto.
1694         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1695         {
1696             \exp_not:V \l__zrefclever_lastsep_tl
1697             \__zrefclever_get_ref:V
1698             \l__zrefclever_range_beg_label_tl
1699         }
1700         \exp_not:V \l__zrefclever_rangesep_tl
1701         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1702     }
1703 }
1704 }
1705 }
1706
1707 % Handle "range" option. The idea is simple: if the queue is not empty,
1708 % we replace it with the end of the range (or pair). We can still
1709 % retrieve the end of the range from 'label_a' since we know to be
1710 % processing the last label of its type at this point.
1711 \bool_if:NT \l__zrefclever_typeset_range_bool
1712 {
1713     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1714     {
1715         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1716         { }
1717         {
1718             \msg_warning:nxx { zref-clever } { single-element-range }
1719             { \l__zrefclever_type_first_label_type_tl }
1720         }
1721     }
1722     {
1723         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1724         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1725         { }
1726         {
1727             \__zrefclever_labels_in_sequence:nn
1728             { \l__zrefclever_type_first_label_tl }
1729             { \l__zrefclever_label_a_tl }
1730         }
1731         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1732         {
1733             \bool_if:NTF \l__zrefclever_next_maybe_range_bool

```

```

1734         { \exp_not:V \l__zrefclever_pairsep_tl }
1735         { \exp_not:V \l__zrefclever_rangesep_tl }
1736         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1737     }
1738 }
1739 }
1740
1741 % Now that the type block is finished, we can add the name and the first
1742 % ref to the queue. Also, if "typeset" option is not "both", handle it
1743 % here as well.
1744 \__zrefclever_type_name_setup:
1745 \bool_if:nTF
1746 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1747 {
1748     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1749     { \__zrefclever_get_ref_first: }
1750 }
1751 {
1752     \bool_if:nTF
1753     { \l__zrefclever_typeset_ref_bool }
1754     {
1755         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1756         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1757     }
1758     {
1759         \bool_if:nTF
1760         { \l__zrefclever_typeset_name_bool }
1761         {
1762             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1763             {
1764                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1765                 {
1766                     \exp_not:N \group_begin:
1767                     \exp_not:V \l__zrefclever_namefont_tl
1768                     % It's two '@s', but escaped for DocStrip.
1769                     \exp_not:N \hyper@@link
1770                     {
1771                         \zref@ifrefcontainsprop
1772                         { \l__zrefclever_type_first_label_tl }
1773                         { urluse }
1774                         {
1775                             \zref@extractdefault
1776                             { \l__zrefclever_type_first_label_tl }
1777                             { urluse } {}
1778                         }
1779                         {
1780                             \zref@extractdefault
1781                             { \l__zrefclever_type_first_label_tl }
1782                             { url } {}
1783                         }
1784                     }
1785                     {
1786                         \zref@extractdefault
1787                         { \l__zrefclever_type_first_label_tl }

```

```

1788         { anchor } {}
1789     }
1790     { \exp_not:V \l__zrefclever_type_name_tl }
1791     \exp_not:N \group_end:
1792 }
1793 {
1794     \exp_not:N \group_begin:
1795     \exp_not:V \l__zrefclever_namefont_tl
1796     \exp_not:V \l__zrefclever_type_name_tl
1797     \exp_not:N \group_end:
1798 }
1799 }
1800 }
1801 {
1802     % Logically, this case would correspond to "typeset=none", but
1803     % it should not occur, given that the options are set up to
1804     % typeset either "ref" or "name". Still, leave here a
1805     % sensible fallback, equal to the behavior of "both".
1806     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1807         { \__zrefclever_get_ref_first: }
1808 }
1809 }
1810 }
1811
1812 % Typeset the previous type, if there is one.
1813 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1814 {
1815     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1816     { \l__zrefclever_tlistsep_tl }
1817     \l__zrefclever_typeset_queue_prev_tl
1818 }
1819
1820 % Wrap up loop, or prepare for next iteration.
1821 \bool_if:NTF \l__zrefclever_typeset_last_bool
1822 {
1823     % We are finishing, typeset the current queue.
1824     \int_case:nnF { \l__zrefclever_type_count_int }
1825     {
1826         % Single type.
1827         { 0 }
1828         { \l__zrefclever_typeset_queue_curr_tl }
1829         % Pair of types.
1830         { 1 }
1831         {
1832             \l__zrefclever_tpairsep_tl
1833             \l__zrefclever_typeset_queue_curr_tl
1834         }
1835     }
1836     {
1837         % Last in list of types.
1838         \l__zrefclever_tlastsep_tl
1839         \l__zrefclever_typeset_queue_curr_tl
1840     }
1841 }

```

```

1842 {
1843   % There are further labels, set variables for next iteration.
1844   \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1845   \l__zrefclever_typeset_queue_curr_tl
1846   \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1847   \tl_clear:N \l__zrefclever_type_first_label_tl
1848   \tl_clear:N \l__zrefclever_type_first_label_type_tl
1849   \tl_clear:N \l__zrefclever_range_beg_label_tl
1850   \int_zero:N \l__zrefclever_label_count_int
1851   \int_incr:N \l__zrefclever_type_count_int
1852   \int_zero:N \l__zrefclever_range_count_int
1853   \int_zero:N \l__zrefclever_range_same_count_int
1854 }
1855 }

```

(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.

```

1856 \cs_new_protected:Npn \__zrefclever_typeset_refs_not_last_of_type:
1857 {
1858   % Signal if next label may form a range with the current one (only
1859   % considered if compression is enabled in the first place).
1860   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1861   \bool_set_false:N \l__zrefclever_next_is_same_bool
1862   \bool_if:NT \l__zrefclever_typeset_compress_bool
1863   {
1864     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1865     { }
1866     {
1867       \__zrefclever_labels_in_sequence:nn
1868       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1869     }
1870   }
1871
1872   % Process the current label to the current queue.
1873   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1874   {
1875     % Current label is the first of its type (also not the last, but it
1876     % doesn't matter here): just store the label.
1877     \tl_set:NV \l__zrefclever_type_first_label_tl
1878     \l__zrefclever_label_a_tl
1879     \tl_set:NV \l__zrefclever_type_first_label_type_tl
1880     \l__zrefclever_label_type_a_tl
1881
1882     % If the next label may be part of a range, we set 'range_beg_label'
1883     % to "empty" (we deal with it as the "first", and must do it there, to
1884     % handle hyperlinking), but also step the range counters.
1885     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1886     {
1887       \tl_clear:N \l__zrefclever_range_beg_label_tl
1888       \int_incr:N \l__zrefclever_range_count_int
1889       \bool_if:NT \l__zrefclever_next_is_same_bool
1890       { \int_incr:N \l__zrefclever_range_same_count_int }
1891     }
1892   }

```

```

1892 }
1893 {
1894 % Current label is neither the first (nor the last) of its type.
1895 \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1896 {
1897 % Starting, or continuing a range.
1898 \int_compare:nNnTF
1899 { \l__zrefclever_range_count_int } = { 0 }
1900 {
1901 % There was no range going, we are starting one.
1902 \tl_set:NV \l__zrefclever_range_beg_label_tl
1903 \l__zrefclever_label_a_tl
1904 \int_incr:N \l__zrefclever_range_count_int
1905 \bool_if:NT \l__zrefclever_next_is_same_bool
1906 { \int_incr:N \l__zrefclever_range_same_count_int }
1907 }
1908 {
1909 % Second or more in the range, but not the last.
1910 \int_incr:N \l__zrefclever_range_count_int
1911 \bool_if:NT \l__zrefclever_next_is_same_bool
1912 { \int_incr:N \l__zrefclever_range_same_count_int }
1913 }
1914 }
1915 {
1916 % Next element is not in sequence: there was no range, or we are
1917 % closing one.
1918 \int_case:nnF { \l__zrefclever_range_count_int }
1919 {
1920 % There was no range going on.
1921 { 0 }
1922 {
1923 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1924 {
1925 \exp_not:V \l__zrefclever_listsep_tl
1926 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1927 }
1928 }
1929 % Last is second in the range: if 'range_same_count' is also
1930 % '1', it's a repetition (drop it), otherwise, it's a "pair
1931 % within a list", treat as list.
1932 { 1 }
1933 {
1934 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1935 {
1936 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1937 {
1938 \exp_not:V \l__zrefclever_listsep_tl
1939 \__zrefclever_get_ref:V
1940 \l__zrefclever_range_beg_label_tl
1941 }
1942 \int_compare:nNnF
1943 { \l__zrefclever_range_same_count_int } = { 1 }
1944 {
1945 \exp_not:V \l__zrefclever_listsep_tl

```

```

1946         \_zrefclever_get_ref:V
1947         \l__zrefclever_label_a_tl
1948     }
1949 }
1950 }
1951 }
1952 {
1953 % Last is third or more in the range: if 'range_count' and
1954 % 'range_same_count' are the same, its a repetition (drop it),
1955 % if they differ by '1', its a list, if they differ by more,
1956 % it is a real range.
1957 \int_case:nnF
1958 {
1959     \l__zrefclever_range_count_int -
1960     \l__zrefclever_range_same_count_int
1961 }
1962 {
1963     { 0 }
1964     {
1965         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1966         {
1967             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1968             {
1969                 \exp_not:V \l__zrefclever_listsep_tl
1970                 \_zrefclever_get_ref:V
1971                 \l__zrefclever_range_beg_label_tl
1972             }
1973         }
1974     }
1975     { 1 }
1976     {
1977         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1978         {
1979             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1980             {
1981                 \exp_not:V \l__zrefclever_listsep_tl
1982                 \_zrefclever_get_ref:V
1983                 \l__zrefclever_range_beg_label_tl
1984             }
1985             \exp_not:V \l__zrefclever_listsep_tl
1986             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1987         }
1988     }
1989 }
1990 {
1991     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1992     {
1993         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1994         {
1995             \exp_not:V \l__zrefclever_listsep_tl
1996             \_zrefclever_get_ref:V
1997             \l__zrefclever_range_beg_label_tl
1998         }
1999         \exp_not:V \l__zrefclever_rangeseq_tl

```

