

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-13

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	Reference format	8
4.3	Languages	10
4.4	Dictionaries	11
4.5	Options	14
5	Configuration	25
5.1	\zcsetup	25
5.2	\zcRefTypeSetup	26
5.3	\zcDeclareTranslations	27
6	User interface	29
6.1	\zceref	29
6.2	\zcpageref	31
7	Sorting	31
8	Typesetting	39
9	Special handling	59
9.1	\appendix	60
9.2	\newtheorem	60
9.3	enumitem package	60

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

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

10	Dictionaries	60
10.1	English	60
10.2	German	64
10.3	French	67
10.4	Portuguese	70
10.5	Spanish	74
Index		77

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

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

Identify the package.

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

2 Dependencies

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

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

3 zref setup

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

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

But the reference itself, stored by `zref-base` in the `default` property, is somewhat a disputed real estate. In particular, the use of `\labelformat` (previously from `varioref`, now in the kernel) will include there the reference “prefix” and complicate the job we are trying to do here. Hence, we isolate `\the\counter` and store it “clean” in `zc@thecnt` for reserved use. Based on the definition of `\@currentlabel` done inside `\refstepcounter` in ‘texdoc source2e’, section ‘ltxref.dtx’. We just drop the `\p@...` prefix.

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

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

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

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

```
33 \zref@newprop { zc@cntval } [0] { \int_use:c { c@ \@currentcounter } }
34 \zref@addprop \ZREF@mainlist { zc@cntval }
35 \zref@newprop* { zc@pgval } [0] { \int_use:c { c@page } }
36 \zref@addprop \ZREF@mainlist { zc@pgval }
```

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

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

Furthermore, even if it is true that most of the definitions of counters, and hence of their reset behavior, is likely to be defined in the preamble, this is not necessarily true. Users can create counters, newtheorems mid-document, and alter their reset behavior along the way. Was that not the case, we could just store the desired information at `\begindocument` in a variable and retrieve it when needed. But since it is, we need to store the information with the label, with the values as current when the label is set.

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of `\newcounter`, `\@addtoreset`, `\counterwithin` and related infrastructure). The canonical optional argument of `\newcounter` establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “supercounter” etc.). This information is a little trickier to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\cl@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see section ‘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 $\langle counter \rangle$.

```

    \_zrefclever_counter_reset_by:n {\counter}

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

```

```

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

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

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

```

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

```

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

```

```

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

```

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

4 Plumbing

4.1 Messages

```

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

```

```

142 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
143 {
144   Option~'hyperref'~only~available~in~the~preamble. \iow_newline:
145   Use~the~starred~version~of~'\iow_char:N\zcref'~instead.
146 }
147 \msg_new:nnn { zref-clever } { missing-hyperref }
148 { Missing~'hyperref'~package. \iow_newline: Setting~'hyperref=false'. }
149 \msg_new:nnn { zref-check } { check-document-only }
150 { Option~'check'~only~available~in~the~document. }
151 \msg_new:nnn { zref-clever } { missing-zref-check }
152 {
153   Option~'check'~requested~\msg_line_context:..~
154   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
155 }
156 \msg_new:nnn { zref-clever } { counters-not-nested }
157 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:.. }
158 \msg_new:nnn { zref-clever } { missing-type }
159 { Reference~type~undefined~for~label~'#1'~\msg_line_context:.. }
160 \msg_new:nnn { zref-clever } { missing-name }
161 { Name~undefined~for~type~'#1'~\msg_line_context:.. }
162 \msg_new:nnn { zref-clever } { 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). These precedence rules are handled / enforced in `__zrefclever_get_ref_string:nN` and `__zrefclever_get_ref_font:nN`, which are the basic functions to retrieve proper values for reference format settings.

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

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

The package itself places the default setup for reference formatting at low precedence levels, and the users can easily and conveniently override them as desired. Indeed, I expect most of the users' needs to be normally achievable with the general options and type-specific options, since references will normally be typeset in a single language (the document's main language) and, hence, multiple translations don't need to be provided.

```
\l__zrefclever_setup_type_tl Store "current" type and language in different places for option and translation
\l__zrefclever_dict_language_tl 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.)
```

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.

```
f_options_necessarily_not_type_specific_seq
ever_ref_options_possibly_type_specific_seq
r_ref_options_necessarily_type_specific_seq
\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

```

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

```

```

246 % {<alias>}{<base language>}
247 \NewDocumentCommand \zcDeclareLanguageAlias { m m }
248 {
249   \prop_if_in:NnTF \g__zrefclever_language_aliases_prop {#2}
250   {
251     \exp_args:NNnx \prop_gput:Nnn \g__zrefclever_language_aliases_prop {#1}
252     { \prop_item:Nn \g__zrefclever_language_aliases_prop {#2} }
253   }
254   { \msg_warning:nnn { zref-clever } { unknown-language-alias } {#2} }
255 }
256 \onlypreamble \zcDeclareLanguageAlias
257

```

4.4 Dictionaries

```

258 \seq_new:N \g__zrefclever_loaded_dictionaries_seq
259 \bool_new:N \l__zrefclever_load_dict_verbose_bool
260
261 % {<language>}
262 \cs_new_protected:Npn \__zrefclever_provide_dictionary:n #1
263 {
264   \group_begin:
265   \prop_get:NnNTF \g__zrefclever_language_aliases_prop {#1}
266   \l__zrefclever_dict_language_tl
267   {
268     \seq_if_in:NVF
269     \g__zrefclever_loaded_dictionaries_seq
270     \l__zrefclever_dict_language_tl
271     {
272       \exp_args:Nx \file_get:nnNTF
273       { zref-clever- \l__zrefclever_dict_language_tl .dict }
274       { \ExplSyntaxOn }
275       \l_tmpa_tl
276       {
277         \prop_if_exist:cF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _pro
278         { \prop_new:c { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop
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   \bool_if:NT \l__zrefclever_load_dict_verbose_bool
297

```

```

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

```

```

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

```

```

406         \prop_get:cnNTF { g__zrefclever_dict_ \l__zrefclever_dict_language_tl _prop }
407         { default- #2 } #3
408         { \prg_return_true: }
409         { \prg_return_false: }
410     }
411     { \prg_return_false: }
412 }
413 \prg_generate_conditional_variant:Nnn \__zrefclever_get_default_transl:nnN { xnN } { F }
414
415 % {<key>}<tl var to set>
416 \prg_new_protected_conditional:Npmn \__zrefclever_get_fallback_transl:nn #1#2 { F }
417 {
418     \prop_get:NnNTF \g__zrefclever_fallback_dict_prop
419     { #1 } #2
420     { \prg_return_true: }
421     { \prg_return_false: }
422 }

```

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”.

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

```

423 \prop_new:N \g__zrefclever_fallback_dict_prop
424 \prop_gset_from_keyval:Nn \g__zrefclever_fallback_dict_prop
425 {
426     tpairsep = {,~} ,
427     tlistsep = {,~} ,
428     tlastsep = {,~} ,
429     notesep  = {~} ,
430     namesep  = {\nobreakspace} ,
431     pairsep  = {,~} ,
432     listsep  = {,~} ,
433     lastsep  = {,~} ,
434     rangesep = {\textendash} ,
435     refpre   = {} ,
436     refpos   = {} ,
437     refpre-in = {} ,
438     refpos-in = {} ,
439 }

```

4.5 Options

Auxiliary

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

`__zrefclever_prop_put_non_empty:Nnn $\langle property list \rangle$ { $\langle key \rangle$ } { $\langle value \rangle$ }`

```

440 \cs_new_protected:Npn \__zrefclever_prop_put_non_empty:Nnn #1#2#3
441 {
442   \tl_if_empty:nTF {#3}
443     { \prop_remove:Nn #1 {#2} }
444     { \prop_put:Nnn #1 {#2} {#3} }
445 }

```

(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`.

```

446 \prop_new:N \l__zrefclever_counter_type_prop
447 \keys_define:nn { zref-clever / label }
448 {
449   countertype .code:n =
450   {
451     \keyval_parse:nnn
452     {
453       \msg_warning:nnnn { zref-clever }
454       { key-requires-value } { countertype }
455     }
456     {
457       \__zrefclever_prop_put_non_empty:Nnn
458       \l__zrefclever_counter_type_prop
459     }
460     {#1}
461   } ,
462   countertype .value_required:n = true ,
463   countertype .initial:n =
464   {
465     subsection      = section ,
466     subsubsection    = section ,
467     subparagraph     = paragraph ,
468     enumi            = item ,
469     enumii           = item ,
470     enumiii          = item ,
471     enumiv           = item ,
472   } ,
473 }

```

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_resettters_seq` with the `counterresetby` option.

```

474 \seq_new:N \l__zrefclever_counter_resettters_seq
475 \keys_define:nn { zref-clever / label }
476 {
477   counterresettters .code:n =
478   {
479     \clist_map_inline:nn {#1}
480     {
481       \seq_if_in:NnF \l__zrefclever_counter_resettters_seq {##1}
482       {
483         \seq_put_right:Nn
484         \l__zrefclever_counter_resettters_seq {##1}
485       }
486     }
487   } ,
488   counterresettters .initial:n =
489   {
490     part ,
491     chapter ,
492     section ,
493     subsection ,
494     subsubsection ,
495     paragraph ,
496     subparagraph ,
497   },
498   typesort .value_required:n = true ,
499 }

```

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`.

```

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

```



```

517     counterresetby .initial:n =
518     {

```

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.

```

519         enumii = enumi ,
520         enumiii = enumii ,
521         enumiv = enumiii ,
522     } ,
523 }

```

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.

```

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

```

```

552     {
553       \keys_define:nn { zref-clever / reference }
554       {
555         ref / title .code:n =
556         { \tl_set:Nn \l__zrefclever_ref_property_tl { title } }
557       }
558     }
559     {
560       \keys_define:nn { zref-clever / reference }
561       {
562         ref / title .code:n =
563         {
564           \msg_warning:nn { zref-clever } { missing-zref-titleref }
565           \tl_set:Nn \l__zrefclever_ref_property_tl { zc@theCNT }
566         }
567       }
568     }
569   }

```

typeset option

```

570 \bool_new:N \l__zrefclever_typeset_ref_bool
571 \bool_new:N \l__zrefclever_typeset_name_bool
572 \keys_define:nn { zref-clever / reference }
573 {
574   typeset .choice: ,
575   typeset / both .code:n =
576   {
577     \bool_set_true:N \l__zrefclever_typeset_ref_bool
578     \bool_set_true:N \l__zrefclever_typeset_name_bool
579   } ,
580   typeset / ref .code:n =
581   {
582     \bool_set_true:N \l__zrefclever_typeset_ref_bool
583     \bool_set_false:N \l__zrefclever_typeset_name_bool
584   } ,
585   typeset / name .code:n =
586   {
587     \bool_set_false:N \l__zrefclever_typeset_ref_bool
588     \bool_set_true:N \l__zrefclever_typeset_name_bool
589   } ,
590   typeset .initial:n = both ,
591   typeset .value_required:n = true ,
592
593   noname .meta:n = { typeset = ref },
594   noname .value_forbidden:n = true ,
595 }

```

sort option

```

596 \bool_new:N \l__zrefclever_typeset_sort_bool
597 \keys_define:nn { zref-clever / reference }
598 {
599   sort .bool_set:N = \l__zrefclever_typeset_sort_bool ,
600   sort .initial:n = true ,

```

```

601     sort .default:n = true ,
602     nosort .meta:n = { sort = false },
603     nosort .value_forbidden:n = true ,
604 }

```

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.

```

605 \seq_new:N \l__zrefclever_typesort_seq
606 \keys_define:nn { zref-clever / reference }
607 {
608     typesort .code:n =
609     {
610         \seq_set_from_clist:Nn \l__zrefclever_typesort_seq {#1}
611         \seq_reverse:N \l__zrefclever_typesort_seq
612     } ,
613     typesort .initial:n =
614     { part , chapter , section , paragraph },
615     typesort .value_required:n = true ,
616     notypesort .code:n =
617     { \seq_clear:N \l__zrefclever_typesort_seq } ,
618     notypesort .value_forbidden:n = true ,
619 }

```

comp option

```

620 \bool_new:N \l__zrefclever_typeset_compress_bool
621 \keys_define:nn { zref-clever / reference }
622 {
623     comp .bool_set:N = \l__zrefclever_typeset_compress_bool ,
624     comp .initial:n = true ,
625     comp .default:n = true ,
626     nocomp .meta:n = { comp = false },
627     nocomp .value_forbidden:n = true ,
628 }

```

range option

```

629 \bool_new:N \l__zrefclever_typeset_range_bool
630 \keys_define:nn { zref-clever / reference }
631 {
632     range .bool_set:N = \l__zrefclever_typeset_range_bool ,
633     range .initial:n = false ,
634     range .default:n = true ,
635 }

```

hyperref option

```

636 \bool_new:N \l__zrefclever_use_hyperref_bool
637 \bool_new:N \l__zrefclever_warn_hyperref_bool
638 \keys_define:nn { zref-clever / reference }
639 {
640     hyperref .choice: ,

```

```

641 hyperref / auto .code:n =
642 {
643   \bool_set_true:N \l__zrefclever_use_hyperref_bool
644   \bool_set_false:N \l__zrefclever_warn_hyperref_bool
645 } ,
646 hyperref / true .code:n =
647 {
648   \bool_set_true:N \l__zrefclever_use_hyperref_bool
649   \bool_set_true:N \l__zrefclever_warn_hyperref_bool
650 } ,
651 hyperref / false .code:n =
652 {
653   \bool_set_false:N \l__zrefclever_use_hyperref_bool
654   \bool_set_false:N \l__zrefclever_warn_hyperref_bool
655 } ,
656 hyperref .initial:n = auto ,
657 hyperref .default:n = auto
658 }
659 \AddToHook { begindocument }
660 {
661   \@ifpackageloaded { hyperref }
662   {
663     \bool_if:NT \l__zrefclever_use_hyperref_bool
664     { \RequirePackage { zref-hyperref } }
665   }
666   {
667     \bool_if:NT \l__zrefclever_warn_hyperref_bool
668     { \msg_warning:nn { zref-clever } { missing-hyperref } }
669     \bool_set_false:N \l__zrefclever_use_hyperref_bool
670   }
671   \keys_define:nn { zref-clever / reference }
672   {
673     hyperref .code:n =
674     { \msg_warning:nn { zref-clever } { hyperref-preamble-only } }
675   }
676 }

```

nameinlink option

```

677 \str_new:N \l__zrefclever_nameinlink_str
678 \keys_define:nn { zref-clever / reference }
679 {
680   nameinlink .choice: ,
681   nameinlink / true .code:n =
682   { \str_set:Nn \l__zrefclever_nameinlink_str { true } } ,
683   nameinlink / false .code:n =
684   { \str_set:Nn \l__zrefclever_nameinlink_str { false } } ,
685   nameinlink / single .code:n =
686   { \str_set:Nn \l__zrefclever_nameinlink_str { single } } ,
687   nameinlink / tsingle .code:n =
688   { \str_set:Nn \l__zrefclever_nameinlink_str { tsingle } } ,
689   nameinlink .initial:n = tsingle ,
690   nameinlink .default:n = true ,
691 }

```

cap and capfirst options

```
692 \bool_new:N \l__zrefclever_capitalize_bool
693 \bool_new:N \l__zrefclever_capitalize_first_bool
694 \keys_define:nn { zref-clever / reference }
695 {
696   cap .bool_set:N = \l__zrefclever_capitalize_bool ,
697   cap .initial:n = false ,
698   cap .default:n = true ,
699   nocap .meta:n = { cap = false },
700   nocap .value_forbidden:n = true ,
701
702   capfirst .bool_set:N = \l__zrefclever_capitalize_first_bool ,
703   capfirst .initial:n = false ,
704   capfirst .default:n = true ,
705
706   C .meta:n =
707     { capfirst = true , noabbrevfirst = true },
708   C .value_forbidden:n = true ,
709 }
```

abbrev and noabbrevfirst options

```
710 \bool_new:N \l__zrefclever_abbrev_bool
711 \bool_new:N \l__zrefclever_noabbrev_first_bool
712 \keys_define:nn { zref-clever / reference }
713 {
714   abbrev .bool_set:N = \l__zrefclever_abbrev_bool ,
715   abbrev .initial:n = false ,
716   abbrev .default:n = true ,
717   noabbrev .meta:n = { abbrev = false },
718   noabbrev .value_forbidden:n = true ,
719
720   noabbrevfirst .bool_set:N = \l__zrefclever_noabbrev_first_bool ,
721   noabbrevfirst .initial:n = false ,
722   noabbrevfirst .default:n = true ,
723 }
```

lang option

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

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

after any options set in the preamble. This hook redefines the `lang` option for immediate execution in the document body, and ensures the `main` language’s dictionary gets loaded, if it hadn’t been already.

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

```

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, so that we are able to distinguish when the user actually gave the option, in which case, the dictionary loading is done verbosely.

```

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

```

```

762         { current }
763         {
764             \tl_set:Nn \l__zrefclever_ref_language_tl
765             { \l__zrefclever_current_language_tl }
766             \__zrefclever_provide_dictionary_verbos:x
767             { \l__zrefclever_ref_language_tl }
768         }
769     }
770 }
771 {
772     \tl_set:Nn \l__zrefclever_ref_language_tl {#1}
773     \__zrefclever_provide_dictionary_verbos:x
774     { \l__zrefclever_ref_language_tl }
775 }
776 }
777 } ,
778 lang .value_required:n = true ,
779 }
780 \AddToHook { begindocument / before }
781 {
782     \AddToHook { begindocument }
783     {

```

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.

```

784     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

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

```

```

811         }
812     } ,
813     lang .value_required:n = true ,
814 }
815 }
816 }

```

font option

```

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

```

note option

```

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

```

check option

Integration with zref-check.

```

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

```


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.

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

Package options

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

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

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

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

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```
877 \NewDocumentCommand \zcsetup { m }
878 { \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}
879 \NewDocumentCommand \zcRefTypeSetup { m m }
880 {
881   \prop_if_exist:cF { l__zrefclever_type_ #1 _options_prop }
882   { \prop_new:c { l__zrefclever_type_ #1 _options_prop } }
883   \tl_set:Nn \l__zrefclever_setup_type_tl {#1}
884   \keys_set:nn { zref-clever / typesetup } {#2}
885 }
```

(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>.

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

```

901 \keys_define:nn { zref-clever / typesetup }
902 {
903   #1 .default:V = \c_novalue_tl ,
904   #1 .code:n =
905   {
906     \tl_if_novalue:nTF {##1}
907     {
908       \prop_remove:cn
909       {
910         l__zrefclever_type_
911         \l__zrefclever_setup_type_tl _options_prop
912       }
913       {#1}
914     }
915     {
916       \prop_put:cnn
917       {
918         l__zrefclever_type_
919         \l__zrefclever_setup_type_tl _options_prop
920       }
921       {#1} {##1}
922     }
923   } ,
924 }
925 }

```

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

```

(End definition for \zcDeclareTranslations.)

```

939 \keys_define:nn { zref-clever / translations }

```

```

940 {
941   type .code:n =
942   {
943     \tl_if_empty:nTF {#1}
944     { \tl_clear:N \l__zrefclever_setup_type_tl }
945     { \tl_set:Nn \l__zrefclever_setup_type_tl {#1} }
946   } ,
947 }
948 % {<language>}{<type>}{<key>}{<translation>}
949 \cs_new_protected:Npn \__zrefclever_declare_type_transl:nnnn #1#2#3#4
950 {
951   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
952   { type- #2 - #3 } {#4}
953 }
954 \cs_generate_variant:Nn \__zrefclever_declare_type_transl:nnnn { VVnn }
955
956 % {<language>}{<key>}{<translation>}
957 \cs_new_protected:Npn \__zrefclever_declare_default_transl:nnn #1#2#3
958 {
959   \prop_gput:cnn { g__zrefclever_dict_ #1 _prop }
960   { default- #2 } {#3}
961 }
962 \cs_generate_variant:Nn \__zrefclever_declare_default_transl:nnn { Vnn }
963 \seq_map_inline:Nn
964   \c__zrefclever_ref_options_necessarily_not_type_specific_seq
965   {
966     \keys_define:nn { zref-clever / translations }
967     {
968       #1 .value_required:n = true ,
969       #1 .code:n =
970       {
971         \tl_if_empty:NTF \l__zrefclever_setup_type_tl
972         {
973           \__zrefclever_declare_default_transl:Vnn
974           \l__zrefclever_dict_language_tl
975           {#1} {##1}
976         }
977         {
978           \msg_warning:nnn { zref-clever }
979           { option-not-type-specific } {#1}
980         }
981       } ,
982     }
983   }
984 \seq_map_inline:Nn
985   \c__zrefclever_ref_options_possibly_type_specific_seq
986   {
987     \keys_define:nn { zref-clever / translations }
988     {
989       #1 .value_required:n = true ,
990       #1 .code:n =
991       {
992         \tl_if_empty:NTF \l__zrefclever_setup_type_tl

```

```

993         {
994             \__zrefclever_declare_default_transl:Vnn
995             \l__zrefclever_dict_language_tl
996             {#1} {##1}
997         }
998         {
999             \__zrefclever_declare_type_transl:VVnn
1000             \l__zrefclever_dict_language_tl
1001             \l__zrefclever_setup_type_tl
1002             {#1} {##1}
1003         }
1004     } ,
1005 }
1006
1007 \seq_map_inline:Nn
1008 \c__zrefclever_ref_options_necessarily_type_specific_seq
1009 {
1010     \keys_define:nn { zref-clever / translations }
1011     {
1012         #1 .value_required:n = true ,
1013         #1 .code:n =
1014         {
1015             \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1016             {
1017                 \msg_warning:nnn { zref-clever }
1018                 { option-only-type-specific } {#1}
1019             }
1020             {
1021                 \__zrefclever_declare_type_transl:VVnn
1022                 \l__zrefclever_dict_language_tl
1023                 \l__zrefclever_setup_type_tl
1024                 {#1} {##1}
1025             }
1026         } ,
1027     }
1028 }

```

6 User interface

6.1 \zcref

```

\zcref          \zcref<*>[<options>]{<labels>}
1029 \NewDocumentCommand \zcref { s O { } m }
1030 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for \zcref.)

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

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

```

1031 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1032 {
1033   \group_begin:

```

Set options.

```

1034   \keys_set:nn { zref-clever / reference } {#3}

```

Store arguments values.

```

1035   \seq_set_from_clist:Nn \l__zrefclever_zcref_labels_seq {#1}
1036   \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.

```

1037   \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

Integration with zref-check.

```

1038   \bool_lazy_and:nnT
1039   { \l__zrefclever_zrefcheck_available_bool }
1040   { \l__zrefclever_zcref_with_check_bool }
1041   { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

```

1042   \bool_lazy_or:nnT
1043   { \l__zrefclever_typeset_sort_bool }
1044   { \l__zrefclever_typeset_range_bool }
1045   { \__zrefclever_sort_labels: }

```

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

```

1046   \group_begin:
1047   \l__zrefclever_ref_typeset_font_tl
1048   \__zrefclever_typeset_refs:
1049   \group_end:

```

Typeset note.

```

1050   \__zrefclever_get_ref_string:nN {notesep} \l__zrefclever_notesep_tl
1051   \l__zrefclever_notesep_tl
1052   \l__zrefclever_zcref_note_tl

```

Integration with zref-check.

```

1053   \bool_lazy_and:nnT
1054   { \l__zrefclever_zrefcheck_available_bool }
1055   { \l__zrefclever_zcref_with_check_bool }
1056   {
1057     \zrefcheck_zcref_end_label_maybe:
1058     \zrefcheck_zcref_run_checks_on_labels:n
1059     { \l__zrefclever_zcref_labels_seq }
1060   }
1061   \group_end:
1062 }

```

(End definition for `__zrefclever_zcref:nnnn`.)

```

\l_zrefclever_zcref_labels_seq
\l_zrefclever_link_star_bool
1063 \seq_new:N \l__zrefclever_zcref_labels_seq
1064 \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 \zcpageref{<*>}[<options>]{<labels>}
1065 \NewDocumentCommand \zcpageref { s O { } m }
1066 {
1067   \IfBooleanTF {#1}
1068   { \zcref*[#2, ref = page] {#3} }
1069   { \zcref [ #2, ref = page] {#3} }
1070 }

(End definition for \zcpageref.)

```

7 Sorting

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

```

\l__zrefclever_label_a_tl 1071 \tl_new:N \l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl 1072 \tl_new:N \l__zrefclever_label_b_tl
\l_zrefclever_label_type_a_tl 1073 \tl_new:N \l__zrefclever_label_type_a_tl
\l_zrefclever_label_type_b_tl 1074 \tl_new:N \l__zrefclever_label_type_b_tl
\l_zrefclever_label_enclcnt_a_tl 1075 \tl_new:N \l__zrefclever_label_enclcnt_a_tl
\l_zrefclever_label_enclcnt_b_tl 1076 \tl_new:N \l__zrefclever_label_enclcnt_b_tl
\l_zrefclever_label_enclval_a_tl 1077 \tl_new:N \l__zrefclever_label_enclval_a_tl
\l_zrefclever_label_enclval_b_tl 1078 \tl_new:N \l__zrefclever_label_enclval_b_tl
\l_zrefclever_label_enclhead_a_tl 1079 \tl_new:N \l__zrefclever_label_enclhead_a_tl
\l_zrefclever_label_enclhead_b_tl 1080 \tl_new:N \l__zrefclever_label_enclhead_b_tl

(End definition for \l__zrefclever_label_a_tl and others.)

1081 \int_new:N \l__zrefclever_sort_prior_a_int
1082 \int_new:N \l__zrefclever_sort_prior_b_int

```

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

```

1083 \bool_new:N \l__zrefclever_sort_decided_bool

(End definition for \l__zrefclever_sort_decided_bool.)

```

Variant not provided by the kernel.

```

1084 \cs_generate_variant:Nn \tl_reverse_items:n { V }

```

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

```

\__zrefclever_label_type_put_new_right:n {<label>}

```

```

1085 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1086 {
1087   \tl_set:Nx \l__zrefclever_label_type_a_tl
1088   { \zref@extractdefault {#1} { zc@type } { \c_empty_tl } }
1089   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1090   {
1091     \seq_if_in:NVF
1092     \l__zrefclever_label_types_seq
1093     \l__zrefclever_label_type_a_tl
1094     {
1095       \seq_put_right:NV \l__zrefclever_label_types_seq
1096       \l__zrefclever_label_type_a_tl
1097     }
1098   }
1099 }

```

(End definition for __zrefclever_label_type_put_new_right:n.)

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

```

1100 \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.

```

1101 \cs_new_protected:Npn \__zrefclever_sort_labels:
1102 {

```

Store label types sequence.

```

1103   \seq_clear:N \l__zrefclever_label_types_seq
1104   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1105   {
1106     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1107     \__zrefclever_label_type_put_new_right:n
1108   }

```

Sort.

```

1109   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1110   {
1111     \zref@ifrefundefined {##1}
1112     {
1113       \zref@ifrefundefined {##2}
1114       {
1115         % Neither label is defined.
1116         \sort_return_same:
1117       }
1118       {
1119         % The second label is defined, but the first isn't, leave the
1120         % undefined first (to be more visible).
1121         \sort_return_same:

```



```

1122     }
1123   }
1124   {
1125     \zref@ifrefundefined {##2}
1126     {
1127       % The first label is defined, but the second isn't, bring the
1128       % second forward.
1129       \sort_return_swapped:
1130     }
1131     {
1132       % The interesting case: both labels are defined. The
1133       % reference to the "default" property/counter or to the page
1134       % are quite different from our perspective, they rely on
1135       % different fields and even use different information for
1136       % sorting, so we branch them here to specialized functions.
1137       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1138         { \__zrefclever_sort_page:nn {##1} {##2} }
1139         { \__zrefclever_sort_default:nn {##1} {##2} }
1140     }
1141   }
1142 }
1143 }

```

(End definition for __zrefclever_sort_labels:.)

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

```

\__zrefclever_sort_default:nn {\label a} {\label b}
1144 \cs_new_protected:Npn \__zrefclever_sort_default:nn #1#2
1145 {
1146   \tl_set:Nx \l__zrefclever_label_type_a_tl
1147     { \zref@extractdefault {#1} {zc@type} {\c_empty_tl} }
1148   \tl_set:Nx \l__zrefclever_label_type_b_tl
1149     { \zref@extractdefault {#2} {zc@type} {\c_empty_tl} }
1150
1151   \bool_if:nTF
1152     {
1153       % The second label has a type, but the first doesn't, leave the
1154       % undefined first (to be more visible).
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_same: }
1159     {
1160       \bool_if:nTF
1161         {
1162           % The first label has a type, but the second doesn't, bring the
1163           % second forward.
1164           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1165           \tl_if_empty_p:N \l__zrefclever_label_type_b_tl

```

```

1166     }
1167     { \sort_return_swapped: }
1168     {
1169         \bool_if:nTF
1170         {
1171             % The interesting case: both labels have a type...
1172             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1173             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1174         }
1175         {
1176             % Here we send this to a couple of auxiliary functions for no
1177             % other reason than to keep this long function a little less
1178             % unreadable.
1179             \tl_if_eq:NNTF
1180             \l__zrefclever_label_type_a_tl
1181             \l__zrefclever_label_type_b_tl
1182             {
1183                 % ...and it's the same type.
1184                 \__zrefclever_sort_default_same_type:nn {#1} {#2}
1185             }
1186             {
1187                 % ...and they are different types.
1188                 \__zrefclever_sort_default_different_types:nn {#1} {#2}
1189             }
1190         }
1191         {
1192             % Neither of the labels has a type. We can't do much of
1193             % meaningful here, but if it's the same counter, compare it.
1194             \exp_args:Nxx \tl_if_eq:nnTF
1195             { \zref@extractdefault {#1} { counter } { } }
1196             { \zref@extractdefault {#2} { counter } { } }
1197             {
1198                 \int_compare:nNnTF
1199                 { \zref@extractdefault {#1} { zc@cntval } {-1} }
1200                 >
1201                 { \zref@extractdefault {#2} { zc@cntval } {-1} }
1202                 { \sort_return_swapped: }
1203                 { \sort_return_same: }
1204             }
1205             { \sort_return_same: }
1206         }
1207     }
1208 }
1209 }

```

(End definition for __zrefclever_sort_default:nn.)