```

2000             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
2001         }
2002     }
2003 }
2004 % Reset counters.
2005 \int_zero:N \l__zrefclever_range_count_int
2006 \int_zero:N \l__zrefclever_range_same_count_int
2007 }
2008 }
2009 % Step label counter for next iteration.
2010 \int_incr:N \l__zrefclever_label_count_int
2011 }

```

(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 T_EXnical requirements of these functions. This because, as we are processing the label stack and accumulating content in the queue, we are using a number of variables which are transient to the current label, the label properties among them, but not only. Hence, these variables *must* be expanded to their current values to be stored in the queue. Indeed, _zrefclever_get_ref:n and _zrefclever_get_ref_first: get called, as they must, in the context of x type expansions. But we don’t want to expand the values of the variables themselves, so we need to get current values, but stop expansion after that. In particular, reference options given by the user should reach the stream for its final typesetting (when the queue itself gets typeset) *unmodified* (“no manipulation”, to use the n signature jargon). We also need to prevent premature expansion of material that can’t be expanded at this point (e.g. grouping, \zref@default or \hyper@@link). In a nutshell, the job of these two functions is putting the pieces in place, but with proper expansion control.

_zrefclever_ref_default: Default values for undefined references and undefined type names, respectively. We are ultimately using \zref@default, but calls to it should be made through these internal functions, according to the case. As a bonus, we don’t need to protect them with \exp_not:N, as \zref@default would require, since we already define them protected.

```

2012 \cs_new_protected:Npn \_zrefclever_ref_default:
2013 { \zref@default }
2014 \cs_new_protected:Npn \_zrefclever_name_default:
2015 { \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:.


```

2016 \cs_new:Npn \__zrefclever_get_ref:n {<label>}
2017 {
2018   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2019   {
2020     \bool_if:nTF
2021     {
2022       \l__zrefclever_use_hyperref_bool &&
2023       ! \l__zrefclever_link_star_bool
2024     }
2025     {
2026       \exp_not:N \group_begin:
2027       \exp_not:V \l__zrefclever_reffont_out_tl
2028       \exp_not:V \l__zrefclever_refpre_out_tl
2029       \exp_not:N \group_begin:
2030       \exp_not:V \l__zrefclever_reffont_in_tl
2031       % It's two '@s', but escaped for DocStrip.
2032       \exp_not:N \hyper@@link
2033       {
2034         \zref@ifrefcontainsprop {#1} { urluse }
2035         { \zref@extractdefault {#1} { urluse } { } }
2036         { \zref@extractdefault {#1} { url } { } }
2037       }
2038       { \zref@extractdefault {#1} { anchor } { } }
2039       {
2040         \exp_not:V \l__zrefclever_refpre_in_tl
2041         \zref@extractdefault {#1}
2042         { \l__zrefclever_ref_property_tl } { }
2043         \exp_not:V \l__zrefclever_refpos_in_tl
2044       }
2045       \exp_not:N \group_end:
2046       \exp_not:V \l__zrefclever_refpos_out_tl
2047       \exp_not:N \group_end:
2048     }
2049     {
2050       \exp_not:N \group_begin:
2051       \exp_not:V \l__zrefclever_reffont_out_tl
2052       \exp_not:V \l__zrefclever_refpre_out_tl
2053       \exp_not:N \group_begin:
2054       \exp_not:V \l__zrefclever_reffont_in_tl
2055       \exp_not:V \l__zrefclever_refpre_in_tl
2056       \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2057       \exp_not:V \l__zrefclever_refpos_in_tl
2058       \exp_not:N \group_end:
2059       \exp_not:V \l__zrefclever_refpos_out_tl
2060       \exp_not:N \group_end:
2061     }
2062   }
2063   { \__zrefclever_ref_default: }
2064 }
2065 \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.

```

2066 \cs_new:Npn \_zrefclever_get_ref_first:
2067 {
2068   \zref@ifrefundefined { \l\_zrefclever_type_first_label_tl }
2069   { \_zrefclever_ref_default: }
2070   {
2071     \bool_if:NTF \l\_zrefclever_name_in_link_bool
2072     {
2073       \zref@ifrefcontainsprop
2074       { \l\_zrefclever_type_first_label_tl }
2075       { \l\_zrefclever_ref_property_tl }
2076       {
2077         % It's two '@s', but escaped for DocStrip.
2078         \exp_not:N \hyper@@link
2079         {
2080           \zref@ifrefcontainsprop
2081           { \l\_zrefclever_type_first_label_tl } { urluse }
2082           {
2083             \zref@extractdefault
2084             { \l\_zrefclever_type_first_label_tl }
2085             { urluse } { }
2086           }
2087           {
2088             \zref@extractdefault
2089             { \l\_zrefclever_type_first_label_tl }
2090             { url } { }
2091           }
2092         }
2093       }
2094       {
2095         \zref@extractdefault
2096         { \l\_zrefclever_type_first_label_tl }
2097         { anchor } { }
2098       }
2099       {
2100         \exp_not:N \group_begin:
2101         \exp_not:V \l\_zrefclever_namefont_tl
2102         \exp_not:V \l\_zrefclever_type_name_tl
2103         \exp_not:N \group_end:
2104         \exp_not:V \l\_zrefclever_namesep_tl
2105         \exp_not:N \group_begin:
2106         \exp_not:V \l\_zrefclever_reffont_out_tl
2107         \exp_not:V \l\_zrefclever_refpre_out_tl
2108         \exp_not:N \group_begin:
2109         \exp_not:V \l\_zrefclever_reffont_in_tl
2110         \exp_not:V \l\_zrefclever_refpre_in_tl
2111         \zref@extractdefault

```

```

2111         { \l__zrefclever_type_first_label_tl }
2112         { \l__zrefclever_ref_property_tl } { }
2113         \exp_not:V \l__zrefclever_refpos_in_tl
2114         \exp_not:N \group_end:
2115         % hyperlink makes it's own group, we'd like to close the
2116         % 'refpre-out' group after 'refpos-out', but... we close
2117         % it here, and give the trailing 'refpos-out' its own
2118         % group. This will result that formatting given to
2119         % 'refpre-out' will not reach 'refpos-out', but I see no
2120         % alternative, and this has to be handled specially.
2121         \exp_not:N \group_end:
2122     }
2123     \exp_not:N \group_begin:
2124     % Ditto: special treatment.
2125     \exp_not:V \l__zrefclever_reffont_out_tl
2126     \exp_not:V \l__zrefclever_refpos_out_tl
2127     \exp_not:N \group_end:
2128 }
2129 {
2130     \exp_not:N \group_begin:
2131     \exp_not:V \l__zrefclever_namefont_tl
2132     \exp_not:V \l__zrefclever_type_name_tl
2133     \exp_not:N \group_end:
2134     \exp_not:V \l__zrefclever_namesep_tl
2135     \__zrefclever_ref_default:
2136 }
2137 }
2138 {
2139     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2140     {
2141         \__zrefclever_name_default:
2142         \exp_not:V \l__zrefclever_namesep_tl
2143     }
2144     {
2145         \exp_not:N \group_begin:
2146         \exp_not:V \l__zrefclever_namefont_tl
2147         \exp_not:V \l__zrefclever_type_name_tl
2148         \exp_not:N \group_end:
2149         \exp_not:V \l__zrefclever_namesep_tl
2150     }
2151 }
2152 \zref@ifrefcontainsprop
2153 { \l__zrefclever_type_first_label_tl }
2154 { \l__zrefclever_ref_property_tl }
2155 {
2156     \bool_if:nTF
2157     {
2158         \l__zrefclever_use_hyperref_bool &&
2159         ! \l__zrefclever_link_star_bool
2160     }
2161     {
2162         \exp_not:N \group_begin:
2163         \exp_not:V \l__zrefclever_reffont_out_tl
2164         \exp_not:V \l__zrefclever_refpre_out_tl
2165         \exp_not:N \group_end:

```

```

2165 \exp_not:V \l__zrefclever_reffont_in_tl
2166 % It's two '@s', but escaped for DocStrip.
2167 \exp_not:N \hyper@@link
2168 {
2169   \zref@ifrefcontainsprop
2170   { \l__zrefclever_type_first_label_tl } { urluse }
2171   {
2172     \zref@extractdefault
2173     { \l__zrefclever_type_first_label_tl }
2174     { urluse } { }
2175   }
2176   {
2177     \zref@extractdefault
2178     { \l__zrefclever_type_first_label_tl }
2179     { url } { }
2180   }
2181 }
2182 {
2183   \zref@extractdefault
2184   { \l__zrefclever_type_first_label_tl }
2185   { anchor } { }
2186 }
2187 {
2188   \exp_not:V \l__zrefclever_refpre_in_tl
2189   \zref@extractdefault
2190   { \l__zrefclever_type_first_label_tl }
2191   { \l__zrefclever_ref_property_tl } { }
2192   \exp_not:V \l__zrefclever_refpos_in_tl
2193 }
2194 \exp_not:N \group_end:
2195 \exp_not:V \l__zrefclever_refpos_out_tl
2196 \exp_not:N \group_end:
2197 }
2198 {
2199   \exp_not:N \group_begin:
2200   \exp_not:V \l__zrefclever_reffont_out_tl
2201   \exp_not:V \l__zrefclever_refpre_out_tl
2202   \exp_not:N \group_begin:
2203   \exp_not:V \l__zrefclever_reffont_in_tl
2204   \exp_not:V \l__zrefclever_refpre_in_tl
2205   \zref@extractdefault
2206   { \l__zrefclever_type_first_label_tl }
2207   { \l__zrefclever_ref_property_tl } { }
2208   \exp_not:V \l__zrefclever_refpos_in_tl
2209   \exp_not:N \group_end:
2210   \exp_not:V \l__zrefclever_refpos_out_tl
2211   \exp_not:N \group_end:
2212 }
2213 }
2214 { \__zrefclever_ref_default: }
2215 }
2216 }
2217 }

```

(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`.