__zrefclever_sort_default_same_type:nn

```

1210 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1211 {
1212     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1213     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1214     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1215     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }

```

```

1216 \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1217 { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1218 \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1219 { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1220 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1221 { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1222 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1223 { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1224 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1225 { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1226 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1227 { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1228
1229 \bool_set_false:N \l__zrefclever_sort_decided_bool
1230 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1231 {
1232   \tl_set:Nx \l__zrefclever_label_enclhead_a_tl
1233   { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1234   \tl_set:Nx \l__zrefclever_label_enclhead_b_tl
1235   { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1236
1237   \bool_if:nTF
1238   {
1239     % Both are empty, meaning: neither labels have any (further)
1240     % ‘‘enclosing counters’’ (left).
1241     \tl_if_empty_p:V \l__zrefclever_label_enclhead_a_tl &&
1242     \tl_if_empty_p:V \l__zrefclever_label_enclhead_b_tl
1243   }
1244   {
1245     \exp_args:Nxx \tl_if_eq:nnTF
1246     { \zref@extractdefault {#1} { counter } { } }
1247     { \zref@extractdefault {#2} { counter } { } }
1248     {
1249       \bool_set_true:N \l__zrefclever_sort_decided_bool
1250       \int_compare:nNnTF
1251       { \zref@extractdefault {#1} { zc@cntval } {-1} }
1252       >
1253       { \zref@extractdefault {#2} { zc@cntval } {-1} }
1254       { \sort_return_swapped: }
1255       { \sort_return_same: }
1256     }
1257   }
1258   \msg_warning:nnnn { zref-clever }
1259   { counters-not-nested } {#1} {#2}
1260   \bool_set_true:N \l__zrefclever_sort_decided_bool
1261   \sort_return_same:
1262 }
1263 }
1264 {
1265   \bool_if:nTF
1266   {
1267     % ‘a’ is empty (and ‘b’ is not), meaning: ‘b’ is (possibly)
1268     % nested in ‘a’.
1269     \tl_if_empty_p:V \l__zrefclever_label_enclhead_a_tl

```

```

1270 }
1271 {
1272   \tl_set:Nx \l_tmpa_tl
1273     { {\zref@extractdefault {#1} { counter } { }} }
1274   \exp_args:NNx \tl_if_in:NnTF
1275     \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1276     {
1277       \bool_set_true:N \l__zrefclever_sort_decided_bool
1278       \sort_return_same:
1279     }
1280   {
1281     \msg_warning:nnnn { zref-clever }
1282     { counters-not-nested } {#1} {#2}
1283     \bool_set_true:N \l__zrefclever_sort_decided_bool
1284     \sort_return_same:
1285   }
1286 }
1287 {
1288   \bool_if:nTF
1289     {
1290       % 'b' is empty (and 'a' is not), meaning: 'a' is
1291       % (possibly) nested in 'b'.
1292       \tl_if_empty_p:V \l__zrefclever_label_enclhead_b_tl
1293     }
1294     {
1295       \tl_set:Nx \l_tmpb_tl
1296         { {\zref@extractdefault {#2} { counter } { }} }
1297       \exp_args:NNx \tl_if_in:NnTF
1298         \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1299         {
1300           \bool_set_true:N \l__zrefclever_sort_decided_bool
1301           \sort_return_swapped:
1302         }
1303       {
1304         \msg_warning:nnnn { zref-clever }
1305         { counters-not-nested } {#1} {#2}
1306         \bool_set_true:N \l__zrefclever_sort_decided_bool
1307         \sort_return_same:
1308       }
1309     }
1310   {
1311     % Neither is empty, meaning: we can (possibly) compare the
1312     % values of the current enclosing counter in the loop, if
1313     % they are equal, we are still in the loop, if they are
1314     % not, a sorting decision can be made directly.
1315     \tl_if_eq:NNTF
1316       \l__zrefclever_label_enclhead_a_tl
1317       \l__zrefclever_label_enclhead_b_tl
1318     {
1319       \int_compare:nNnTF
1320         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1321         =
1322         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1323     {

```

```

1324         \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1325         { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1326         \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1327         { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1328         \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1329         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1330         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1331         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1332     }
1333     {
1334         \bool_set_true:N \l__zrefclever_sort_decided_bool
1335         \int_compare:nNnTF
1336         { \tl_head:N \l__zrefclever_label_enclval_a_tl
1337           >
1338           { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1339           { \sort_return_swapped: }
1340           { \sort_return_same: }
1341         }
1342     }
1343     {
1344         \msg_warning:nnnn { zref-clever }
1345         { counters-not-nested } {#1} {#2}
1346         \bool_set_true:N \l__zrefclever_sort_decided_bool
1347         \sort_return_same:
1348     }
1349 }
1350 }
1351 }
1352 }
1353 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

`__zrefclever_sort_default_different_types:nn`

```

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

```

```

1374         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1375         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1376     }
1377 }
1378 }
1379 \bool_if:nTF
1380 {
1381     \int_compare_p:nNn
1382     { \l__zrefclever_sort_prior_a_int } <
1383     { \l__zrefclever_sort_prior_b_int }
1384 }
1385 { \sort_return_same: }
1386 {
1387     \bool_if:nTF
1388     {
1389         \int_compare_p:nNn
1390         { \l__zrefclever_sort_prior_a_int } >
1391         { \l__zrefclever_sort_prior_b_int }
1392     }
1393     { \sort_return_swapped: }
1394     {
1395         % Sort priorities are equal for different types: the type that
1396         % occurs first in 'labels', as given by the user, is kept (or
1397         % brought) forward.
1398         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1399         {
1400             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1401             { \seq_map_break:n { \sort_return_same: } }
1402             {
1403                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1404                 { \seq_map_break:n { \sort_return_swapped: } }
1405             }
1406         }
1407     }
1408 }
1409 }

```

(End definition for `__zrefclever_sort_default_different_types:nn`.)

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

```

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

1410 \cs_new_protected:Npn \__zrefclever_sort_page:nn #1#2
1411 {
1412     \int_compare:nNnTF
1413     { \zref@extractdefault {#1} { abspage } {-1} }
1414     >
1415     { \zref@extractdefault {#2} { abspage } {-1} }
1416     { \sort_return_swapped: }
1417     { \sort_return_same: }

```

```
1418 }
```

(End definition for `_zrefclever_sort_page:nn`.)

8 Typesetting

About possible alternatives to signal compression inhibition for individual labels, see <https://tex.stackexchange.com/q/611370> (thanks Enrico Gregorio, Phelype Oleinik, and Steven B. Segletes). Yet another alternative would be to receive an optional argument with the label(s) not to be compressed. This would be a repetition, but would keep the syntax “clean”. All in all, and rethinking this here, probably the best is simply to not allow individual inhibition of compression. We can already control compression of each individual call of `\zcref` with existing options, this should be enough. I don’t think the small extra flexibility this would grant is worth the syntax disruption it entails. Anyway, I have kept a “handle” to deal with this in case the need arises, in the form of `\l__zrefclever_range_inhibit_next_bool`, which is currently no-op, but is in place.

Variables

```
\l_zrefclever_typeset_last_bool
\l_zrefclever_last_of_type_bool
```

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

```
1419 \bool_new:N \l__zrefclever_typeset_last_bool
1420 \bool_new:N \l__zrefclever_last_of_type_bool
```

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

```
\l_zrefclever_typeset_labels_seq
\l_zrefclever_typeset_queue_prev_tl
\l_zrefclever_typeset_queue_curr_tl
\l_zrefclever_type_first_label_tl
\l_zrefclever_type_first_label_type_tl
```

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

```
1421 \seq_new:N \l__zrefclever_typeset_labels_seq
1422 \tl_new:N \l__zrefclever_typeset_queue_prev_tl
1423 \tl_new:N \l__zrefclever_typeset_queue_curr_tl
1424 \tl_new:N \l__zrefclever_type_first_label_tl
1425 \tl_new:N \l__zrefclever_type_first_label_type_tl
```

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

```
\l_zrefclever_label_count_int
\l_zrefclever_type_count_int
```

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

```
1426 \int_new:N \l__zrefclever_label_count_int
1427 \int_new:N \l__zrefclever_type_count_int
```

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

```

\l_zrefclever_range_count_int
\l_zrefclever_range_same_count_int
\l_zrefclever_range_beg_label_tl
\l_zrefclever_next_maybe_range_bool
\l_zrefclever_next_is_same_bool
\l_zrefclever_range_inhibit_next_bool

```

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

```

1428 \int_new:N \l_zrefclever_range_count_int
1429 \int_new:N \l_zrefclever_range_same_count_int
1430 \tl_new:N \l_zrefclever_range_beg_label_tl
1431 \bool_new:N \l_zrefclever_next_maybe_range_bool
1432 \bool_new:N \l_zrefclever_next_is_same_bool
1433 \bool_new:N \l_zrefclever_range_inhibit_next_bool

```

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

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

```

1434 \tl_new:N \l_zrefclever_namefont_tl
1435 \tl_new:N \l_zrefclever_reffont_out_tl
1436 \tl_new:N \l_zrefclever_reffont_in_tl
1437 \tl_new:N \l_zrefclever_namesep_tl
1438 \tl_new:N \l_zrefclever_rangesep_tl
1439 \tl_new:N \l_zrefclever_pairsep_tl
1440 \tl_new:N \l_zrefclever_listsep_tl
1441 \tl_new:N \l_zrefclever_lastsep_tl
1442 \tl_new:N \l_zrefclever_tpairsep_tl
1443 \tl_new:N \l_zrefclever_tlistsep_tl
1444 \tl_new:N \l_zrefclever_tlastsep_tl
1445 \tl_new:N \l_zrefclever_notesep_tl
1446 \tl_new:N \l_zrefclever_refpre_out_tl
1447 \tl_new:N \l_zrefclever_refpos_out_tl
1448 \tl_new:N \l_zrefclever_refpre_in_tl
1449 \tl_new:N \l_zrefclever_refpos_in_tl

```

(End definition for .)

```

\l_zrefclever_type_name_tl
\l_zrefclever_name_in_link_bool
\l_zrefclever_name_format_tl
\l_zrefclever_name_format_fallback_tl

```

Auxiliary variables for `__zrefclever_get_ref_first:` and `__zrefclever_type_name_setup:`.

```

1450 \tl_new:N \l_zrefclever_type_name_tl
1451 \bool_new:N \l_zrefclever_name_in_link_bool
1452 \tl_new:N \l_zrefclever_name_format_tl
1453 \tl_new:N \l_zrefclever_name_format_fallback_tl

```

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

Main functions

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

```

1454 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1455 {

```



```

1456 \seq_set_eq:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_zceref_labels_seq
1457 \tl_clear:N \l__zrefclever_typeset_queue_prev_tl
1458 \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1459 \tl_clear:N \l__zrefclever_type_first_label_tl
1460 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1461 \tl_clear:N \l__zrefclever_range_beg_label_tl
1462 \int_zero:N \l__zrefclever_label_count_int
1463 \int_zero:N \l__zrefclever_type_count_int
1464 \int_zero:N \l__zrefclever_range_count_int
1465 \int_zero:N \l__zrefclever_range_same_count_int
1466
1467 % Get not-type-specific separators and refpre/pos options.
1468 \__zrefclever_get_ref_string:nN {tpairsep} \l__zrefclever_tpairsep_tl
1469 \__zrefclever_get_ref_string:nN {tlistsep} \l__zrefclever_tlistsep_tl
1470 \__zrefclever_get_ref_string:nN {tlastsep} \l__zrefclever_tlastsep_tl
1471
1472 % Loop over the label list in sequence.
1473 \bool_set_false:N \l__zrefclever_typeset_last_bool
1474 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1475 {
1476   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_a_tl
1477   \seq_if_empty:NTF \l__zrefclever_typeset_labels_seq
1478   {
1479     \tl_clear:N \l__zrefclever_label_b_tl
1480     \bool_set_true:N \l__zrefclever_typeset_last_bool
1481   }
1482   { \seq_get_left:NN \l__zrefclever_typeset_labels_seq \l__zrefclever_label_b_tl }
1483
1484   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1485   {
1486     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1487     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1488   }
1489   {
1490     \tl_set:Nx \l__zrefclever_label_type_a_tl
1491     {
1492       \zref@extractdefault
1493       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1494     }
1495     \tl_set:Nx \l__zrefclever_label_type_b_tl
1496     {
1497       \zref@extractdefault
1498       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1499     }
1500   }
1501
1502   % First, we establish whether the ‘‘current label’’ (i.e. ‘a’) is the
1503   % last one of its type. This can happen because the ‘‘next label’’
1504   % (i.e. ‘b’) is of a different type (or different definition status),
1505   % or because we are at the end of the list.
1506   \bool_if:NTF \l__zrefclever_typeset_last_bool
1507   { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1508   {
1509     \zref@ifrefundefined { \l__zrefclever_label_a_tl }

```

```

1510     {
1511         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1512         { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1513         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1514     }
1515     {
1516         \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1517         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1518         {
1519             % Neither is undefined, we must check the types.
1520             \bool_if:nTF
1521             { % Both empty: same ‘type’.
1522                 {
1523                     \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1524                     \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1525                 }
1526                 { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1527                 {
1528                     \bool_if:nTF
1529                     { % Neither empty: compare types.
1530                         {
1531                             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1532                             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1533                         }
1534                         {
1535                             \tl_if_eq:NNTF
1536                             \l__zrefclever_label_type_a_tl \l__zrefclever_label_type_b_tl
1537                             { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1538                             { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1539                         }
1540                         % One empty, the other not: different ‘types’.
1541                         { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1542                     }
1543                 }
1544             }
1545         }
1546
1547 % Handle warnings in case of reference or type undefined.
1548 \zref@refused { \l__zrefclever_label_a_tl }
1549 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1550 {}
1551 {
1552     \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1553     {
1554         \msg_warning:nxx { zref-clever } { missing-type }
1555         { \l__zrefclever_label_a_tl }
1556     }
1557 }
1558
1559 % Get type-specific separators, refpre/pos and font options, once per
1560 % type.
1561 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1562 {
1563     \__zrefclever_get_ref_font:nN {namefont} \l__zrefclever_namefont_tl

```

```

1564         \_zrefclever_get_ref_font:nN {reffont}          \l_zrefclever_reffont_out_tl
1565         \_zrefclever_get_ref_font:nN {reffont-in}        \l_zrefclever_reffont_in_tl
1566         \_zrefclever_get_ref_string:nN {namesep}         \l_zrefclever_namesep_tl
1567         \_zrefclever_get_ref_string:nN {rangesep}        \l_zrefclever_rangesep_tl
1568         \_zrefclever_get_ref_string:nN {pairsep}         \l_zrefclever_pairsep_tl
1569         \_zrefclever_get_ref_string:nN {listsep}         \l_zrefclever_listsep_tl
1570         \_zrefclever_get_ref_string:nN {lastsep}         \l_zrefclever_lastsep_tl
1571         \_zrefclever_get_ref_string:nN {refpre}          \l_zrefclever_refpre_out_tl
1572         \_zrefclever_get_ref_string:nN {refpos}          \l_zrefclever_refpos_out_tl
1573         \_zrefclever_get_ref_string:nN {refpre-in}        \l_zrefclever_refpre_in_tl
1574         \_zrefclever_get_ref_string:nN {refpos-in}        \l_zrefclever_refpos_in_tl
1575     }
1576
1577     % Here we send this to a couple of auxiliary functions for no other
1578     % reason than to keep this long function a little less unreadable.
1579     \bool_if:NTF \l_zrefclever_last_of_type_bool
1580     {
1581         % There exists no next label of the same type as the current.
1582         \_zrefclever_typeset_refs_aux_last_of_type:
1583     }
1584     {
1585         % There exists a next label of the same type as the current.
1586         \_zrefclever_typeset_refs_aux_not_last_of_type:
1587     }
1588 }
1589 }

```

(End definition for _zrefclever_typeset_refs:.)

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

```

1590 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_last_of_type:
1591 {
1592     % Process the current label to the current queue.
1593     \int_case:nnF { \l_zrefclever_label_count_int }
1594     {
1595         % It is the last label of its type, but also the first one, and that's
1596         % what matters here: just store it.
1597         { 0 }
1598         {
1599             \tl_set:NV \l_zrefclever_type_first_label_tl \l_zrefclever_label_a_tl
1600             \tl_set:NV \l_zrefclever_type_first_label_type_tl \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 % If neither the first, nor the second: we have the last label
1617 % on the current type list (if not repeated).
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 { \l__zrefclever_range_same_count_int } = {1}
1641         {
1642           \exp_not:V \l__zrefclever_lastsep_tl
1643           \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1644         }
1645       }
1646     }
1647   }
1648   % Last in the range is third or more in it.
1649   {
1650     \int_case:nnF
1651     { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1652     {
1653       % Repetition, not a range.
1654       {0}
1655       {
1656         % If 'range_beg_label' is empty, it means it was also the
1657         % first of the type, and hence was already handled.
1658         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1659         {
1660           \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1661           {
1662             \exp_not:V \l__zrefclever_lastsep_tl
1663             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1664           }
1665         }
1666       }
1667       % A 'range', but with no skipped value, treat as list.

```

```

1668     {1}
1669     {
1670         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1671         {
1672             % Ditto.
1673             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1674             {
1675                 \exp_not:V \l__zrefclever_listsep_tl
1676                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1677             }
1678             \exp_not:V \l__zrefclever_lastsep_tl
1679             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1680         }
1681     }
1682 }
1683 {
1684     % An actual range.
1685     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1686     {
1687         % Ditto.
1688         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1689         {
1690             \exp_not:V \l__zrefclever_lastsep_tl
1691             \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1692         }
1693         \exp_not:V \l__zrefclever_rangesep_tl
1694         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1695     }
1696 }
1697 }
1698 }
1699
1700 % Handle ‘‘range’’ option. The idea is simple: if the queue is not empty,
1701 % we replace it with the end of the range (or pair). We can still
1702 % retrieve the end of the range from \cs{l__zrefclever_label_a_tl} since we know to
1703 % be processing the last label of its type at this point.
1704 \bool_if:NT \l__zrefclever_typeset_range_bool
1705 {
1706     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1707     {
1708         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1709         { }
1710         {
1711             \msg_warning:nxx { zref-clever } { single-element-range }
1712             { \l__zrefclever_type_first_label_type_tl }
1713         }
1714     }
1715     {
1716         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1717         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1718         { }
1719         {
1720             \__zrefclever_labels_in_sequence:nn
1721             { \l__zrefclever_type_first_label_tl } { \l__zrefclever_label_a_tl }

```

```

1722     }
1723     \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1724     {
1725         \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1726         { \exp_not:V \l__zrefclever_pairsep_tl }
1727         { \exp_not:V \l__zrefclever_rangesep_tl }
1728         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1729     }
1730 }
1731 }
1732
1733 % Now that the type is finished, we can add the name and the first ref to
1734 % the queue. Or, if ‘typeset’ option is not ‘both’, handle it here
1735 % too.
1736 \__zrefclever_type_name_setup:
1737 \bool_if:NTF
1738 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1739 {
1740     \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1741     { \__zrefclever_get_ref_first: }
1742 }
1743 {
1744     \bool_if:NTF
1745     { \l__zrefclever_typeset_ref_bool }
1746     {
1747         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1748         { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1749     }
1750     {
1751         \bool_if:NTF
1752         { \l__zrefclever_typeset_name_bool }
1753         {
1754             \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1755             {
1756                 \bool_if:NTF \l__zrefclever_name_in_link_bool
1757                 {
1758                     \exp_not:N \group_begin:
1759                     \exp_not:V \l__zrefclever_namefont_tl
1760                     % It's two 's', but escaped for DocStrip.
1761                     \exp_not:N \hyper@@link
1762                     {
1763                         \zref@ifrefcontainsprop
1764                         { \l__zrefclever_type_first_label_tl } { urluse }
1765                         {
1766                             \zref@extractdefault
1767                             { \l__zrefclever_type_first_label_tl }
1768                             { urluse } {}
1769                         }
1770                     }
1771                     \zref@extractdefault
1772                     { \l__zrefclever_type_first_label_tl }
1773                     { url } {}
1774                 }
1775             }

```

```

1776         {
1777             \zref@extractdefault
1778             { \l__zrefclever_type_first_label_tl } { anchor } {}
1779         }
1780         { \exp_not:V \l__zrefclever_type_name_tl }
1781         \exp_not:N \group_end:
1782     }
1783     {
1784         \exp_not:N \group_begin:
1785         \exp_not:V \l__zrefclever_namefont_tl
1786         \exp_not:V \l__zrefclever_type_name_tl
1787         \exp_not:N \group_end:
1788     }
1789 }
1790 }
1791 {
1792     % This case would correspond to "typeset=none" but should not
1793     % happen, given the options are set up to typeset at least one
1794     % of "ref" or "name", but a sensible fallback, equal to the
1795     % behavior of ‘‘both’’.
1796     \tl_put_left:Nx
1797         \l__zrefclever_typeset_queue_curr_tl { \__zrefclever_get_ref_first: }
1798 }
1799 }
1800 }
1801
1802 % Typeset the previous type, if there is one.
1803 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1804 {
1805     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1806     { \l__zrefclever_tlistsep_tl }
1807     \l__zrefclever_typeset_queue_prev_tl
1808 }
1809
1810 % Wrap up loop, or prepare for next iteration.
1811 \bool_if:NTF \l__zrefclever_typeset_last_bool
1812 {
1813     % We are finishing, typeset the current queue.
1814     \int_case:nnF { \l__zrefclever_type_count_int }
1815     {
1816         % Single type.
1817         { 0 }
1818         { \l__zrefclever_typeset_queue_curr_tl }
1819         % Pair of types.
1820         { 1 }
1821         {
1822             \l__zrefclever_tpairsep_tl
1823             \l__zrefclever_typeset_queue_curr_tl
1824         }
1825     }
1826 }
1827 {
1828     % Last in list of types.
1829     \l__zrefclever_tlastsep_tl
1830     \l__zrefclever_typeset_queue_curr_tl

```

```

1830     }
1831   }
1832   {
1833     % There are further labels, set variables for next iteration.
1834     \tl_set_eq:NN
1835       \l__zrefclever_typeset_queue_prev_tl \l__zrefclever_typeset_queue_curr_tl
1836     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1837     \tl_clear:N \l__zrefclever_type_first_label_tl
1838     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1839     \tl_clear:N \l__zrefclever_range_beg_label_tl
1840     \int_zero:N \l__zrefclever_label_count_int
1841     \int_incr:N \l__zrefclever_type_count_int
1842     \int_zero:N \l__zrefclever_range_count_int
1843     \int_zero:N \l__zrefclever_range_same_count_int
1844   }
1845 }

```

(End definition for `__zrefclever_typeset_refs_aux_last_of_type:`)

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

```

1846 \cs_new_protected:Npn \__zrefclever_typeset_refs_aux_not_last_of_type:
1847 {
1848   % Signal if next label may form a range with the current one (of
1849   % course, only considered if compression is enabled in the first
1850   % place).
1851   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1852   \bool_set_false:N \l__zrefclever_next_is_same_bool
1853   \bool_lazy_and:nnT
1854     { \l__zrefclever_typeset_compress_bool }
1855     % Currently no-op, but kept as ‘‘handle’’ to inhibit compression of
1856     % individual labels.
1857     { ! \l__zrefclever_range_inhibit_next_bool }
1858   {
1859     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1860     { }
1861     {
1862       \__zrefclever_labels_in_sequence:nn
1863         { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1864     }
1865   }
1866
1867   % Process the current label to the current queue.
1868   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1869   {
1870     % Current label is the first of its type (also not the last, but it
1871     % doesn't matter here): just store the label.
1872     \tl_set:NV \l__zrefclever_type_first_label_tl \l__zrefclever_label_a_tl
1873     \tl_set:NV \l__zrefclever_type_first_label_type_tl \l__zrefclever_label_type_a_tl
1874
1875     % If the next label may be part of a range, we set ‘range_beg_label’
1876     % to ‘‘empty’’ (we deal with it as the ‘‘first’’, and must do it
1877     % there, to handle hyperlinking), but also step the range counters.
1878     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1879     {

```



```

1880         \tl_clear:N \l__zrefclever_range_beg_label_tl
1881         \int_incr:N \l__zrefclever_range_count_int
1882         \bool_if:NT \l__zrefclever_next_is_same_bool
1883         { \int_incr:N \l__zrefclever_range_same_count_int }
1884     }
1885 }
1886 {
1887     % Current label is neither the first (nor the last) of its
1888     % type.
1889     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1890     {
1891         % Starting, or continuing a range.
1892         \int_compare:nNnTF
1893         { \l__zrefclever_range_count_int } = {0}
1894         {
1895             % There was no range going, we are starting one.
1896             \tl_set:NV \l__zrefclever_range_beg_label_tl \l__zrefclever_label_a_tl
1897             \int_incr:N \l__zrefclever_range_count_int
1898             \bool_if:NT \l__zrefclever_next_is_same_bool
1899             { \int_incr:N \l__zrefclever_range_same_count_int }
1900         }
1901         {
1902             % Second or more in the range, but not the last.
1903             \int_incr:N \l__zrefclever_range_count_int
1904             \bool_if:NT \l__zrefclever_next_is_same_bool
1905             { \int_incr:N \l__zrefclever_range_same_count_int }
1906         }
1907     }
1908 }
1909 % Next element is not in sequence, meaning: there was no range, or
1910 % we are closing one.
1911 \int_case:nnF { \l__zrefclever_range_count_int }
1912 {
1913     % There was no range going on.
1914     {0}
1915     {
1916         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1917         {
1918             \exp_not:V \l__zrefclever_listsep_tl
1919             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1920         }
1921     }
1922     % Last is second in the range: if 'range_same_count' is also
1923     % '1', it's a repetition (drop it), otherwise, it's a 'pair
1924     % within a list'', treat as list.
1925     {1}
1926     {
1927         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1928         {
1929             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1930             {
1931                 \exp_not:V \l__zrefclever_listsep_tl
1932                 \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1933             }

```

```

1934         \int_compare:nNnF { \l__zrefclever_range_same_count_int } = {1}
1935         {
1936             \exp_not:V \l__zrefclever_listsep_tl
1937             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1938         }
1939     }
1940 }
1941 }
1942 {
1943     % Last is third or more in the range: if 'range_count' and
1944     % 'range_same_count' are the same, its a repetition (drop it),
1945     % if they differ by '1', its a list, if they differ by more,
1946     % it is a real range.
1947     \int_case:nnF
1948     { \l__zrefclever_range_count_int - \l__zrefclever_range_same_count_int }
1949     {
1950         {0}
1951         {
1952             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1953             {
1954                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1955                 {
1956                     \exp_not:V \l__zrefclever_listsep_tl
1957                     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1958                 }
1959             }
1960         }
1961         {1}
1962         {
1963             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1964             {
1965                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1966                 {
1967                     \exp_not:V \l__zrefclever_listsep_tl
1968                     \__zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1969                 }
1970                 \exp_not:V \l__zrefclever_listsep_tl
1971                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1972             }
1973         }
1974     }
1975 }
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 \l__zrefclever_range_beg_label_tl
1983         }
1984         \exp_not:V \l__zrefclever_rangesep_tl
1985         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1986     }
1987 }

```

```

1988         % Reset counters.
1989         \int_zero:N \l__zrefclever_range_count_int
1990         \int_zero:N \l__zrefclever_range_same_count_int
1991     }
1992 }
1993 % Step label counter for next iteration.
1994 \int_incr:N \l__zrefclever_label_count_int
1995 }

```

(End definition for __zrefclever_typeset_refs_aux_not_last_of_type:.)

Aux functions

```

1996 \cs_new_protected:Npn \__zrefclever_ref_default:
1997 { \zref@default }
1998 \cs_new_protected:Npn \__zrefclever_name_default:
1999 { \zref@default }

```

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

```

2000 \cs_new:Npn \__zrefclever_get_ref:n #1
2001 {
2002   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2003   {
2004     \bool_if:nTF
2005       { \l__zrefclever_use_hyperref_bool && ! \l__zrefclever_link_star_bool }
2006       {
2007         \exp_not:N \group_begin:
2008         \exp_not:V \l__zrefclever_reffont_out_tl
2009         \exp_not:V \l__zrefclever_refpre_out_tl
2010         \exp_not:N \group_begin:
2011         \exp_not:V \l__zrefclever_reffont_in_tl
2012         % It's two '@s', but escaped for DocStrip.
2013         \exp_not:N \hyper@@link
2014         {
2015           \zref@ifrefcontainsprop {#1} { urluse }
2016           { \zref@extractdefault {#1} { urluse } {} }
2017           { \zref@extractdefault {#1} { url } {} }
2018         }
2019         { \zref@extractdefault {#1} { anchor } {} }
2020         {
2021           \exp_not:V \l__zrefclever_refpre_in_tl
2022           \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2023           \exp_not:V \l__zrefclever_refpos_in_tl
2024         }
2025         \exp_not:N \group_end:
2026         \exp_not:V \l__zrefclever_refpos_out_tl
2027         \exp_not:N \group_end:
2028       }

```