```

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

```

```

2260     }
2261     \l__zrefclever_name_format_tl
2262     \l__zrefclever_type_name_tl
2263     {
2264         \__zrefclever_get_type_transl:xxxNF
2265         { \l__zrefclever_ref_language_tl }
2266         { \l__zrefclever_type_first_label_type_tl }
2267         { \l__zrefclever_name_format_tl }
2268         \l__zrefclever_type_name_tl
2269         {
2270             \tl_clear:N \l__zrefclever_type_name_tl
2271             \msg_warning:nnx { zref-clever } { missing-name }
2272             { \l__zrefclever_type_first_label_type_tl }
2273         }
2274     }
2275 }
2276 {
2277     \prop_get:cVNF
2278     {
2279         l__zrefclever_type_
2280         \l__zrefclever_type_first_label_type_tl _options_prop
2281     }
2282     \l__zrefclever_name_format_tl
2283     \l__zrefclever_type_name_tl
2284     {
2285         \prop_get:cVNF
2286         {
2287             l__zrefclever_type_
2288             \l__zrefclever_type_first_label_type_tl _options_prop
2289         }
2290         \l__zrefclever_name_format_fallback_tl
2291         \l__zrefclever_type_name_tl
2292         {
2293             \__zrefclever_get_type_transl:xxxNF
2294             { \l__zrefclever_ref_language_tl }
2295             { \l__zrefclever_type_first_label_type_tl }
2296             { \l__zrefclever_name_format_tl }
2297             \l__zrefclever_type_name_tl
2298             {
2299                 \__zrefclever_get_type_transl:xxxNF
2300                 { \l__zrefclever_ref_language_tl }
2301                 { \l__zrefclever_type_first_label_type_tl }
2302                 { \l__zrefclever_name_format_fallback_tl }
2303                 \l__zrefclever_type_name_tl
2304                 {
2305                     \tl_clear:N \l__zrefclever_type_name_tl
2306                     \msg_warning:nnx { zref-clever }
2307                     { missing-name }
2308                     { \l__zrefclever_type_first_label_type_tl }
2309                 }
2310             }
2311         }
2312     }
2313 }

```

```

2314     }
2315 }
2316
2317 % Signal whether the type name is to be included in the hyperlink or not.
2318 \bool_lazy_any:nTF
2319 {
2320   { ! \l__zrefclever_use_hyperref_bool }
2321   { \l__zrefclever_link_star_bool }
2322   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2323   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2324 }
2325 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2326 {
2327   \bool_lazy_any:nTF
2328   {
2329     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2330     {
2331       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2332       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2333     }
2334     {
2335       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2336       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2337       \l__zrefclever_typeset_last_bool &&
2338       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2339     }
2340   }
2341   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2342   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2343 }
2344 }

```

(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>}
2345 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2346 {
2347   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2348   {
2349     \exp_args:Nxx \tl_if_eq:nnT
2350     { \zref@extractdefault {#1} { zc@pgfmt } { } }
2351     { \zref@extractdefault {#2} { zc@pgfmt } { } }
2352     {
2353       \int_compare:nNnTF
2354       { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2355       =

```

```

2356         { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2357     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2358     {
2359         \int_compare:nNnT
2360             { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2361             =
2362             { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2363         {
2364             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2365             \bool_set_true:N \l__zrefclever_next_is_same_bool
2366         }
2367     }
2368 }
2369 }
2370 {
2371     \exp_args:Nxx \tl_if_eq:nnT
2372     { \zref@extractdefault {#1} { counter } { } }
2373     { \zref@extractdefault {#2} { counter } { } }
2374     {
2375         \exp_args:Nxx \tl_if_eq:nnT
2376         { \zref@extractdefault {#1} { zc@enclval } { } }
2377         { \zref@extractdefault {#2} { zc@enclval } { } }
2378         {
2379             \int_compare:nNnTF
2380                 { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2381                 =
2382                 { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2383             { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2384             {
2385                 \int_compare:nNnT
2386                     { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2387                     =
2388                     { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2389                 {
2390                     \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2391                     \bool_set_true:N \l__zrefclever_next_is_same_bool
2392                 }
2393             }
2394         }
2395     }
2396 }
2397 }

```

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

```

```

2447             \msg_warning:nnn { zref-clever }
2448             { missing-string } {#1}
2449         }
2450     }
2451 }
2452 }
2453 }
2454 }

```

(End definition for `__zrefclever_get_ref_string:nN`.)

```

\__zrefclever_get_ref_font:nN      \__zrefclever_get_ref_font:nN {<option>} {<tl variable>}
2455 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2456 {
2457     % First attempt: general options.
2458     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2459     {
2460         % If not found, try type specific options.
2461         \bool_lazy_and:nnTF
2462         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2463         {
2464             \prop_if_exist_p:c
2465             {
2466                 l__zrefclever_type_
2467                 \l__zrefclever_label_type_a_tl _options_prop
2468             }
2469         }
2470         {
2471             \prop_get:cnNF
2472             {
2473                 l__zrefclever_type_
2474                 \l__zrefclever_label_type_a_tl _options_prop
2475             }
2476             {#1} #2
2477             { \tl_clear:N #2 }
2478         }
2479         { \tl_clear:N #2 }
2480     }
2481 }

```

(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.

```
2482 \</package>
```

10 Dictionaries

10.1 English

```
2483 <package>\zcDeclareLanguage { english }
2484 <package>\zcDeclareLanguageAlias { american } { english }
2485 <package>\zcDeclareLanguageAlias { australian } { english }
2486 <package>\zcDeclareLanguageAlias { british } { english }
2487 <package>\zcDeclareLanguageAlias { canadian } { english }
2488 <package>\zcDeclareLanguageAlias { newzealand } { english }
2489 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2490 <package>\zcDeclareLanguageAlias { USenglish } { english }
2491 <*dict-english>
2492 namesep = {\nobreakspace} ,
2493 pairsep = {\~and\nobreakspace} ,
2494 listsep = {\~,~} ,
2495 lastsep = {\~and\nobreakspace} ,
2496 tpairsep = {\~and\nobreakspace} ,
2497 tlistsep = {\~,~} ,
2498 tlastsep = {\~,~and\nobreakspace} ,
2499 notesep = {\~} ,
2500 rangesep = {\~to\nobreakspace} ,
2501
2502 type = part ,
2503   Name-sg = Part ,
2504   name-sg = part ,
2505   Name-pl = Parts ,
2506   name-pl = parts ,
2507
2508 type = chapter ,
2509   Name-sg = Chapter ,
2510   name-sg = chapter ,
2511   Name-pl = Chapters ,
```

```

2512     name-pl = chapters ,
2513
2514 type = section ,
2515     Name-sg = Section ,
2516     name-sg = section ,
2517     Name-pl = Sections ,
2518     name-pl = sections ,
2519
2520 type = paragraph ,
2521     Name-sg = Paragraph ,
2522     name-sg = paragraph ,
2523     Name-pl = Paragraphs ,
2524     name-pl = paragraphs ,
2525     Name-sg-ab = Par. ,
2526     name-sg-ab = par. ,
2527     Name-pl-ab = Par. ,
2528     name-pl-ab = par. ,
2529
2530 type = appendix ,
2531     Name-sg = Appendix ,
2532     name-sg = appendix ,
2533     Name-pl = Appendices ,
2534     name-pl = appendices ,
2535
2536 type = page ,
2537     Name-sg = Page ,
2538     name-sg = page ,
2539     Name-pl = Pages ,
2540     name-pl = pages ,
2541     name-sg-ab = p. ,
2542     name-pl-ab = pp. ,
2543
2544 type = line ,
2545     Name-sg = Line ,
2546     name-sg = line ,
2547     Name-pl = Lines ,
2548     name-pl = lines ,
2549
2550 type = figure ,
2551     Name-sg = Figure ,
2552     name-sg = figure ,
2553     Name-pl = Figures ,
2554     name-pl = figures ,
2555     Name-sg-ab = Fig. ,
2556     name-sg-ab = fig. ,
2557     Name-pl-ab = Figs. ,
2558     name-pl-ab = figs. ,
2559
2560 type = table ,
2561     Name-sg = Table ,
2562     name-sg = table ,
2563     Name-pl = Tables ,
2564     name-pl = tables ,
2565

```

```

2566 type = item ,
2567     Name-sg = Item ,
2568     name-sg = item ,
2569     Name-pl = Items ,
2570     name-pl = items ,
2571
2572 type = footnote ,
2573     Name-sg = Footnote ,
2574     name-sg = footnote ,
2575     Name-pl = Footnotes ,
2576     name-pl = footnotes ,
2577
2578 type = note ,
2579     Name-sg = Note ,
2580     name-sg = note ,
2581     Name-pl = Notes ,
2582     name-pl = notes ,
2583
2584 type = equation ,
2585     Name-sg = Equation ,
2586     name-sg = equation ,
2587     Name-pl = Equations ,
2588     name-pl = equations ,
2589     Name-sg-ab = Eq. ,
2590     name-sg-ab = eq. ,
2591     Name-pl-ab = Eqs. ,
2592     name-pl-ab = eqs. ,
2593     refpre-in = {(} ,
2594     refpos-in = {)} ,
2595
2596 type = theorem ,
2597     Name-sg = Theorem ,
2598     name-sg = theorem ,
2599     Name-pl = Theorems ,
2600     name-pl = theorems ,
2601
2602 type = lemma ,
2603     Name-sg = Lemma ,
2604     name-sg = lemma ,
2605     Name-pl = Lemmas ,
2606     name-pl = lemmas ,
2607
2608 type = corollary ,
2609     Name-sg = Corollary ,
2610     name-sg = corollary ,
2611     Name-pl = Corollaries ,
2612     name-pl = corollaries ,
2613
2614 type = proposition ,
2615     Name-sg = Proposition ,
2616     name-sg = proposition ,
2617     Name-pl = Propositions ,
2618     name-pl = propositions ,
2619

```