```

2029     {
2030         \exp_not:N \group_begin:
2031         \exp_not:V \l__zrefclever_reffont_out_tl
2032         \exp_not:V \l__zrefclever_refpre_out_tl
2033         \exp_not:N \group_begin:
2034         \exp_not:V \l__zrefclever_reffont_in_tl
2035         \exp_not:V \l__zrefclever_refpre_in_tl
2036         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } {}
2037         \exp_not:V \l__zrefclever_refpos_in_tl
2038         \exp_not:N \group_end:
2039         \exp_not:V \l__zrefclever_refpos_out_tl
2040         \exp_not:N \group_end:
2041     }
2042 }
2043 { \exp_not:N \__zrefclever_ref_default: }
2044 }
2045 \cs_generate_variant:Nn \__zrefclever_get_ref:n { V }

```

(End definition for __zrefclever_get_ref:n.)

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

```

2046 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2047 {
2048     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2049     { \tl_clear:N \l__zrefclever_type_name_tl }
2050     {
2051         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2052         { \tl_clear:N \l__zrefclever_type_name_tl }
2053         {

```

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

```

2054     \bool_lazy_or:nnTF
2055     { \l__zrefclever_capitalize_bool }
2056     {
2057         \l__zrefclever_capitalize_first_bool &&
2058         \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2059     }
2060     { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2061     { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2062     % If the queue is empty, we have a singular, otherwise, plural.
2063     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2064     { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2065     { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2066     \bool_lazy_and:nnTF
2067     { \l__zrefclever_abbrev_bool }
2068     {
2069         ! \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 } ||
2070         ! \l__zrefclever_noabbrev_first_bool
2071     }
2072     {
2073         \tl_set:NV \l__zrefclever_name_format_fallback_tl \l__zrefclever_name_format
2074         \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2075     }
2076     { \tl_clear:N \l__zrefclever_name_format_fallback_tl }

```

```

2077
2078 \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2079 {
2080   \prop_get:cVNF
2081     { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2082       \l__zrefclever_name_format_tl
2083       \l__zrefclever_type_name_tl
2084       {
2085         \__zrefclever_get_type_transl:xxxNF
2086         { \l__zrefclever_ref_language_tl }
2087         { \l__zrefclever_type_first_label_type_tl }
2088         { \l__zrefclever_name_format_tl }
2089         \l__zrefclever_type_name_tl
2090         {
2091           \tl_clear:N \l__zrefclever_type_name_tl
2092           \msg_warning:nnx { zref-clever } { missing-name }
2093           { \l__zrefclever_type_first_label_type_tl }
2094         }
2095       }
2096     }
2097   {
2098     \prop_get:cVNF
2099       { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options_pro
2100         \l__zrefclever_name_format_tl
2101         \l__zrefclever_type_name_tl
2102         {
2103           \prop_get:cVNF
2104             { \l__zrefclever_type_ \l__zrefclever_type_first_label_type_tl _options
2105               \l__zrefclever_name_format_fallback_tl
2106               \l__zrefclever_type_name_tl
2107               {
2108                 \__zrefclever_get_type_transl:xxxNF
2109                 { \l__zrefclever_ref_language_tl }
2110                 { \l__zrefclever_type_first_label_type_tl }
2111                 { \l__zrefclever_name_format_tl }
2112                 \l__zrefclever_type_name_tl
2113                 {
2114                   \__zrefclever_get_type_transl:xxxNF
2115                   { \l__zrefclever_ref_language_tl }
2116                   { \l__zrefclever_type_first_label_type_tl }
2117                   { \l__zrefclever_name_format_fallback_tl }
2118                   \l__zrefclever_type_name_tl
2119                   {
2120                     \tl_clear:N \l__zrefclever_type_name_tl
2121                     \msg_warning:nnx { zref-clever } { missing-name }
2122                     { \l__zrefclever_type_first_label_type_tl }
2123                   }
2124                 }
2125               }
2126             }
2127           }
2128         }
2129       }

```

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

```

2130 \bool_lazy_any:nTF
2131 {
2132   { ! \l__zrefclever_use_hyperref_bool }
2133   { \l__zrefclever_link_star_bool }
2134   { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2135   { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2136 }
2137 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2138 {
2139   \bool_lazy_any:nTF
2140   {
2141     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2142     {
2143       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2144       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2145     }
2146     {
2147       \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2148       \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2149       \l__zrefclever_typeset_last_bool &&
2150       \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2151     }
2152   }
2153   { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2154   { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2155 }
2156 }

```

(End definition for __zrefclever_type_name_setup:.)

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

```

2157 \cs_new:Npn \__zrefclever_get_ref_first:
2158 {
2159   \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2160   { \exp_not:N \__zrefclever_ref_default: }
2161   {
2162     \bool_if:NTF \l__zrefclever_name_in_link_bool
2163     {
2164       \zref@ifrefcontainsprop
2165       { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2166       {
2167         % It's two '@s', but escaped for DocStrip.
2168         \exp_not:N \hyper@@link
2169         {
2170           \zref@ifrefcontainsprop
2171           { \l__zrefclever_type_first_label_tl } { urluse }
2172           {
2173             \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2174             { urluse } {}
2175           }
2176         }

```

```

2177         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2178         { url } {}
2179     }
2180 }
2181 {
2182     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2183     { anchor } {}
2184 }
2185 {
2186     \exp_not:N \group_begin:
2187     \exp_not:V \l__zrefclever_namefont_tl
2188     \exp_not:V \l__zrefclever_type_name_tl
2189     \exp_not:N \group_end:
2190     \exp_not:V \l__zrefclever_namesep_tl
2191     \exp_not:N \group_begin:
2192     \exp_not:V \l__zrefclever_reffont_out_tl
2193     \exp_not:V \l__zrefclever_refpre_out_tl
2194     \exp_not:N \group_begin:
2195     \exp_not:V \l__zrefclever_reffont_in_tl
2196     \exp_not:V \l__zrefclever_refpre_in_tl
2197     \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2198     { \l__zrefclever_ref_property_tl } {}
2199     \exp_not:V \l__zrefclever_refpos_in_tl
2200     \exp_not:N \group_end:
2201     % hyperlink makes it's own group, we'd like to close the
2202     % 'refpre-out' group after 'refpos-out', but... we close
2203     % it here, and give the trailing 'refpos-out' its own
2204     % group. This will result that formatting given to
2205     % 'refpre-out' will not reach 'refpos-out', but I see no
2206     % alternative, and this has to be handled specially.
2207     \exp_not:N \group_end:
2208 }
2209 \exp_not:N \group_begin:
2210 % Ditto: special treatment.
2211 \exp_not:V \l__zrefclever_reffont_out_tl
2212 \exp_not:V \l__zrefclever_refpos_out_tl
2213 \exp_not:N \group_end:
2214 }
2215 {
2216     \exp_not:N \group_begin:
2217     \exp_not:V \l__zrefclever_namefont_tl
2218     \exp_not:V \l__zrefclever_type_name_tl
2219     \exp_not:N \group_end:
2220     \exp_not:V \l__zrefclever_namesep_tl
2221     \exp_not:N \l__zrefclever_ref_default:
2222 }
2223 }
2224 {
2225     \tl_if_empty:NTF \l__zrefclever_type_name_tl
2226     {
2227         \exp_not:N \l__zrefclever_name_default:
2228         \exp_not:V \l__zrefclever_namesep_tl
2229     }
2230     {

```

```

2231 \exp_not:N \group_begin:
2232 \exp_not:V \l__zrefclever_namefont_tl
2233 \exp_not:V \l__zrefclever_type_name_tl
2234 \exp_not:N \group_end:
2235 \exp_not:V \l__zrefclever_namesep_tl
2236 }
2237 \zref@ifrefcontainsprop
2238 { \l__zrefclever_type_first_label_tl } { \l__zrefclever_ref_property_tl }
2239 {
2240 \bool_if:nTF
2241 {
2242 \l__zrefclever_use_hyperref_bool &&
2243 ! \l__zrefclever_link_star_bool
2244 }
2245 {
2246 \exp_not:N \group_begin:
2247 \exp_not:V \l__zrefclever_reffont_out_tl
2248 \exp_not:V \l__zrefclever_refpre_out_tl
2249 \exp_not:N \group_begin:
2250 \exp_not:V \l__zrefclever_reffont_in_tl
2251 % It's two '@s', but escaped for DocStrip.
2252 \exp_not:N \hyper@@link
2253 {
2254 \zref@ifrefcontainsprop
2255 { \l__zrefclever_type_first_label_tl } { urluse }
2256 {
2257 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2258 { urluse } {}
2259 }
2260 {
2261 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2262 { url } {}
2263 }
2264 }
2265 {
2266 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2267 { anchor } {}
2268 }
2269 {
2270 \exp_not:V \l__zrefclever_refpre_in_tl
2271 \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2272 { \l__zrefclever_ref_property_tl } {}
2273 \exp_not:V \l__zrefclever_refpos_in_tl
2274 }
2275 \exp_not:N \group_end:
2276 \exp_not:V \l__zrefclever_refpos_out_tl
2277 \exp_not:N \group_end:
2278 }
2279 {
2280 \exp_not:N \group_begin:
2281 \exp_not:V \l__zrefclever_reffont_out_tl
2282 \exp_not:V \l__zrefclever_refpre_out_tl
2283 \exp_not:N \group_begin:
2284 \exp_not:V \l__zrefclever_reffont_in_tl

```



```

2285         \exp_not:V \l__zrefclever_refpre_in_tl
2286         \zref@extractdefault { \l__zrefclever_type_first_label_tl }
2287         { \l__zrefclever_ref_property_tl } {}
2288         \exp_not:V \l__zrefclever_refpos_in_tl
2289         \exp_not:N \group_end:
2290         \exp_not:V \l__zrefclever_refpos_out_tl
2291         \exp_not:N \group_end:
2292     }
2293 }
2294 { \exp_not:N \__zrefclever_ref_default: }
2295 }
2296 }
2297 }

```

(End definition for __zrefclever_get_ref_first:.)

__zrefclever_get_ref_string:nN

```

2298 % \Arg{option} \Arg{var to store result}
2299 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2300 {
2301     % First attempt options stored in \cs{l__zrefclever_ref_options_prop}.
2302     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2303     {
2304         % If not found, try the type specific options.
2305         \bool_lazy_all:nTF
2306         {
2307             { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2308             {
2309                 \prop_if_exist_p:c
2310                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop }
2311             }
2312             {
2313                 \prop_if_in_p:cn
2314                 { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1}
2315             }
2316         }
2317         {
2318             \prop_get:cnN
2319             { l__zrefclever_type_ \l__zrefclever_label_type_a_tl _options_prop } {#1} #2
2320         }
2321         {
2322             % If not found, try the type specific translations.
2323             \__zrefclever_get_type_transl:xxnNF
2324             { \l__zrefclever_ref_language_tl }
2325             { \l__zrefclever_label_type_a_tl }
2326             {#1} #2
2327             {
2328                 % If not found, try default translations.
2329                 \__zrefclever_get_default_transl:xxnNF
2330                 { \l__zrefclever_ref_language_tl }
2331                 {#1} #2
2332                 {
2333                     % If not found, try fallback.
2334                     \__zrefclever_get_fallback_transl:nNF {#1} #2

```

```

2335         {
2336             \tl_clear:N #2
2337             \msg_warning:nnn { zref-clever }
2338                 { missing-string } {#1}
2339         }
2340     }
2341 }
2342 }
2343 }
2344 }

```

(End definition for _zrefclever_get_ref_string:nN.)

_zrefclever_get_ref_font:nN

```

2345 \cs_new_protected:Npn \_zrefclever_get_ref_font:nN #1#2
2346 {
2347     % First attempt options stored in \cs{l\_zrefclever_ref_options_prop}.
2348     \prop_get:NnNF \l\_zrefclever_ref_options_prop {#1} #2
2349     {
2350         % If not found, try the type specific options.
2351         \bool_lazy_and:nnTF
2352             { ! \tl_if_empty_p:N \l\_zrefclever_label_type_a_tl }
2353             {
2354                 \prop_if_exist_p:c
2355                     { l\_zrefclever_type_ \l\_zrefclever_label_type_a_tl _options_prop }
2356             }
2357             {
2358                 \prop_get:cnNF
2359                     { l\_zrefclever_type_ \l\_zrefclever_label_type_a_tl _options_prop } {#1} #2
2360                     { \tl_clear:N #2 }
2361             }
2362             { \tl_clear:N #2 }
2363     }
2364 }

```

(End definition for _zrefclever_get_ref_font:nN.)

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

```

2365 \cs_new_protected:Npn \_zrefclever_labels_in_sequence:nn #1#2
2366 {
2367     \tl_if_eq:NnTF \l\_zrefclever_ref_property_tl { page }
2368     {
2369         \exp_args:Nxx \tl_if_eq:nnT
2370             { \zref@extractdefault {#1} { zc@pgfmt } { } }
2371             { \zref@extractdefault {#2} { zc@pgfmt } { } }
2372         {
2373             \int_compare:nNnTF
2374                 { \zref@extractdefault {#1} { zc@pgval } {-2} + 1 }
2375                 =
2376                 { \zref@extractdefault {#2} { zc@pgval } {-1} }
2377                 { \bool_set_true:N \l\_zrefclever_next_maybe_range_bool }
2378         {

```

```

2379         \int_compare:nNnT
2380         { \zref@extractdefault {#1} { zc@pgval } {-1} }
2381         =
2382         { \zref@extractdefault {#2} { zc@pgval } {-1} }
2383         {
2384             \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2385             \bool_set_true:N \l__zrefclever_next_is_same_bool
2386         }
2387     }
2388 }
2389 }
2390 {
2391     \exp_args:Nxx \tl_if_eq:nnT
2392     { \zref@extractdefault {#1} { counter } { } }
2393     { \zref@extractdefault {#2} { counter } { } }
2394     {
2395         \exp_args:Nxx \tl_if_eq:nnT
2396         { \zref@extractdefault {#1} { zc@enclval } { } }
2397         { \zref@extractdefault {#2} { zc@enclval } { } }
2398         {
2399             \int_compare:nNnTF
2400             { \zref@extractdefault {#1} { zc@cntval } {-2} + 1 }
2401             =
2402             { \zref@extractdefault {#2} { zc@cntval } {-1} }
2403             { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2404             {
2405                 \int_compare:nNnT
2406                 { \zref@extractdefault {#1} { zc@cntval } {-1} }
2407                 =
2408                 { \zref@extractdefault {#2} { zc@cntval } {-1} }
2409                 {
2410                     \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2411                     \bool_set_true:N \l__zrefclever_next_is_same_bool
2412                 }
2413             }
2414         }
2415     }
2416 }
2417 }

```

(End definition for `__zrefclever_labels_in_sequence:nn`.)

9 Special handling

This section is meant to aggregate any “special handling” needed for L^AT_EX kernel features, document classes, and packages, needed for `zref-clever` to work properly with them. It is not meant to be a “kitchen sink of workarounds”. Rather, I intend to keep this as lean as possible, trying to add things selectively when they are safe and reasonable. And, hopefully, doing so by proper setting of `zref-clever`’s options, not by messing with other packages’ code. In particular, I do not mean to compensate for “lack of support for `zref`” by individual packages here, unless there is really no alternative.

9.1 \appendix

Another relevant use case of the same general problem of different types for the same counter is the `\appendix` which in some document classes, including the standard ones, change the sectioning commands looks but, of course, keep using the same counter (`book.cls` and `report.cls` reset counters `chapter` and `section` to 0, change `\@chapapp` to use `\appendixname` and use `\@Alph` for `\thechapter`; `article.cls` resets counters `section` and `subsection` to 0, and uses `\@Alph` for `\thesection`; `memoir.cls`, `scrbook.cls` and `scrarticle.cls` do the same as their corresponding standard classes, and sometimes a little more, but what interests us here is pretty much the same; see also the `appendix` package).

9.2 \newtheorem

9.3 enumitem package

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