```

2620 type = definition ,
2621     Name-sg = Definition ,
2622     name-sg = definition ,
2623     Name-pl = Definitions ,
2624     name-pl = definitions ,
2625
2626 type = proof ,
2627     Name-sg = Proof ,
2628     name-sg = proof ,
2629     Name-pl = Proofs ,
2630     name-pl = proofs ,
2631
2632 type = result ,
2633     Name-sg = Result ,
2634     name-sg = result ,
2635     Name-pl = Results ,
2636     name-pl = results ,
2637
2638 type = example ,
2639     Name-sg = Example ,
2640     name-sg = example ,
2641     Name-pl = Examples ,
2642     name-pl = examples ,
2643
2644 type = remark ,
2645     Name-sg = Remark ,
2646     name-sg = remark ,
2647     Name-pl = Remarks ,
2648     name-pl = remarks ,
2649
2650 type = algorithm ,
2651     Name-sg = Algorithm ,
2652     name-sg = algorithm ,
2653     Name-pl = Algorithms ,
2654     name-pl = algorithms ,
2655
2656 type = listing ,
2657     Name-sg = Listing ,
2658     name-sg = listing ,
2659     Name-pl = Listings ,
2660     name-pl = listings ,
2661
2662 type = exercise ,
2663     Name-sg = Exercise ,
2664     name-sg = exercise ,
2665     Name-pl = Exercises ,
2666     name-pl = exercises ,
2667
2668 type = solution ,
2669     Name-sg = Solution ,
2670     name-sg = solution ,
2671     Name-pl = Solutions ,
2672     name-pl = solutions ,
2673 </dict-english>

```

10.2 German

```
2674 <package>\zcDeclareLanguage { german }
2675 <package>\zcDeclareLanguageAlias { austrian      } { german }
2676 <package>\zcDeclareLanguageAlias { germanb       } { german }
2677 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2678 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2679 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2680 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2681 <*dict-german>

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

```

2726     name-pl = Seiten ,
2727
2728 type = line ,
2729     Name-sg = Zeile ,
2730     name-sg = Zeile ,
2731     Name-pl = Zeilen ,
2732     name-pl = Zeilen ,
2733
2734 type = figure ,
2735     Name-sg = Abbildung ,
2736     name-sg = Abbildung ,
2737     Name-pl = Abbildungen ,
2738     name-pl = Abbildungen ,
2739     Name-sg-ab = Abb. ,
2740     name-sg-ab = Abb. ,
2741     Name-pl-ab = Abb. ,
2742     name-pl-ab = Abb. ,
2743
2744 type = table ,
2745     Name-sg = Tabelle ,
2746     name-sg = Tabelle ,
2747     Name-pl = Tabellen ,
2748     name-pl = Tabellen ,
2749
2750 type = item ,
2751     Name-sg = Punkt ,
2752     name-sg = Punkt ,
2753     Name-pl = Punkte ,
2754     name-pl = Punkte ,
2755
2756 type = footnote ,
2757     Name-sg = Fußnote ,
2758     name-sg = Fußnote ,
2759     Name-pl = Fußnoten ,
2760     name-pl = Fußnoten ,
2761
2762 type = note ,
2763     Name-sg = Anmerkung ,
2764     name-sg = Anmerkung ,
2765     Name-pl = Anmerkungen ,
2766     name-pl = Anmerkungen ,
2767
2768 type = equation ,
2769     Name-sg = Gleichung ,
2770     name-sg = Gleichung ,
2771     Name-pl = Gleichungen ,
2772     name-pl = Gleichungen ,
2773     refpre-in = {()} ,
2774     refpos-in = {} } ,
2775
2776 type = theorem ,
2777     Name-sg = Theorem ,
2778     name-sg = Theorem ,
2779     Name-pl = Theoreme ,

```



```

2780     name-pl = Theoreme ,
2781
2782 type = lemma ,
2783     Name-sg = Lemma ,
2784     name-sg = Lemma ,
2785     Name-pl = Lemmata ,
2786     name-pl = Lemmata ,
2787
2788 type = corollary ,
2789     Name-sg = Korollar ,
2790     name-sg = Korollar ,
2791     Name-pl = Korollare ,
2792     name-pl = Korollare ,
2793
2794 type = proposition ,
2795     Name-sg = Satz ,
2796     name-sg = Satz ,
2797     Name-pl = Sätze ,
2798     name-pl = Sätze ,
2799
2800 type = definition ,
2801     Name-sg = Definition ,
2802     name-sg = Definition ,
2803     Name-pl = Definitionen ,
2804     name-pl = Definitionen ,
2805
2806 type = proof ,
2807     Name-sg = Beweis ,
2808     name-sg = Beweis ,
2809     Name-pl = Beweise ,
2810     name-pl = Beweise ,
2811
2812 type = result ,
2813     Name-sg = Ergebnis ,
2814     name-sg = Ergebnis ,
2815     Name-pl = Ergebnisse ,
2816     name-pl = Ergebnisse ,
2817
2818 type = example ,
2819     Name-sg = Beispiel ,
2820     name-sg = Beispiel ,
2821     Name-pl = Beispiele ,
2822     name-pl = Beispiele ,
2823
2824 type = remark ,
2825     Name-sg = Bemerkung ,
2826     name-sg = Bemerkung ,
2827     Name-pl = Bemerkungen ,
2828     name-pl = Bemerkungen ,
2829
2830 type = algorithm ,
2831     Name-sg = Algorithmus ,
2832     name-sg = Algorithmus ,
2833     Name-pl = Algorithmen ,

```

```

2834   name-pl = Algorithmen ,
2835
2836   type = listing ,
2837   Name-sg = Listing , % CHECK
2838   name-sg = Listing , % CHECK
2839   Name-pl = Listings , % CHECK
2840   name-pl = Listings , % CHECK
2841
2842   type = exercise ,
2843   Name-sg = Übungsaufgabe ,
2844   name-sg = Übungsaufgabe ,
2845   Name-pl = Übungsaufgaben ,
2846   name-pl = Übungsaufgaben ,
2847
2848   type = solution ,
2849   Name-sg = Lösung ,
2850   name-sg = Lösung ,
2851   Name-pl = Lösungen ,
2852   name-pl = Lösungen ,
2853 </dict-german>

```

10.3 French

```

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

```

```

2885     Name-pl = Sections ,
2886     name-pl = sections ,
2887
2888 type = paragraph ,
2889     Name-sg = Paragraphe ,
2890     name-sg = paragraphe ,
2891     Name-pl = Paragraphes ,
2892     name-pl = paragraphes ,
2893
2894 type = appendix ,
2895     Name-sg = Annexe ,
2896     name-sg = annexe ,
2897     Name-pl = Annexes ,
2898     name-pl = annexes ,
2899
2900 type = page ,
2901     Name-sg = Page ,
2902     name-sg = page ,
2903     Name-pl = Pages ,
2904     name-pl = pages ,
2905
2906 type = line ,
2907     Name-sg = Ligne ,
2908     name-sg = ligne ,
2909     Name-pl = Lignes ,
2910     name-pl = lignes ,
2911
2912 type = figure ,
2913     Name-sg = Figure ,
2914     name-sg = figure ,
2915     Name-pl = Figures ,
2916     name-pl = figures ,
2917
2918 type = table ,
2919     Name-sg = Table ,
2920     name-sg = table ,
2921     Name-pl = Tables ,
2922     name-pl = tables ,
2923
2924 type = item ,
2925     Name-sg = Point ,
2926     name-sg = point ,
2927     Name-pl = Points ,
2928     name-pl = points ,
2929
2930 type = footnote ,
2931     Name-sg = Note ,
2932     name-sg = note ,
2933     Name-pl = Notes ,
2934     name-pl = notes ,
2935
2936 type = note ,
2937     Name-sg = Note ,
2938     name-sg = note ,

```

```

2939   Name-pl = Notes ,
2940   name-pl = notes ,
2941
2942   type = equation ,
2943   Name-sg = Équation ,
2944   name-sg = équation ,
2945   Name-pl = Équations ,
2946   name-pl = équations ,
2947   refpre-in = {()} ,
2948   refpos-in = {} } ,
2949
2950   type = theorem ,
2951   Name-sg = Théorème ,
2952   name-sg = théorème ,
2953   Name-pl = Théorèmes ,
2954   name-pl = théorèmes ,
2955
2956   type = lemma ,
2957   Name-sg = Lemme ,
2958   name-sg = lemme ,
2959   Name-pl = Lemmes ,
2960   name-pl = lemmes ,
2961
2962   type = corollary ,
2963   Name-sg = Corollaire ,
2964   name-sg = corollaire ,
2965   Name-pl = Corollaires ,
2966   name-pl = corollaires ,
2967
2968   type = proposition ,
2969   Name-sg = Proposition ,
2970   name-sg = proposition ,
2971   Name-pl = Propositions ,
2972   name-pl = propositions ,
2973
2974   type = definition ,
2975   Name-sg = Définition ,
2976   name-sg = définition ,
2977   Name-pl = Définitions ,
2978   name-pl = définitions ,
2979
2980   type = proof ,
2981   Name-sg = Démonstration ,
2982   name-sg = démonstration ,
2983   Name-pl = Démonstrations ,
2984   name-pl = démonstrations ,
2985
2986   type = result ,
2987   Name-sg = Résultat ,
2988   name-sg = résultat ,
2989   Name-pl = Résultats ,
2990   name-pl = résultats ,
2991
2992   type = example ,