```
2418 \</package>
```

10 Dictionaries

10.1 English

```
2419 \<package>\zcDeclareLanguage { english }
2420 \<package>\zcDeclareLanguageAlias { american } { english }
2421 \<package>\zcDeclareLanguageAlias { australian } { english }
2422 \<package>\zcDeclareLanguageAlias { british } { english }
2423 \<package>\zcDeclareLanguageAlias { canadian } { english }
2424 \<package>\zcDeclareLanguageAlias { newzealand } { english }
2425 \<package>\zcDeclareLanguageAlias { UKenglish } { english }
2426 \<package>\zcDeclareLanguageAlias { USenglish } { english }
2427 \<*dict-english>

2428 namesep = {\nobreakspace} ,
2429 pairsep = {\~and\nobreakspace} ,
2430 listsep = {\~,~} ,
2431 lastsep = {\~and\nobreakspace} ,
2432 tpairsep = {\~and\nobreakspace} ,
2433 tlistsep = {\~,~} ,
2434 tlastsep = {\~,~and\nobreakspace} ,
2435 notesep = {\~} ,
2436 rangesep = {\~to\nobreakspace} ,
2437
2438 type = part ,
2439 Name-sg = Part ,
2440 name-sg = part ,
2441 Name-pl = Parts ,
2442 name-pl = parts ,
2443
2444 type = chapter ,
2445 Name-sg = Chapter ,
```

```

2446     name-sg = chapter ,
2447     Name-pl = Chapters ,
2448     name-pl = chapters ,
2449
2450 type = section ,
2451     Name-sg = Section ,
2452     name-sg = section ,
2453     Name-pl = Sections ,
2454     name-pl = sections ,
2455
2456 type = paragraph ,
2457     Name-sg = Paragraph ,
2458     name-sg = paragraph ,
2459     Name-pl = Paragraphs ,
2460     name-pl = paragraphs ,
2461     Name-sg-ab = Par. ,
2462     name-sg-ab = par. ,
2463     Name-pl-ab = Par. ,
2464     name-pl-ab = par. ,
2465
2466 type = appendix ,
2467     Name-sg = Appendix ,
2468     name-sg = appendix ,
2469     Name-pl = Appendices ,
2470     name-pl = appendices ,
2471
2472 type = page ,
2473     Name-sg = Page ,
2474     name-sg = page ,
2475     Name-pl = Pages ,
2476     name-pl = pages ,
2477     name-sg-ab = p. ,
2478     name-pl-ab = pp. ,
2479
2480 type = line ,
2481     Name-sg = Line ,
2482     name-sg = line ,
2483     Name-pl = Lines ,
2484     name-pl = lines ,
2485
2486 type = figure ,
2487     Name-sg = Figure ,
2488     name-sg = figure ,
2489     Name-pl = Figures ,
2490     name-pl = figures ,
2491     Name-sg-ab = Fig. ,
2492     name-sg-ab = fig. ,
2493     Name-pl-ab = Figs. ,
2494     name-pl-ab = figs. ,
2495
2496 type = table ,
2497     Name-sg = Table ,
2498     name-sg = table ,
2499     Name-pl = Tables ,

```

```

2500     name-pl = tables ,
2501
2502 type = item ,
2503     Name-sg = Item ,
2504     name-sg = item ,
2505     Name-pl = Items ,
2506     name-pl = items ,
2507
2508 type = footnote ,
2509     Name-sg = Footnote ,
2510     name-sg = footnote ,
2511     Name-pl = Footnotes ,
2512     name-pl = footnotes ,
2513
2514 type = note ,
2515     Name-sg = Note ,
2516     name-sg = note ,
2517     Name-pl = Notes ,
2518     name-pl = notes ,
2519
2520 type = equation ,
2521     Name-sg = Equation ,
2522     name-sg = equation ,
2523     Name-pl = Equations ,
2524     name-pl = equations ,
2525     Name-sg-ab = Eq. ,
2526     name-sg-ab = eq. ,
2527     Name-pl-ab = Eqs. ,
2528     name-pl-ab = eqs. ,
2529     refpre-in = {(} ,
2530     refpos-in = {)} ,
2531
2532 type = theorem ,
2533     Name-sg = Theorem ,
2534     name-sg = theorem ,
2535     Name-pl = Theorems ,
2536     name-pl = theorems ,
2537
2538 type = lemma ,
2539     Name-sg = Lemma ,
2540     name-sg = lemma ,
2541     Name-pl = Lemmas ,
2542     name-pl = lemmas ,
2543
2544 type = corollary ,
2545     Name-sg = Corollary ,
2546     name-sg = corollary ,
2547     Name-pl = Corollaries ,
2548     name-pl = corollaries ,
2549
2550 type = proposition ,
2551     Name-sg = Proposition ,
2552     name-sg = proposition ,
2553     Name-pl = Propositions ,

```

```

2554     name-pl = propositions ,
2555
2556 type = definition ,
2557     Name-sg = Definition ,
2558     name-sg = definition ,
2559     Name-pl = Definitions ,
2560     name-pl = definitions ,
2561
2562 type = proof ,
2563     Name-sg = Proof ,
2564     name-sg = proof ,
2565     Name-pl = Proofs ,
2566     name-pl = proofs ,
2567
2568 type = result ,
2569     Name-sg = Result ,
2570     name-sg = result ,
2571     Name-pl = Results ,
2572     name-pl = results ,
2573
2574 type = example ,
2575     Name-sg = Example ,
2576     name-sg = example ,
2577     Name-pl = Examples ,
2578     name-pl = examples ,
2579
2580 type = remark ,
2581     Name-sg = Remark ,
2582     name-sg = remark ,
2583     Name-pl = Remarks ,
2584     name-pl = remarks ,
2585
2586 type = algorithm ,
2587     Name-sg = Algorithm ,
2588     name-sg = algorithm ,
2589     Name-pl = Algorithms ,
2590     name-pl = algorithms ,
2591
2592 type = listing ,
2593     Name-sg = Listing ,
2594     name-sg = listing ,
2595     Name-pl = Listings ,
2596     name-pl = listings ,
2597
2598 type = exercise ,
2599     Name-sg = Exercise ,
2600     name-sg = exercise ,
2601     Name-pl = Exercises ,
2602     name-pl = exercises ,
2603
2604 type = solution ,
2605     Name-sg = Solution ,
2606     name-sg = solution ,
2607     Name-pl = Solutions ,

```

```

2608     name-pl = solutions ,
2609 </dict-english>

```

10.2 German

```

2610 <package>\zcDeclareLanguage { german }
2611 <package>\zcDeclareLanguageAlias { austrian      } { german }
2612 <package>\zcDeclareLanguageAlias { germanb       } { german }
2613 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2614 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2615 <package>\zcDeclareLanguageAlias { nswissgerman  } { german }
2616 <package>\zcDeclareLanguageAlias { swissgerman   } { german }
2617 <*dict-german>

2618 namesep = {\nobreakspace} ,
2619 pairsep  = {\und\nobreakspace} ,
2620 listsep  = {,~} ,
2621 lastsep  = {\und\nobreakspace} ,
2622 tpairsep = {\und\nobreakspace} ,
2623 tlistsep = {,~} ,
2624 tlastsep = {\und\nobreakspace} ,
2625 notesep  = {~} ,
2626 rangesep = {\bis\nobreakspace} ,
2627
2628 type = part ,
2629     Name-sg = Teil ,
2630     name-sg = Teil ,
2631     Name-pl = Teile ,
2632     name-pl = Teile ,
2633
2634 type = chapter ,
2635     Name-sg = Kapitel ,
2636     name-sg = Kapitel ,
2637     Name-pl = Kapitel ,
2638     name-pl = Kapitel ,
2639
2640 type = section ,
2641     Name-sg = Abschnitt ,
2642     name-sg = Abschnitt ,
2643     Name-pl = Abschnitte ,
2644     name-pl = Abschnitte ,
2645
2646 type = paragraph ,
2647     Name-sg = Absatz ,
2648     name-sg = Absatz ,
2649     Name-pl = Absätze ,
2650     name-pl = Absätze ,
2651
2652 type = appendix ,
2653     Name-sg = Anhang ,
2654     name-sg = Anhang ,
2655     Name-pl = Anhänge ,
2656     name-pl = Anhänge ,
2657
2658 type = page ,

```



```

2659     Name-sg = Seite ,
2660     name-sg = Seite ,
2661     Name-pl = Seiten ,
2662     name-pl = Seiten ,
2663
2664     type = line ,
2665     Name-sg = Zeile ,
2666     name-sg = Zeile ,
2667     Name-pl = Zeilen ,
2668     name-pl = Zeilen ,
2669
2670     type = figure ,
2671     Name-sg = Abbildung ,
2672     name-sg = Abbildung ,
2673     Name-pl = Abbildungen ,
2674     name-pl = Abbildungen ,
2675     Name-sg-ab = Abb. ,
2676     name-sg-ab = Abb. ,
2677     Name-pl-ab = Abb. ,
2678     name-pl-ab = Abb. ,
2679
2680     type = table ,
2681     Name-sg = Tabelle ,
2682     name-sg = Tabelle ,
2683     Name-pl = Tabellen ,
2684     name-pl = Tabellen ,
2685
2686     type = item ,
2687     Name-sg = Punkt ,
2688     name-sg = Punkt ,
2689     Name-pl = Punkte ,
2690     name-pl = Punkte ,
2691
2692     type = footnote ,
2693     Name-sg = Fußnote ,
2694     name-sg = Fußnote ,
2695     Name-pl = Fußnoten ,
2696     name-pl = Fußnoten ,
2697
2698     type = note ,
2699     Name-sg = Anmerkung ,
2700     name-sg = Anmerkung ,
2701     Name-pl = Anmerkungen ,
2702     name-pl = Anmerkungen ,
2703
2704     type = equation ,
2705     Name-sg = Gleichung ,
2706     name-sg = Gleichung ,
2707     Name-pl = Gleichungen ,
2708     name-pl = Gleichungen ,
2709     refpre-in = {() ,
2710     refpos-in = {()}} ,
2711
2712     type = theorem ,

```

```

2713 Name-sg = Theorem ,
2714 name-sg = Theorem ,
2715 Name-pl = Theoreme ,
2716 name-pl = Theoreme ,
2717
2718 type = lemma ,
2719 Name-sg = Lemma ,
2720 name-sg = Lemma ,
2721 Name-pl = Lemmata ,
2722 name-pl = Lemmata ,
2723
2724 type = corollary ,
2725 Name-sg = Korollar ,
2726 name-sg = Korollar ,
2727 Name-pl = Korollare ,
2728 name-pl = Korollare ,
2729
2730 type = proposition ,
2731 Name-sg = Satz ,
2732 name-sg = Satz ,
2733 Name-pl = Sätze ,
2734 name-pl = Sätze ,
2735
2736 type = definition ,
2737 Name-sg = Definition ,
2738 name-sg = Definition ,
2739 Name-pl = Definitionen ,
2740 name-pl = Definitionen ,
2741
2742 type = proof ,
2743 Name-sg = Beweis ,
2744 name-sg = Beweis ,
2745 Name-pl = Beweise ,
2746 name-pl = Beweise ,
2747
2748 type = result ,
2749 Name-sg = Ergebnis ,
2750 name-sg = Ergebnis ,
2751 Name-pl = Ergebnisse ,
2752 name-pl = Ergebnisse ,
2753
2754 type = example ,
2755 Name-sg = Beispiel ,
2756 name-sg = Beispiel ,
2757 Name-pl = Beispiele ,
2758 name-pl = Beispiele ,
2759
2760 type = remark ,
2761 Name-sg = Bemerkung ,
2762 name-sg = Bemerkung ,
2763 Name-pl = Bemerkungen ,
2764 name-pl = Bemerkungen ,
2765
2766 type = algorithm ,

```

```

2767   Name-sg = Algorithmus ,
2768   name-sg = Algorithmus ,
2769   Name-pl = Algorithmen ,
2770   name-pl = Algorithmen ,
2771
2772   type = listing ,
2773   Name-sg = Listing , % CHECK
2774   name-sg = Listing , % CHECK
2775   Name-pl = Listings , % CHECK
2776   name-pl = Listings , % CHECK
2777
2778   type = exercise ,
2779   Name-sg = Übungsaufgabe ,
2780   name-sg = Übungsaufgabe ,
2781   Name-pl = Übungsaufgaben ,
2782   name-pl = Übungsaufgaben ,
2783
2784   type = solution ,
2785   Name-sg = Lösung ,
2786   name-sg = Lösung ,
2787   Name-pl = Lösungen ,
2788   name-pl = Lösungen ,
2789 </dict-german>

```

10.3 French

```

2790 <package>\zcDeclareLanguage { french }
2791 <package>\zcDeclareLanguageAlias { acadian } { french }
2792 <package>\zcDeclareLanguageAlias { canadien } { french }
2793 <package>\zcDeclareLanguageAlias { francais } { french }
2794 <package>\zcDeclareLanguageAlias { frenchb } { french }
2795 <*dict-french>
2796
2796 namesep = {\nobreakspace} ,
2797 pairsep = {\~et\nobreakspace} ,
2798 listsep = {,~} ,
2799 lastsep = {\~et\nobreakspace} ,
2800 tpairsep = {\~et\nobreakspace} ,
2801 tlistsep = {,~} ,
2802 tlastsep = {\~et\nobreakspace} ,
2803 notesep = {\~} ,
2804 rangesep = {\~à\nobreakspace} ,
2805
2806 type = part ,
2807   Name-sg = Partie ,
2808   name-sg = partie ,
2809   Name-pl = Parties ,
2810   name-pl = parties ,
2811
2812 type = chapter ,
2813   Name-sg = Chapitre ,
2814   name-sg = chapitre ,
2815   Name-pl = Chapitres ,
2816   name-pl = chapitres ,
2817

```

```

2818 type = section ,
2819     Name-sg = Section ,
2820     name-sg = section ,
2821     Name-pl = Sections ,
2822     name-pl = sections ,
2823
2824 type = paragraph ,
2825     Name-sg = Paragraphe ,
2826     name-sg = paragraphe ,
2827     Name-pl = Paragraphes ,
2828     name-pl = paragraphes ,
2829
2830 type = appendix ,
2831     Name-sg = Annexe ,
2832     name-sg = annexe ,
2833     Name-pl = Annexes ,
2834     name-pl = annexes ,
2835
2836 type = page ,
2837     Name-sg = Page ,
2838     name-sg = page ,
2839     Name-pl = Pages ,
2840     name-pl = pages ,
2841
2842 type = line ,
2843     Name-sg = Ligne ,
2844     name-sg = ligne ,
2845     Name-pl = Lignes ,
2846     name-pl = lignes ,
2847
2848 type = figure ,
2849     Name-sg = Figure ,
2850     name-sg = figure ,
2851     Name-pl = Figures ,
2852     name-pl = figures ,
2853
2854 type = table ,
2855     Name-sg = Table ,
2856     name-sg = table ,
2857     Name-pl = Tables ,
2858     name-pl = tables ,
2859
2860 type = item ,
2861     Name-sg = Point ,
2862     name-sg = point ,
2863     Name-pl = Points ,
2864     name-pl = points ,
2865
2866 type = footnote ,
2867     Name-sg = Note ,
2868     name-sg = note ,
2869     Name-pl = Notes ,
2870     name-pl = notes ,
2871

```

```

2872 type = note ,
2873     Name-sg = Note ,
2874     name-sg = note ,
2875     Name-pl = Notes ,
2876     name-pl = notes ,
2877
2878 type = equation ,
2879     Name-sg = Équation ,
2880     name-sg = équation ,
2881     Name-pl = Équations ,
2882     name-pl = équations ,
2883     refpre-in = {()} ,
2884     refpos-in = {} ,
2885
2886 type = theorem ,
2887     Name-sg = Théorème ,
2888     name-sg = théorème ,
2889     Name-pl = Théorèmes ,
2890     name-pl = théorèmes ,
2891
2892 type = lemma ,
2893     Name-sg = Lemme ,
2894     name-sg = lemme ,
2895     Name-pl = Lemmes ,
2896     name-pl = lemmes ,
2897
2898 type = corollary ,
2899     Name-sg = Corollaire ,
2900     name-sg = corollaire ,
2901     Name-pl = Corollaires ,
2902     name-pl = corollaires ,
2903
2904 type = proposition ,
2905     Name-sg = Proposition ,
2906     name-sg = proposition ,
2907     Name-pl = Propositions ,
2908     name-pl = propositions ,
2909
2910 type = definition ,
2911     Name-sg = Définition ,
2912     name-sg = définition ,
2913     Name-pl = Définitions ,
2914     name-pl = définitions ,
2915
2916 type = proof ,
2917     Name-sg = Démonstration ,
2918     name-sg = démonstration ,
2919     Name-pl = Démonstrations ,
2920     name-pl = démonstrations ,
2921
2922 type = result ,
2923     Name-sg = Résultat ,
2924     name-sg = résultat ,
2925     Name-pl = Résultats ,

```

```

2926   name-pl = résultats ,
2927
2928 type = example ,
2929   Name-sg = Exemple ,
2930   name-sg = exemple ,
2931   Name-pl = Exemples ,
2932   name-pl = exemples ,
2933
2934 type = remark ,
2935   Name-sg = Remarque ,
2936   name-sg = remarque ,
2937   Name-pl = Remarques ,
2938   name-pl = remarques ,
2939
2940 type = algorithm ,
2941   Name-sg = Algorithme ,
2942   name-sg = algorithme ,
2943   Name-pl = Algorithmes ,
2944   name-pl = algorithmes ,
2945
2946 type = listing ,
2947   Name-sg = Liste ,
2948   name-sg = liste ,
2949   Name-pl = Listes ,
2950   name-pl = listes ,
2951
2952 type = exercise ,
2953   Name-sg = Exercice ,
2954   name-sg = exercice ,
2955   Name-pl = Exercices ,
2956   name-pl = exercices ,
2957
2958 type = solution ,
2959   Name-sg = Solution ,
2960   name-sg = solution ,
2961   Name-pl = Solutions ,
2962   name-pl = solutions ,
2963 </dict-french>

```

10.4 Portuguese

```

2964 <package>\zcDeclareLanguage { portuguese }
2965 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
2966 <package>\zcDeclareLanguageAlias { brazil } { portuguese }
2967 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
2968 <*dict-portuguese>
2969 namesep = {\nobreakspace} ,
2970 pairsep = {\sim\nobreakspace} ,
2971 listsep = {,~} ,
2972 lastsep = {\sim\nobreakspace} ,
2973 tpairsep = {\sim\nobreakspace} ,
2974 tlistsep = {,~} ,
2975 tlastsep = {\sim\nobreakspace} ,
2976 notesep = {~} ,

```

```

2977 rangesep = {\~a\nobreakspace} ,
2978
2979 type = part ,
2980     Name-sg = Parte ,
2981     name-sg = parte ,
2982     Name-pl = Partes ,
2983     name-pl = partes ,
2984
2985 type = chapter ,
2986     Name-sg = Capítulo ,
2987     name-sg = capítulo ,
2988     Name-pl = Capítulos ,
2989     name-pl = capítulos ,
2990
2991 type = section ,
2992     Name-sg = Seção ,
2993     name-sg = seção ,
2994     Name-pl = Seções ,
2995     name-pl = seções ,
2996
2997 type = paragraph ,
2998     Name-sg = Parágrafo ,
2999     name-sg = parágrafo ,
3000     Name-pl = Parágrafos ,
3001     name-pl = parágrafos ,
3002     Name-sg-ab = Par. ,
3003     name-sg-ab = par. ,
3004     Name-pl-ab = Par. ,
3005     name-pl-ab = par. ,
3006
3007 type = appendix ,
3008     Name-sg = Apêndice ,
3009     name-sg = apêndice ,
3010     Name-pl = Apêndices ,
3011     name-pl = apêndices ,
3012
3013 type = page ,
3014     Name-sg = Página ,
3015     name-sg = página ,
3016     Name-pl = Páginas ,
3017     name-pl = páginas ,
3018     name-sg-ab = p. ,
3019     name-pl-ab = pp. ,
3020
3021 type = line ,
3022     Name-sg = Linha ,
3023     name-sg = linha ,
3024     Name-pl = Linhas ,
3025     name-pl = linhas ,
3026
3027 type = figure ,
3028     Name-sg = Figura ,
3029     name-sg = figura ,
3030     Name-pl = Figuras ,

```

```

3031 name-pl = figuras ,
3032 Name-sg-ab = Fig. ,
3033 name-sg-ab = fig. ,
3034 Name-pl-ab = Figs. ,
3035 name-pl-ab = figs. ,
3036
3037 type = table ,
3038 Name-sg = Tabela ,
3039 name-sg = tabela ,
3040 Name-pl = Tabelas ,
3041 name-pl = tabelas ,
3042
3043 type = item ,
3044 Name-sg = Item ,
3045 name-sg = item ,
3046 Name-pl = Itens ,
3047 name-pl = itens ,
3048
3049 type = footnote ,
3050 Name-sg = Nota ,
3051 name-sg = nota ,
3052 Name-pl = Notas ,
3053 name-pl = notas ,
3054
3055 type = note ,
3056 Name-sg = Nota ,
3057 name-sg = nota ,
3058 Name-pl = Notas ,
3059 name-pl = notas ,
3060
3061 type = equation ,
3062 Name-sg = Equação ,
3063 name-sg = equação ,
3064 Name-pl = Equações ,
3065 name-pl = equações ,
3066 Name-sg-ab = Eq. ,
3067 name-sg-ab = eq. ,
3068 Name-pl-ab = Eqs. ,
3069 name-pl-ab = eqs. ,
3070 refpre-in = {() ,
3071 refpos-in = {} } ,
3072
3073 type = theorem ,
3074 Name-sg = Teorema ,
3075 name-sg = teorema ,
3076 Name-pl = Teoremas ,
3077 name-pl = teoremas ,
3078
3079 type = lemma ,
3080 Name-sg = Lema ,
3081 name-sg = lema ,
3082 Name-pl = Lemas ,
3083 name-pl = lemas ,
3084

```



```

3085 type = corollary ,
3086     Name-sg = Corolário ,
3087     name-sg = corolário ,
3088     Name-pl = Corolários ,
3089     name-pl = corolários ,
3090
3091 type = proposition ,
3092     Name-sg = Proposição ,
3093     name-sg = proposição ,
3094     Name-pl = Proposições ,
3095     name-pl = proposições ,
3096
3097 type = definition ,
3098     Name-sg = Definição ,
3099     name-sg = definição ,
3100     Name-pl = Definições ,
3101     name-pl = definições ,
3102
3103 type = proof ,
3104     Name-sg = Demonstração ,
3105     name-sg = demonstração ,
3106     Name-pl = Demonstrações ,
3107     name-pl = demonstrações ,
3108
3109 type = result ,
3110     Name-sg = Resultado ,
3111     name-sg = resultado ,
3112     Name-pl = Resultados ,
3113     name-pl = resultados ,
3114
3115 type = example ,
3116     Name-sg = Exemplo ,
3117     name-sg = exemplo ,
3118     Name-pl = Exemplos ,
3119     name-pl = exemplos ,
3120
3121 type = remark ,
3122     Name-sg = Observação ,
3123     name-sg = observação ,
3124     Name-pl = Observações ,
3125     name-pl = observações ,
3126
3127 type = algorithm ,
3128     Name-sg = Algoritmo ,
3129     name-sg = algoritmo ,
3130     Name-pl = Algoritmos ,
3131     name-pl = algoritmos ,
3132
3133 type = listing ,
3134     Name-sg = Listagem ,
3135     name-sg = listagem ,
3136     Name-pl = Listagens ,
3137     name-pl = listagens ,
3138

```

```

3139 type = exercise ,
3140   Name-sg = Exercício ,
3141   name-sg = exercício ,
3142   Name-pl = Exercícios ,
3143   name-pl = exercícios ,
3144
3145 type = solution ,
3146   Name-sg = Solução ,
3147   name-sg = solução ,
3148   Name-pl = Soluções ,
3149   name-pl = soluções ,
3150 </dict-portuguese>

```

10.5 Spanish

```

3151 <package>\zcDeclareLanguage { spanish }
3152 <*dict-spanish>
3153 namesep = {\nobreakspace} ,
3154 pairsep = {\~y\nobreakspace} ,
3155 listsep = {,~} ,
3156 lastsep = {\~y\nobreakspace} ,
3157 tpairsep = {\~y\nobreakspace} ,
3158 tlistsep = {,~} ,
3159 tlastsep = {\~y\nobreakspace} ,
3160 notesep = {\~} ,
3161 rangesep = {\~a\nobreakspace} ,
3162
3163 type = part ,
3164   Name-sg = Parte ,
3165   name-sg = parte ,
3166   Name-pl = Partes ,
3167   name-pl = partes ,
3168
3169 type = chapter ,
3170   Name-sg = Capítulo ,
3171   name-sg = capítulo ,
3172   Name-pl = Capítulos ,
3173   name-pl = capítulos ,
3174
3175 type = section ,
3176   Name-sg = Sección ,
3177   name-sg = sección ,
3178   Name-pl = Secciones ,
3179   name-pl = secciones ,
3180
3181 type = paragraph ,
3182   Name-sg = Párrafo ,
3183   name-sg = párrafo ,
3184   Name-pl = Párrafos ,
3185   name-pl = párrafos ,
3186
3187 type = appendix ,
3188   Name-sg = Apéndice ,
3189   name-sg = apéndice ,

```

```

3190     Name-pl = Apéndices ,
3191     name-pl = apéndices ,
3192
3193     type = page ,
3194     Name-sg = Página ,
3195     name-sg = página ,
3196     Name-pl = Páginas ,
3197     name-pl = páginas ,
3198
3199     type = line ,
3200     Name-sg = Línea ,
3201     name-sg = línea ,
3202     Name-pl = Líneas ,
3203     name-pl = líneas ,
3204
3205     type = figure ,
3206     Name-sg = Figura ,
3207     name-sg = figura ,
3208     Name-pl = Figuras ,
3209     name-pl = figuras ,
3210
3211     type = table ,
3212     Name-sg = Cuadro ,
3213     name-sg = cuadro ,
3214     Name-pl = Cuadros ,
3215     name-pl = cuadros ,
3216
3217     type = item ,
3218     Name-sg = Punto ,
3219     name-sg = punto ,
3220     Name-pl = Puntos ,
3221     name-pl = puntos ,
3222
3223     type = footnote ,
3224     Name-sg = Nota ,
3225     name-sg = nota ,
3226     Name-pl = Notas ,
3227     name-pl = notas ,
3228
3229     type = note ,
3230     Name-sg = Nota ,
3231     name-sg = nota ,
3232     Name-pl = Notas ,
3233     name-pl = notas ,
3234
3235     type = equation ,
3236     Name-sg = Ecuación ,
3237     name-sg = ecuación ,
3238     Name-pl = Ecuaciones ,
3239     name-pl = ecuaciones ,
3240     refpre-in = { ( } ,
3241     refpos-in = { ) } ,
3242
3243     type = theorem ,

```

```

3244 Name-sg = Teorema ,
3245 name-sg = teorema ,
3246 Name-pl = Teoremas ,
3247 name-pl = teoremas ,
3248
3249 type = lemma ,
3250 Name-sg = Lema ,
3251 name-sg = lema ,
3252 Name-pl = Lemas ,
3253 name-pl = lemas ,
3254
3255 type = corollary ,
3256 Name-sg = Corolario ,
3257 name-sg = corolario ,
3258 Name-pl = Corolarios ,
3259 name-pl = corolarios ,
3260
3261 type = proposition ,
3262 Name-sg = Proposición ,
3263 name-sg = proposición ,
3264 Name-pl = Propositiones ,
3265 name-pl = proposiciones ,
3266
3267 type = definition ,
3268 Name-sg = Definición ,
3269 name-sg = definición ,
3270 Name-pl = Definiciones ,
3271 name-pl = definiciones ,
3272
3273 type = proof ,
3274 Name-sg = Demostración ,
3275 name-sg = demostración ,
3276 Name-pl = Demostraciones ,
3277 name-pl = demostraciones ,
3278
3279 type = result ,
3280 Name-sg = Resultado ,
3281 name-sg = resultado ,
3282 Name-pl = Resultados ,
3283 name-pl = resultados ,
3284
3285 type = example ,
3286 Name-sg = Ejemplo ,
3287 name-sg = ejemplo ,
3288 Name-pl = Ejemplos ,
3289 name-pl = ejemplos ,
3290
3291 type = remark ,
3292 Name-sg = Observación ,
3293 name-sg = observación ,
3294 Name-pl = Observaciones ,
3295 name-pl = observaciones ,
3296
3297 type = algorithm ,

```