```

```

2993 Name-sg = Exemple ,
2994 name-sg = exemple ,
2995 Name-pl = Exemples ,
2996 name-pl = exemples ,
2997
2998 type = remark ,
2999 Name-sg = Remarque ,
3000 name-sg = remarque ,
3001 Name-pl = Remarques ,
3002 name-pl = remarques ,
3003
3004 type = algorithm ,
3005 Name-sg = Algorithme ,
3006 name-sg = algorithme ,
3007 Name-pl = Algorithmes ,
3008 name-pl = algorithmes ,
3009
3010 type = listing ,
3011 Name-sg = Liste ,
3012 name-sg = liste ,
3013 Name-pl = Listes ,
3014 name-pl = listes ,
3015
3016 type = exercise ,
3017 Name-sg = Exercice ,
3018 name-sg = exercice ,
3019 Name-pl = Exercices ,
3020 name-pl = exercices ,
3021
3022 type = solution ,
3023 Name-sg = Solution ,
3024 name-sg = solution ,
3025 Name-pl = Solutions ,
3026 name-pl = solutions ,
3027 </dict-french>

```

10.4 Portuguese

```

3028 <package>\zcDeclareLanguage { portuguese }
3029 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3030 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
3031 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3032 <*dict-portuguese>
3033 namesep = {\nobreakspace} ,
3034 pairsep = {\~e\nobreakspace} ,
3035 listsep = {\~,~} ,
3036 lastsep = {\~e\nobreakspace} ,
3037 tpairsep = {\~e\nobreakspace} ,
3038 tlistsep = {\~,~} ,
3039 tlastsep = {\~e\nobreakspace} ,
3040 notesep = {\~} ,
3041 rangesep = {\~a\nobreakspace} ,
3042
3043 type = part ,

```

```

3044     Name-sg = Parte ,
3045     name-sg = parte ,
3046     Name-pl = Partes ,
3047     name-pl = partes ,
3048
3049     type = chapter ,
3050     Name-sg = Capítulo ,
3051     name-sg = capítulo ,
3052     Name-pl = Capítulos ,
3053     name-pl = capítulos ,
3054
3055     type = section ,
3056     Name-sg = Seção ,
3057     name-sg = seção ,
3058     Name-pl = Seções ,
3059     name-pl = seções ,
3060
3061     type = paragraph ,
3062     Name-sg = Parágrafo ,
3063     name-sg = parágrafo ,
3064     Name-pl = Parágrafos ,
3065     name-pl = parágrafos ,
3066     Name-sg-ab = Par. ,
3067     name-sg-ab = par. ,
3068     Name-pl-ab = Par. ,
3069     name-pl-ab = par. ,
3070
3071     type = appendix ,
3072     Name-sg = Apêndice ,
3073     name-sg = apêndice ,
3074     Name-pl = Apêndices ,
3075     name-pl = apêndices ,
3076
3077     type = page ,
3078     Name-sg = Página ,
3079     name-sg = página ,
3080     Name-pl = Páginas ,
3081     name-pl = páginas ,
3082     name-sg-ab = p. ,
3083     name-pl-ab = pp. ,
3084
3085     type = line ,
3086     Name-sg = Linha ,
3087     name-sg = linha ,
3088     Name-pl = Linhas ,
3089     name-pl = linhas ,
3090
3091     type = figure ,
3092     Name-sg = Figura ,
3093     name-sg = figura ,
3094     Name-pl = Figuras ,
3095     name-pl = figuras ,
3096     Name-sg-ab = Fig. ,
3097     name-sg-ab = fig. ,

```

```

3098     Name-pl-ab = Figs. ,
3099     name-pl-ab = figs. ,
3100
3101 type = table ,
3102     Name-sg = Tabela ,
3103     name-sg = tabela ,
3104     Name-pl = Tabelas ,
3105     name-pl = tabelas ,
3106
3107 type = item ,
3108     Name-sg = Item ,
3109     name-sg = item ,
3110     Name-pl = Itens ,
3111     name-pl = itens ,
3112
3113 type = footnote ,
3114     Name-sg = Nota ,
3115     name-sg = nota ,
3116     Name-pl = Notas ,
3117     name-pl = notas ,
3118
3119 type = note ,
3120     Name-sg = Nota ,
3121     name-sg = nota ,
3122     Name-pl = Notas ,
3123     name-pl = notas ,
3124
3125 type = equation ,
3126     Name-sg = Equação ,
3127     name-sg = equação ,
3128     Name-pl = Equações ,
3129     name-pl = equações ,
3130     Name-sg-ab = Eq. ,
3131     name-sg-ab = eq. ,
3132     Name-pl-ab = Eqs. ,
3133     name-pl-ab = eqs. ,
3134     refpre-in = {()} ,
3135     refpos-in = {} } ,
3136
3137 type = theorem ,
3138     Name-sg = Teorema ,
3139     name-sg = teorema ,
3140     Name-pl = Teoremas ,
3141     name-pl = teoremas ,
3142
3143 type = lemma ,
3144     Name-sg = Lema ,
3145     name-sg = lema ,
3146     Name-pl = Lemas ,
3147     name-pl = lemas ,
3148
3149 type = corollary ,
3150     Name-sg = Corolário ,
3151     name-sg = corolário ,

```

```

3152     Name-pl = Corolários ,
3153     name-pl = corolários ,
3154
3155     type = proposition ,
3156     Name-sg = Proposição ,
3157     name-sg = proposição ,
3158     Name-pl = Proposições ,
3159     name-pl = proposições ,
3160
3161     type = definition ,
3162     Name-sg = Definição ,
3163     name-sg = definição ,
3164     Name-pl = Definições ,
3165     name-pl = definições ,
3166
3167     type = proof ,
3168     Name-sg = Demonstração ,
3169     name-sg = demonstração ,
3170     Name-pl = Demonstrações ,
3171     name-pl = demonstrações ,
3172
3173     type = result ,
3174     Name-sg = Resultado ,
3175     name-sg = resultado ,
3176     Name-pl = Resultados ,
3177     name-pl = resultados ,
3178
3179     type = example ,
3180     Name-sg = Exemplo ,
3181     name-sg = exemplo ,
3182     Name-pl = Exemplos ,
3183     name-pl = exemplos ,
3184
3185     type = remark ,
3186     Name-sg = Observação ,
3187     name-sg = observação ,
3188     Name-pl = Observações ,
3189     name-pl = observações ,
3190
3191     type = algorithm ,
3192     Name-sg = Algoritmo ,
3193     name-sg = algoritmo ,
3194     Name-pl = Algoritmos ,
3195     name-pl = algoritmos ,
3196
3197     type = listing ,
3198     Name-sg = Listagem ,
3199     name-sg = listagem ,
3200     Name-pl = Listagens ,
3201     name-pl = listagens ,
3202
3203     type = exercise ,
3204     Name-sg = Exercício ,
3205     name-sg = exercício ,

```



```

3206   Name-pl = Exercícios ,
3207   name-pl = exercícios ,
3208
3209   type = solution ,
3210   Name-sg = Solução ,
3211   name-sg = solução ,
3212   Name-pl = Soluções ,
3213   name-pl = soluções ,
3214 </dict-portuguese>

```

10.5 Spanish

```

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

```

```

3257 type = page ,
3258     Name-sg = Página ,
3259     name-sg = página ,
3260     Name-pl = Páginas ,
3261     name-pl = páginas ,
3262
3263 type = line ,
3264     Name-sg = Línea ,
3265     name-sg = línea ,
3266     Name-pl = Líneas ,
3267     name-pl = líneas ,
3268
3269 type = figure ,
3270     Name-sg = Figura ,
3271     name-sg = figura ,
3272     Name-pl = Figuras ,
3273     name-pl = figuras ,
3274
3275 type = table ,
3276     Name-sg = Cuadro ,
3277     name-sg = cuadro ,
3278     Name-pl = Cuadros ,
3279     name-pl = cuadros ,
3280
3281 type = item ,
3282     Name-sg = Punto ,
3283     name-sg = punto ,
3284     Name-pl = Puntos ,
3285     name-pl = puntos ,
3286
3287 type = footnote ,
3288     Name-sg = Nota ,
3289     name-sg = nota ,
3290     Name-pl = Notas ,
3291     name-pl = notas ,
3292
3293 type = note ,
3294     Name-sg = Nota ,
3295     name-sg = nota ,
3296     Name-pl = Notas ,
3297     name-pl = notas ,
3298
3299 type = equation ,
3300     Name-sg = Ecuación ,
3301     name-sg = ecuación ,
3302     Name-pl = Ecuaciones ,
3303     name-pl = ecuaciones ,
3304     refpre-in = {()} ,
3305     refpos-in = {} } ,
3306
3307 type = theorem ,
3308     Name-sg = Teorema ,
3309     name-sg = teorema ,
3310     Name-pl = Teoremas ,

```

```

3311     name-pl = teoremas ,
3312
3313 type = lemma ,
3314     Name-sg = Lema ,
3315     name-sg = lema ,
3316     Name-pl = Lemas ,
3317     name-pl = lemas ,
3318
3319 type = corollary ,
3320     Name-sg = Corolario ,
3321     name-sg = corolario ,
3322     Name-pl = Corolarios ,
3323     name-pl = corolarios ,
3324
3325 type = proposition ,
3326     Name-sg = Proposición ,
3327     name-sg = proposición ,
3328     Name-pl = Proposiciones ,
3329     name-pl = proposiciones ,
3330
3331 type = definition ,
3332     Name-sg = Definición ,
3333     name-sg = definición ,
3334     Name-pl = Definiciones ,
3335     name-pl = definiciones ,
3336
3337 type = proof ,
3338     Name-sg = Demostración ,
3339     name-sg = demostración ,
3340     Name-pl = Demostraciones ,
3341     name-pl = demostraciones ,
3342
3343 type = result ,
3344     Name-sg = Resultado ,
3345     name-sg = resultado ,
3346     Name-pl = Resultados ,
3347     name-pl = resultados ,
3348
3349 type = example ,
3350     Name-sg = Ejemplo ,
3351     name-sg = ejemplo ,
3352     Name-pl = Ejemplos ,
3353     name-pl = ejemplos ,
3354
3355 type = remark ,
3356     Name-sg = Observación ,
3357     name-sg = observación ,
3358     Name-pl = Observaciones ,
3359     name-pl = observaciones ,
3360
3361 type = algorithm ,
3362     Name-sg = Algoritmo ,
3363     name-sg = algoritmo ,
3364     Name-pl = Algoritmos ,

```

```

3365 name-pl = algoritmos ,
3366
3367 type = listing ,
3368 Name-sg = Listado ,
3369 name-sg = listado ,
3370 Name-pl = Listados ,
3371 name-pl = listados ,
3372
3373 type = exercise ,
3374 Name-sg = Ejercicio ,
3375 name-sg = ejercicio ,
3376 Name-pl = Ejercicios ,
3377 name-pl = ejercicios ,
3378
3379 type = solution ,
3380 Name-sg = Solución ,
3381 name-sg = solución ,
3382 Name-pl = Soluciones ,
3383 name-pl = soluciones ,
3384 </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	
\\	<u>103</u> , <u>109</u> , <u>118</u> , <u>119</u> , <u>124</u> , <u>125</u> , <u>134</u> , <u>135</u> , <u>145</u>
A	
\AddToHook	<u>91</u> , <u>532</u> , <u>547</u> , <u>657</u> , <u>727</u> , <u>752</u> , <u>781</u> , <u>783</u> , <u>834</u>
\appendix	<u>67</u>
\appendixname	<u>67</u>
B	
\babelname	<u>737</u>
\babelprovide	<u>12</u> , <u>24</u>
bool commands:	
\bool_case_true:	<u>2</u>
\bool_if:NTF	<u>288</u> , <u>297</u> , <u>661</u> , <u>665</u> , <u>1486</u> , <u>1581</u> , <u>1711</u> , <u>1733</u> , <u>1764</u> , <u>1821</u> , <u>1862</u> , <u>1885</u> , <u>1889</u> , <u>1895</u> , <u>1905</u> , <u>1911</u> , <u>2071</u>
\bool_if:nTF	<u>59</u> , <u>1142</u> , <u>1151</u> , <u>1160</u> , <u>1217</u> , <u>1245</u> , <u>1266</u> , <u>1352</u> , <u>1360</u> , <u>1500</u> , <u>1508</u> , <u>1745</u> , <u>1752</u> , <u>1759</u> , <u>2020</u> , <u>2155</u>
\bool_lazy_all:nTF	<u>2404</u>
\bool_lazy_and:nnTF	<u>1036</u> , <u>1054</u> , <u>2240</u> , <u>2461</u>
\bool_lazy_any:nTF	<u>2318</u> , <u>2327</u>
\bool_lazy_or:nnTF	<u>1040</u> , <u>2228</u>
\bool_new:N	<u>251</u> , <u>568</u> , <u>569</u> , <u>594</u> , <u>618</u> , <u>627</u> , <u>634</u> , <u>635</u> , <u>690</u> , <u>691</u> , <u>704</u> , <u>705</u> , <u>827</u> , <u>828</u> , <u>1065</u> , <u>1078</u> , <u>1392</u> , <u>1393</u> , <u>1403</u> , <u>1409</u> , <u>1410</u>
\bool_set:Nn	<u>1034</u>
\bool_set_false:N	<u>581</u> , <u>585</u> , <u>642</u> , <u>651</u> , <u>652</u> , <u>667</u> , <u>849</u> , <u>1214</u> , <u>1449</u> , <u>1492</u> , <u>1506</u> , <u>1520</u> , <u>1723</u> , <u>1860</u> , <u>1861</u> , <u>2325</u> , <u>2342</u>
\bool_set_true:N	<u>306</u> , <u>575</u> , <u>576</u> , <u>580</u> , <u>586</u> , <u>641</u> , <u>646</u> , <u>647</u> , <u>838</u> , <u>843</u> , <u>1229</u> , <u>1240</u> , <u>1255</u> , <u>1261</u> , <u>1276</u> , <u>1282</u> , <u>1310</u> , <u>1322</u> , <u>1457</u> , <u>1487</u> , <u>1493</u> , <u>1497</u> , <u>1524</u> , <u>1530</u> , <u>2341</u> , <u>2357</u> , <u>2364</u> , <u>2365</u> , <u>2383</u> , <u>2390</u> , <u>2391</u>
\bool_until_do:Nn	<u>1215</u> , <u>1450</u>
C	
clist commands:	
\clist_map_inline:nn	<u>477</u>
\counterwithin	<u>4</u>
cs commands:	
\cs_generate_variant:Nn	<u>55</u> , <u>56</u> , <u>302</u> , <u>310</u> , <u>945</u> , <u>951</u> , <u>1194</u> , <u>2065</u>
\cs_if_exist:NTF	<u>39</u> , <u>48</u> , <u>69</u>
\cs_new:Npn	<u>37</u> , <u>46</u> , <u>57</u> , <u>67</u> , <u>78</u> , <u>2016</u> , <u>2066</u>

\cs_new_protected:Npn	252, 303, 311, 317, 438, 940, 946, 1029, 1082, 1124, 1135, 1195, 1330, 1382, 1426, 1588, 1856, 2012, 2014, 2218, 2345, 2398, 2455
\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, 1251, 1272
\exp_args:Nnx	313
\exp_args:Nx	262
\exp_args:Nxx	1178, 1225, 1291, 2349, 2371, 2375
\exp_not:N	56, 1766, 1769, 1791, 1794, 1797, 2026, 2029, 2032, 2045, 2047, 2050, 2053, 2058, 2060, 2078, 2099, 2102, 2104, 2107, 2114, 2121, 2123, 2127, 2130, 2133, 2145, 2148, 2161, 2164, 2167, 2194, 2196, 2199, 2202, 2209, 2211
\exp_not:n	1610, 1626, 1638, 1643, 1666, 1680, 1684, 1696, 1700, 1734, 1735, 1767, 1790, 1795, 1796, 1925, 1938, 1945, 1969, 1981, 1985, 1995, 1999, 2027, 2028, 2030, 2040, 2043, 2046, 2051, 2052, 2054, 2055, 2057, 2059, 2100, 2101, 2103, 2105, 2106, 2108, 2109, 2113, 2125, 2126, 2131, 2132, 2134, 2142, 2146, 2147, 2149, 2162, 2163, 2165, 2188, 2192, 2195, 2200, 2201, 2203, 2204, 2208, 2210
\ExplSyntaxOn	12, 264
F	
file commands:	
\file_get:nnNTF	262
\fmtversion	3
G	
group commands:	
\group_begin:	93, 254, 305, 929, 1031, 1044, 1766, 1794, 2026, 2029, 2050, 2053, 2099, 2104, 2107, 2123, 2130, 2145, 2161, 2164, 2199, 2202
\group_end:	96, 300, 308, 937, 1047, 1062, 1791, 1797, 2045, 2047, 2058, 2060, 2102, 2114, 2121, 2127, 2133, 2148, 2194, 2196, 2209, 2211
I	
\IfBooleanTF	1068
\IfFormatAtLeastTF	3, 4
\input	12
int commands:	
\int_case:nnTF	1591, 1619, 1651, 1824, 1918, 1957
\int_compare:nNnTF	1182, 1230, 1295, 1311, 1338, 1340, 1384, 1552, 1606, 1640, 1813, 1815, 1873, 1898, 1942, 2353, 2359, 2379, 2385
\int_compare_p:nNn	1354, 1362, 2232, 2243, 2338
\int_eval:n	90
\int_incr:N	1851, 1888, 1890, 1904, 1906, 1910, 1912, 2010
\int_new:N	1079, 1080, 1394, 1395, 1406, 1407
\int_set:Nn	1339, 1341, 1345, 1348
\int_use:N	33, 35, 50
\int_zero:N	1332, 1333, 1435, 1436, 1437, 1438, 1850, 1852, 1853, 2005, 2006
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, 706, 718, 748, 786, 819, 822, 829, 839, 850, 861, 872, 890, 902, 952, 964, 985, 1008
\keys_set:nn	29, 33, 281, 844, 879, 885, 934, 1032
keyval commands:	
\keyval_parse:nnn	449, 503
L	
\labelformat	3
\language	24, 731
M	
\mainbabelname	24, 738
\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

<code>\@onlypreamble</code>	235, 249, 939	<code>\tl_if_empty_p:N</code>	1146, 1147, 1155, 1156, 1163, 1164, 1503, 1504, 1511, 1513, 2322, 2332, 2336, 2406, 2462
<code>\bbl@loaded</code>	25	<code>\tl_if_empty_p:n</code>	1221, 1222, 1248, 1269
<code>\bbl@main@language</code>	24, 732	<code>\tl_if_eq:NNTF</code>	1167, 1516
<code>\c@</code>	3	<code>\tl_if_eq:NnTF</code>	1085, 1117, 1344, 1347, 1372, 1375, 1464, 2347
<code>\c@page</code>	6, 94	<code>\tl_if_eq:nNTF</code>	1178, 1225, 1291, 1336, 2349, 2371, 2375
<code>\cl@</code>	4	<code>\tl_if_in:NnTF</code>	1251, 1272
<code>\hyper@link</code> 56, 1769, 2032, 2078, 2167		<code>\tl_if_novalue:nTF</code>	866, 907
<code>\p@...</code>	3	<code>\tl_map_break:n</code>	81
<code>\zref@addprop</code> 22, 32, 34, 36, 87, 88, 99		<code>\tl_map_tokens:Nn</code>	73
<code>\zref@default</code>	56, 2013, 2015	<code>\tl_new:N</code>	89, 169, 170, 522, 724, 725, 726, 818, 821, 1072, 1073, 1074, 1075, 1076, 1077, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1404, 1405, 1408, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425
<code>\zref@extractdefault</code>		<code>\tl_put_left:Nn</code>	1748, 1755, 1806
.	1127, 1138, 1140, 1179, 1180, 1183, 1185, 1198, 1202, 1206, 1210, 1226, 1227, 1231, 1233, 1253, 1274, 1385, 1387, 1472, 1477, 1775, 1780, 1786, 2035, 2036, 2038, 2041, 2056, 2083, 2088, 2094, 2110, 2172, 2177, 2183, 2189, 2205, 2350, 2351, 2354, 2356, 2360, 2362, 2372, 2373, 2376, 2377, 2380, 2382, 2386, 2388	<code>\tl_put_right:Nn</code>	1608, 1624, 1633, 1664, 1675, 1691, 1923, 1934, 1965, 1977, 1991, 2238, 2239, 2250
<code>\zref@ifpropundefined</code>	20	<code>\tl_reverse_items:n</code> 1194, 1200, 1204, 1208, 1212
<code>\zref@ifrefcontainsprop</code>	20, 1771, 2018, 2034, 2073, 2080, 2151, 2169	<code>\tl_set:Nn</code> 329, 527, 529, 535, 538, 554, 563, 731, 732, 737, 738, 741, 742, 745, 758, 766, 773, 794, 802, 809, 884, 958, 1126, 1137, 1139, 1197, 1199, 1201, 1203, 1205, 1207, 1209, 1211, 1300, 1302, 1304, 1306, 1466, 1467, 1470, 1475, 1597, 1599, 1731, 1762, 1877, 1879, 1902, 2234, 2235, 2248
<code>\zref@ifrefundefined</code>		<code>\tl_set_eq:NN</code>	1844
.	1092, 1094, 1106, 1489, 1491, 1496, 1540, 1715, 1724, 1864, 2068, 2220	<code>\tl_tail:N</code>	1301, 1303, 1305, 1307
<code>\ZREF@mainlist</code> 22, 32, 34, 36, 87, 88, 99		<code>\l_tmpa_tl</code>	265, 281, 1050, 1051
<code>\zref@newprop</code> 4, 21, 23, 33, 35, 83, 85, 98			
<code>\zref@refused</code>	1539		
<code>\zref@wrapper@babel</code>	33, 1028		
<code>\textendash</code>	391		
<code>\the</code>	3		
<code>\thechapter</code>	67		
<code>\thepage</code>	6, 95		
<code>\thesection</code>	67		
tl commands:			
<code>\c_empty_tl</code>	1127, 1138, 1140, 1198, 1202, 1206, 1210, 1473, 1478		
<code>\c_novalue_tl</code>	863, 904		
<code>\tl_clear:N</code> 279, 328, 933, 957, 1430, 1431, 1432, 1433, 1434, 1456, 1846, 1847, 1848, 1849, 1887, 2221, 2224, 2252, 2270, 2305, 2446, 2477, 2479			
<code>\tl_gset:Nn</code>	95		
<code>\tl_head:N</code>			
.	1292, 1293, 1296, 1298, 1312, 1314		
<code>\tl_if_empty:NNTF</code>			
.	71, 340, 357, 371, 969, 990, 1013, 1048, 1543, 1713, 2139, 2237, 2254		
<code>\tl_if_empty:nTF</code>			
.	228, 238, 327, 440, 956, 1662, 1678, 1694, 1936, 1967, 1979, 1993, 2223		

U

use commands:

`\use:N` 21

Z

`\zcDeclareLanguage`
10, 226, 2483, 2674, 2854, 3028, 3215

`\zcDeclareLanguageAlias`
. 11, 236, 2484, 2485,
2486, 2487, 2488, 2489, 2490, 2675,
2676, 2677, 2678, 2679, 2680, 2855,
2856, 2857, 2858, 3029, 3030, 3031

`\zcDeclareTranslations`
. 9, 11, 12, 29–31, 927

`\zcpageref` 34, 1066

`\zceref` 8,
 27, 28, 32–36, 43, 45, 1027, 1069, 1070
`\zcRefTypeSetup` 9, 29, 880
`\zcsetup` 8, 24, 27, 28, 878
 zrefcheck commands:
 `\zrefcheck_zceref_beg_label:` .. 1039
 `\zrefcheck_zceref_end_label_-`
 maybe: 1058
 `\zrefcheck_zceref_run_checks_on_-`
 labels:n 1059
 zrefclever internal commands:
 `\l__zrefclever_abbrev_bool`
 704, 708, 2241
 `\l__zrefclever_capitalize_bool` ..
 690, 694, 2229
 `\l__zrefclever_capitalize_first_-`
 bool 691, 700, 2231
 `__zrefclever_counter_reset_by:n`
 .. 5, 18, 19, 39, 41, 43, 48, 50, 52, 57
 `__zrefclever_counter_reset_by_-`
 aux:n 64, 67
 `__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, 726, 731, 737, 741, 767, 803
 `__zrefclever_declare_default_-`
 transl:nnn 31, 940, 971, 992
 `__zrefclever_declare_type_-`
 transl:nnnn 31, 940, 997, 1019
 `\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, 931, 972, 993, 998, 1020
 `\g__zrefclever_fallback_dict_-`
 prop 9, 380, 381, 433
 `__zrefclever_get_default_-`
 transl:nnN 9, 414, 428
 `__zrefclever_get_default_-`
 transl:nnNTF 16, 413, 2439
 `__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, 2444
 `__zrefclever_get_ref:n`
 56, 57, 1611, 1627,
 1639, 1644, 1667, 1681, 1685, 1697,
 1701, 1736, 1756, 1926, 1939, 1946,
 1970, 1982, 1986, 1996, 2000, 2016
 `__zrefclever_get_ref_first:` ...
 56, 61, 1749, 1807, 2066
 `__zrefclever_get_ref_font:nN` . 8,
 15, 28, 64–66, 1572, 1574, 1576, 2455
 `__zrefclever_get_ref_string:nN` .
 .. 8, 9, 15, 28, 64, 65, 1050, 1441,
 1443, 1445, 1554, 1556, 1558, 1560,
 1562, 1564, 1566, 1568, 1570, 2398
 `__zrefclever_get_type_transl:nnnN`
 9, 398, 412
 `__zrefclever_get_type_transl:nnnNTF`
 16, 397, 2264, 2293, 2299, 2433
 `\l__zrefclever_label_a_tl`
 . 42, 1396, 1453, 1473, 1489, 1539,
 1540, 1546, 1598, 1611, 1627, 1644,
 1685, 1701, 1729, 1736, 1864, 1868,
 1878, 1903, 1926, 1947, 1986, 2000
 `\l__zrefclever_label_b_tl`
 42, 1396,
 1456, 1461, 1478, 1491, 1496, 1868
 `\l__zrefclever_label_count_int` ..
 43, 1394,
 1435, 1552, 1591, 1850, 1873, 2010
 `\l__zrefclever_label_enclcnt_a_-`
 tl 1072, 1197, 1199, 1200,
 1221, 1248, 1273, 1292, 1300, 1301
 `\l__zrefclever_label_enclcnt_b_-`
 tl 1072, 1201, 1203, 1204,
 1222, 1252, 1269, 1293, 1302, 1303
 `\l__zrefclever_label_enclval_a_-`
 tl 1072, 1205,
 1207, 1208, 1296, 1304, 1305, 1312
 `\l__zrefclever_label_enclval_b_-`
 tl 1072, 1209,
 1211, 1212, 1298, 1306, 1307, 1314
 `\l__zrefclever_label_type_a_tl` ..
 64, 1072, 1126, 1129,
 1132, 1137, 1146, 1155, 1163, 1168,
 1344, 1372, 1466, 1470, 1503, 1511,
 1517, 1543, 1600, 1880, 2406, 2411,
 2418, 2427, 2435, 2462, 2467, 2474
 `\l__zrefclever_label_type_b_tl` ..
 1072,
 1139, 1147, 1156, 1164, 1169, 1347,
 1375, 1467, 1475, 1504, 1513, 1518
 `__zrefclever_label_type_put_-`
 new_right:n 35, 36, 1088, 1124


```

\l_zrefclever_label_types_seq . .
    . . . . . 36, 1081, 1084, 1128, 1131, 1370
\_zrefclever_labels_in_sequence:nn
    . . . . . 43, 63, 1727, 1867, 2345
\g_zrefclever_languages_prop . .
    . . . . . 11, 225, 230,
    232, 240, 243, 244, 255, 400, 416, 930
\l_zrefclever_last_of_type_bool
    . . . . . 42, 1391, 1487, 1492, 1493,
    1497, 1506, 1521, 1525, 1531, 1581
\l_zrefclever_lastsep_tl . 1411,
    1563, 1626, 1643, 1666, 1684, 1696
\l_zrefclever_link_star_bool . .
    . . . . . 1034, 1064, 2023, 2158, 2321
\l_zrefclever_listsep_tl . . . . .
    . . . 1411, 1561, 1638, 1680, 1925,
    1938, 1945, 1969, 1981, 1985, 1995
\l_zrefclever_load_dict_-
    verbose_bool . . . 251, 288, 297, 306
\g_zrefclever_loaded_dictionaries_-
    seq . . . . . 250, 259, 282
\l_zrefclever_main_language_tl .
    . 24, 725, 732, 738, 742, 746, 759, 795
\_zrefclever_name_default: . . . .
    . . . . . 2012, 2141
\l_zrefclever_name_format_-
    fallback_tl . . . . .
    . . 1402, 2248, 2252, 2254, 2290, 2302
\l_zrefclever_name_format_tl . .
    . . . 1402, 2234, 2235, 2238, 2239,
    2249, 2250, 2261, 2267, 2282, 2296
\l_zrefclever_name_in_link_bool
    . . . . . 58,
    61, 1402, 1764, 2071, 2325, 2341, 2342
\l_zrefclever_namefont_tl 1411,
    1573, 1767, 1795, 2100, 2131, 2146
\l_zrefclever_nameinlink_str . .
    . . . . . 675, 680,
    682, 684, 686, 2323, 2329, 2331, 2335
\l_zrefclever_namesep_tl . . . . .
    . . 1411, 1555, 2103, 2134, 2142, 2149
\l_zrefclever_next_is_same_bool
    . . . . . 43, 63, 1406,
    1861, 1889, 1905, 1911, 2365, 2391
\l_zrefclever_next_maybe_range_-
    bool . . . . .
    . . 43, 63, 1406, 1723, 1733, 1860,
    1885, 1895, 2357, 2364, 2383, 2390
\l_zrefclever_noabbrev_first_-
    bool . . . . . 705, 714, 2245
\_zrefclever_page_format_aux: . .
    . . . . . 90, 94
\g_zrefclever_page_format_tl . .
    . . . . . 6, 89, 95, 98

\l_zrefclever_pairsep_tl . . . . .
    . . . . . 1411, 1559, 1610, 1734
\_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, 785, 1035
\_zrefclever_provide_dictionary_-
    verbose:n . . . . . 13,
    303, 760, 768, 774, 796, 804, 810
\l_zrefclever_range_beg_label_-
    tl . . . . . 43, 1406, 1434,
    1639, 1662, 1668, 1678, 1682, 1694,
    1698, 1849, 1887, 1902, 1936, 1940,
    1967, 1971, 1979, 1983, 1993, 1997
\l_zrefclever_range_count_int . .
    . . . . . 43,
    1406, 1437, 1619, 1653, 1852, 1888,
    1899, 1904, 1910, 1918, 1959, 2005
\l_zrefclever_range_same_count_-
    int . . . . . 43,
    1406, 1438, 1606, 1641, 1654, 1853,
    1890, 1906, 1912, 1943, 1960, 2006
\l_zrefclever_rangesep_tl . . . .
    . . . . . 1411, 1557, 1700, 1735, 1999
\_zrefclever_ref_default: . . . .
    . . . . . 2012, 2063, 2069, 2135, 2214
\l_zrefclever_ref_language_tl . .
    . . . . . 24, 25,
    724, 745, 758, 761, 766, 769, 773,
    775, 785, 794, 797, 802, 805, 809,
    811, 1035, 2265, 2294, 2300, 2434, 2440
\c_zrefclever_ref_options_font_-
    seq . . . . . 10, 15, 171
\c_zrefclever_ref_options_-
    necessarily_not_type_specific_-
    seq . . . . . 15, 171, 333, 888, 962
\c_zrefclever_ref_options_-
    necessarily_type_specific_seq
    . . . . . 171, 364, 1006
\c_zrefclever_ref_options_-
    possibly_type_specific_seq . .
    . . . . . 15, 171, 350, 983
\l_zrefclever_ref_options_prop .
    . . . . 28, 29, 857, 867, 868, 2401, 2458
\c_zrefclever_ref_options_-
    reference_seq . . . . . 171, 859
\c_zrefclever_ref_options_-
    typesetup_seq . . . . . 171, 900
\l_zrefclever_ref_property_tl . .
    . . . . . 20,

```

522, 527, 529, 535, 538, 554, 563,
 1085, 1117, 1464, 2018, 2042, 2056,
 2075, 2112, 2153, 2191, 2207, 2347
 \l_zrefclever_ref_typeset_font_
 tl 818, 820, 1045
 \l_zrefclever_reffont_in_tl 1411,
 1577, 2030, 2054, 2108, 2165, 2203
 \l_zrefclever_reffont_out_tl ...
 1411, 1575,
 2027, 2051, 2105, 2125, 2162, 2200
 \l_zrefclever_refpos_in_tl 1411,
 1571, 2043, 2057, 2113, 2192, 2208
 \l_zrefclever_refpos_out_tl 1411,
 1567, 2046, 2059, 2126, 2195, 2210
 \l_zrefclever_refpre_in_tl 1411,
 1569, 2040, 2055, 2109, 2188, 2204
 \l_zrefclever_refpre_out_tl 1411,
 1565, 2028, 2052, 2106, 2163, 2201
 \l_zrefclever_setup_type_tl ...
 14, 169, 279, 315, 328,
 329, 340, 357, 371, 884, 912, 920,
 933, 957, 958, 969, 990, 999, 1013, 1021
 \l_zrefclever_sort_decided_bool
 ... 1078, 1214, 1215, 1229, 1240,
 1255, 1261, 1276, 1282, 1310, 1322
 _zrefclever_sort_default:nn ...
 36, 1119, 1135
 _zrefclever_sort_default_
 different_types:nn
 21, 35, 40, 1173, 1330
 _zrefclever_sort_default_same_
 type:nn 35, 38, 1171, 1195
 _zrefclever_sort_labels:
 35, 36, 41, 1043, 1082
 _zrefclever_sort_page:nn
 42, 1118, 1382
 \l_zrefclever_sort_prior_a_int .
 1079,
 1332, 1338, 1339, 1345, 1355, 1363
 \l_zrefclever_sort_prior_b_int .
 1079,
 1333, 1340, 1341, 1348, 1356, 1364
 \l_zrefclever_tlastsep_tl
 1411, 1446, 1838
 \l_zrefclever_tlistsep_tl
 1411, 1444, 1816
 \l_zrefclever_tpairsep_tl
 1411, 1442, 1832
 \l_zrefclever_type<type>_
 options_prop 29
 \l_zrefclever_type_count_int ...
 43, 61, 1394, 1436, 1813,
 1815, 1824, 1851, 2232, 2244, 2338
 \l_zrefclever_type_first_label_
 tl 43, 58, 1396, 1432, 1597, 1715,
 1724, 1728, 1756, 1772, 1776, 1781,
 1787, 1847, 1877, 2068, 2074, 2081,
 2084, 2089, 2095, 2111, 2152, 2170,
 2173, 2178, 2184, 2190, 2206, 2220
 \l_zrefclever_type_first_label_
 type_tl 43, 61, 1396, 1433, 1599,
 1719, 1848, 1879, 2223, 2259, 2266,
 2272, 2280, 2288, 2295, 2301, 2308
 _zrefclever_type_name_setup: ..
 8, 9, 58, 1744, 2218
 \l_zrefclever_type_name_tl
 58, 61,
1402, 1790, 1796, 2101, 2132, 2139,
 2147, 2221, 2224, 2262, 2268, 2270,
 2283, 2291, 2297, 2303, 2305, 2322
 \l_zrefclever_typeset_compress_
 bool 618, 621, 1862
 \l_zrefclever_typeset_labels_
 seq 42, 1391, 1428, 1452, 1454, 1460
 \l_zrefclever_typeset_last_bool
 42, 1391,
 1449, 1450, 1457, 1486, 1821, 2337
 \l_zrefclever_typeset_name_bool
 569, 576, 581, 586, 1746, 1760
 \l_zrefclever_typeset_queue_
 curr_tl 43,
56, 61, 1396, 1431, 1608, 1624,
 1633, 1664, 1675, 1691, 1713,
 1731, 1748, 1755, 1762, 1806, 1828,
 1833, 1839, 1845, 1846, 1923, 1934,
 1965, 1977, 1991, 2237, 2332, 2336
 \l_zrefclever_typeset_queue_
 prev_tl . 43, 1396, 1430, 1817, 1844
 \l_zrefclever_typeset_range_
 bool 627, 630, 1042, 1711
 \l_zrefclever_typeset_ref_bool .
 568, 575, 580, 585, 1746, 1753
 _zrefclever_typeset_refs:
 42-44, 1046, 1426
 _zrefclever_typeset_refs_last_
 of_type: . 48, 56, 58, 61, 1583, 1588
 _zrefclever_typeset_refs_not_
 last_of_type:
 43, 48, 56, 63, 1585, 1856
 \l_zrefclever_typeset_sort_bool
 594, 597, 1041
 \l_zrefclever_typeset_sort_seq
 21, 40, 603, 608, 609, 615, 1334
 \l_zrefclever_use_hyperref_bool
 634, 641,
 646, 651, 661, 667, 2022, 2157, 2320

\l_zrefclever_warn_hyperref_-	\l_zrefclever_zcref_note_tl ...
bool 635, 642, 647, 652, 665 821, 824, 1048, 1052
_zrefclever_zcref:nnn .. 1028, 1029	\l_zrefclever_zcref_with_check_-
_zrefclever_zcref:nnnn 33, 35, 1029	bool 828, 843, 1038, 1056
\l_zrefclever_zcref_labels_seq .	\l_zrefclever_zrefcheck_-
..... 35,	available_bool
36, 1033, 1060, 1064, 1087, 1090, 1429 827, 838, 849, 1037, 1055