```

3298   Name-sg = Algoritmo ,
3299   name-sg = algoritmo ,
3300   Name-pl = Algoritmos ,
3301   name-pl = algoritmos ,
3302
3303   type = listing ,
3304   Name-sg = Listado ,
3305   name-sg = listado ,
3306   Name-pl = Listados ,
3307   name-pl = listados ,
3308
3309   type = exercise ,
3310   Name-sg = Ejercicio ,
3311   name-sg = ejercicio ,
3312   Name-pl = Ejercicios ,
3313   name-pl = ejercicios ,
3314
3315   type = solution ,
3316   Name-sg = Solución ,
3317   name-sg = solución ,
3318   Name-pl = Soluciones ,
3319   name-pl = soluciones ,
3320 </dict-spanish>

```

Index

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

Symbols	
\\	103, 109, 120, 125, 126, 135, 145
A	
\AddToHook	91, 534, 549, 659, 727, 751, 780, 782, 833
\appendix	60
\appendixname	60
\Arg	2298
B	
\babelname	737
\babelprovide	22
bool commands:	
\bool_case_true:	2
\bool_if:N	288, 297, 663, 667, 1506, 1579, 1704, 1725, 1756, 1811, 1878, 1882, 1889, 1898, 1904, 2162
\bool_if:nTF	59, 1151, 1160, 1169, 1237, 1265, 1288, 1379, 1387, 1520, 1528, 1737, 1744, 1751, 2004, 2240
\bool_lazy_all:nTF	2305
\bool_lazy_and:nnTF	1038, 1053, 1853, 2066, 2351
\bool_lazy_any:nTF	2130, 2139
\bool_lazy_or:nnTF	1042, 2054
\bool_new:N	259, 570, 571, 596, 620, 629, 636, 637, 692, 693, 710, 711, 826, 827, 1064, 1083, 1419, 1420, 1431, 1432, 1433, 1451
\bool_set:Nn	1036
\bool_set_false:N	583, 587, 644, 653, 654, 669, 848, 1229, 1473, 1512, 1526, 1537, 1716, 1851, 1852, 2137, 2154
\bool_set_true:N	307, 577, 578, 582, 588, 643, 648, 649, 837, 842, 1249, 1260, 1277, 1283, 1300, 1306, 1334, 1346, 1480, 1507, 1513, 1517, 1538, 1541, 2153, 2377, 2384, 2385, 2403, 2410, 2411
\bool_until_do:Nn	1230, 1474

C

clist commands:	
\clist_map_inline:nn	479
\counterwithin	4
\cs	1358, 1702, 2301, 2347
cs commands:	
\cs_generate_variant:Nn	55, 56, 302, 311, 954, 962, 1084, 2045
\cs_if_exist:NTF	39, 48, 69
\cs_new:Npn	37, 46, 57, 67, 78, 2000, 2157
\cs_new_protected:Npn	262, 304, 314, 322, 440, 949, 957, 1031, 1085, 1101, 1144, 1210, 1354, 1410, 1454, 1590, 1846, 1996, 1998, 2046, 2299, 2345, 2365
\cs_new_protected:Npx	90
\cs_set_eq:NN	94

E

\endinput	12
exp commands:	
\exp_args:NNe	27
\exp_args:NNnx	252
\exp_args:NnV	280
\exp_args:NNx	95, 1274, 1297
\exp_args:Nnx	316
\exp_args:Nx	272
\exp_args:Nxx	1194, 1245, 2369, 2391, 2395
\exp_not:N	1758, 1761, 1781, 1784, 1787, 2007, 2010, 2013, 2025, 2027, 2030, 2033, 2038, 2040, 2043, 2160, 2168, 2186, 2189, 2191, 2194, 2200, 2207, 2209, 2213, 2216, 2219, 2221, 2227, 2231, 2234, 2246, 2249, 2252, 2275, 2277, 2280, 2283, 2289, 2291, 2294
\exp_not:n	1610, 1626, 1638, 1642, 1662, 1675, 1678, 1690, 1693, 1726, 1727, 1759, 1780, 1785, 1786, 1918, 1931, 1936, 1956, 1967, 1970, 1980, 1983, 2008, 2009, 2011, 2021, 2023, 2026, 2031, 2032, 2034, 2035, 2037, 2039, 2187, 2188, 2190, 2192, 2193, 2195, 2196, 2199, 2211, 2212, 2217, 2218, 2220, 2228, 2232, 2233, 2235, 2247, 2248, 2250, 2270, 2273, 2276, 2281, 2282, 2284, 2285, 2288, 2290
\ExplSyntaxOn	274

F

file commands:	
\file_get:nnNTF	272
\fmtversion	3

G

group commands:	
\group_begin:	93, 264, 306, 928, 1033, 1046, 1758, 1784, 2007, 2010, 2030, 2033, 2186, 2191, 2194, 2209, 2216, 2231, 2246, 2249, 2280, 2283
\group_end:	96, 300, 309, 936, 1049, 1061, 1781, 1787, 2025, 2027, 2038, 2040, 2189, 2200, 2207, 2213, 2219, 2234, 2275, 2277, 2289, 2291

H

\hyperlink	51
------------	----

I

\IfBooleanTF	1067
\IfFormatAtLeastTF	3, 4
int commands:	
\int_case:nnTF	1593, 1619, 1650, 1814, 1911, 1947
\int_compare:nNnTF	1198, 1250, 1319, 1335, 1365, 1367, 1412, 1561, 1606, 1640, 1803, 1805, 1868, 1892, 1934, 2373, 2379, 2399, 2405
\int_compare_p:nNn	1381, 1389, 2058, 2069, 2150
\int_eval:n	90
\int_incr:N	1841, 1881, 1883, 1897, 1899, 1903, 1905, 1994
\int_new:N	1081, 1082, 1426, 1427, 1428, 1429
\int_set:Nn	1366, 1368, 1372, 1375
\int_use:N	33, 35, 50
\int_zero:N	1356, 1357, 1462, 1463, 1464, 1465, 1840, 1842, 1843, 1989, 1990
iow commands:	
\iow_char:N	103, 109, 120, 125, 126, 135, 145
\iow_newline:	144, 148

K

keys commands:	
\keys_define:nn	26, 328, 340, 357, 371, 447, 475, 501, 525, 553, 560, 572, 597, 606, 621, 630, 638, 671, 678, 694, 712, 747, 785, 818, 821, 828, 838, 849, 860, 871, 889, 901, 939, 966, 987, 1010
\keys_set:nn	26, 30, 281, 843, 878, 884, 933, 1034
keyval commands:	
\keyval_parse:nnn	451, 505

L	
<code>\labelformat</code>	3
<code>\language</code>	21, 731
M	
<code>\mainbabelname</code>	21, 738
<code>\MessageBreak</code>	10
msg commands:	
<code>\msg_info:nnn</code>	348, 378
<code>\msg_line_context:</code>	102,
108, 139, 153, 157, 159, 161, 164, 168	
<code>\msg_new:nnn</code> 100, 106, 111, 113, 115,	
117, 122, 128, 130, 132, 137, 142,	
147, 149, 151, 156, 158, 160, 162, 167	
<code>\msg_note:nnn</code>	284
<code>\msg_warning:nn</code>	
..... 539, 564, 668, 674, 831, 852	
<code>\msg_warning:nnn</code>	
. 236, 255, 290, 298, 507, 893, 935,	
978, 1017, 1554, 1711, 2092, 2121, 2337	
<code>\msg_warning:nnnn</code>	
.... 238, 453, 1258, 1281, 1304, 1344	
N	
<code>\newcounter</code>	4
<code>\NewDocumentCommand</code>	
.. 228, 248, 877, 879, 926, 1029, 1065	
<code>\newtheorem</code>	60
<code>\nobreakspace</code>	430,
2428, 2429, 2431, 2432, 2434, 2436,	
2618, 2619, 2621, 2622, 2624, 2626,	
2796, 2797, 2799, 2800, 2802, 2804,	
2969, 2970, 2972, 2973, 2975, 2977,	
3153, 3154, 3156, 3157, 3159, 3161	
P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	6
prg commands:	
<code>\prg_generate_conditional_</code>	
variant:Nnn	398, 413
<code>\prg_new_protected_conditional:Npnn</code>	
..... 386, 401, 416	
<code>\prg_return_false:</code>	
..... 394, 396, 409, 411, 421	
<code>\prg_return_true:</code>	393, 408, 420
<code>\ProcessKeysOptions</code>	876
prop commands:	
<code>\prop_get:NnN</code>	2318
<code>\prop_get:NnNTF</code>	
.... 265, 388, 391, 403, 406, 418,	
929, 2080, 2098, 2103, 2302, 2348, 2358	
<code>\prop_gput:Nnn</code>	242, 252, 951, 959
<code>\prop_gput_if_new:Nnn</code>	316, 324
<code>\prop_gset_from_keyval:Nn</code>	424
<code>\prop_if_exist:NTF</code>	277, 881
<code>\prop_if_exist_p:N</code>	2309, 2354
<code>\prop_if_in:NnTF</code>	25, 232, 250
<code>\prop_if_in_p:Nn</code>	60, 2313
<code>\prop_item:Nn</code> ...	27, 61, 235, 239, 253
<code>\prop_new:N</code>	
.... 225, 278, 423, 446, 500, 856, 882	
<code>\prop_put:Nnn</code>	444, 867, 916
<code>\prop_remove:Nn</code>	443, 866, 908
<code>\providecommand</code>	3
<code>\ProvidesExplPackage</code>	14
R	
<code>\refstepcounter</code>	3
<code>\RequirePackage</code> ...	16, 17, 18, 19, 20, 664
S	
seq commands:	
<code>\seq_clear:N</code>	617, 1103
<code>\seq_const_from_clist:Nn</code>	
..... 171, 179, 192, 204	
<code>\seq_gconcat:NNN</code> ...	212, 215, 219, 222
<code>\seq_get_left:NN</code>	1482
<code>\seq_gput_right:Nn</code>	282
<code>\seq_if_empty:NTF</code>	1477
<code>\seq_if_in:NnTF</code>	268, 481, 1091
<code>\seq_map_break:n</code>	81, 1401, 1404
<code>\seq_map_function:NN</code>	1106
<code>\seq_map_indexed_inline:Nn</code> .	19, 1361
<code>\seq_map_inline:Nn</code>	337, 354,
368, 857, 886, 898, 963, 984, 1007, 1398	
<code>\seq_map_tokens:Nn</code>	63
<code>\seq_new:N</code>	211,
218, 258, 474, 605, 1063, 1100, 1421	
<code>\seq_pop_left:NN</code>	1476
<code>\seq_put_right:Nn</code>	483, 1095
<code>\seq_reverse:N</code>	611
<code>\seq_set_eq:NN</code>	1456
<code>\seq_set_from_clist:Nn</code>	610, 1035
<code>\seq_sort:Nn</code>	1109
sort commands:	
<code>\sort_return_same:</code>	
..... 33, 38, 1116, 1121, 1158,	
1203, 1205, 1255, 1261, 1278, 1284,	
1307, 1340, 1347, 1385, 1401, 1417	
<code>\sort_return_swapped:</code>	
..... 33, 38, 1129, 1167, 1202,	
1254, 1301, 1339, 1393, 1404, 1416	
str commands:	
<code>\str_case:nnTF</code>	753, 789
<code>\str_if_eq:nnTF</code>	80, 234
<code>\str_if_eq_p:nn</code>	2135, 2141, 2143, 2147
<code>\str_new:N</code>	677
<code>\str_set:Nn</code>	682, 684, 686, 688

T

TeX and L^AT_EX 2_ε commands:

<code>\@Alph</code>	60
<code>\@addtoreset</code>	4
<code>\@chapapp</code>	60
<code>\@currentcounter</code>	4, 21, 25, 28, 30, 33, 84, 86
<code>\@currentlabel</code>	3
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	536, 551, 661, 729, 735, 835
<code>\@onlypreamble</code>	245, 257, 938
<code>\bbl@loaded</code>	22
<code>\bbl@main@language</code>	21, 732
<code>\c@</code>	3
<code>\c@page</code>	6, 94
<code>\cl@</code>	4
<code>\hyper@@link</code>	1761, 2013, 2168, 2252
<code>\p@...</code>	3
<code>\zref@addprop</code>	22, 32, 34, 36, 87, 88, 99
<code>\zref@default</code>	1997, 1999
<code>\zref@extractdefault</code>	51, 1088, 1147, 1149, 1195, 1196, 1199, 1201, 1213, 1217, 1221, 1225, 1246, 1247, 1251, 1253, 1273, 1296, 1413, 1415, 1492, 1497, 1766, 1771, 1777, 2016, 2017, 2019, 2022, 2036, 2173, 2177, 2182, 2197, 2257, 2261, 2266, 2271, 2286, 2370, 2371, 2374, 2376, 2380, 2382, 2392, 2393, 2396, 2397, 2400, 2402, 2406, 2408
<code>\zref@ifpropundefined</code>	17
<code>\zref@ifrefcontainsprop</code>	17, 1763, 2002, 2015, 2164, 2170, 2237, 2254
<code>\zref@ifrefundefined</code>	1111, 1113, 1125, 1509, 1511, 1516, 1549, 1708, 1717, 1859, 2048, 2159
<code>\ZREF@mainlist</code>	22, 32, 34, 36, 87, 88, 99
<code>\zref@newprop</code>	4, 21, 23, 33, 35, 83, 85, 98
<code>\zref@refused</code>	1548
<code>\zref@wrapper@babel</code>	29, 1030
<code>\textendash</code>	434
<code>\the</code>	3
<code>\thechapter</code>	60
<code>\thepage</code>	6, 95
<code>\thesection</code>	60
tl commands:	
<code>\c_empty_tl</code>	1088, 1147, 1149, 1213, 1217, 1221, 1225, 1493, 1498
<code>\c_novalue_tl</code>	862, 903
<code>\tl_clear:N</code>	279, 333, 932, 944, 1457, 1458, 1459, 1460, 1461, 1479, 1836, 1837, 1838, 1839, 1880, 2049, 2052, 2076, 2091, 2120, 2336, 2360, 2362

<code>\tl_gset:Nn</code>	95
<code>\tl_head:N</code>	1233, 1235, 1320, 1322, 1336, 1338
<code>\tl_if_empty:N</code>	71, 345, 362, 376, 971, 992, 1015, 1089, 1552, 1706, 2063, 2078, 2225
<code>\tl_if_empty:n</code>	230, 332, 442, 943, 1658, 1673, 1688, 1929, 1954, 1965, 1978, 2051
<code>\tl_if_empty_p:N</code>	1155, 1156, 1164, 1165, 1172, 1173, 1523, 1524, 1531, 1532, 2134, 2144, 2148, 2307, 2352
<code>\tl_if_empty_p:n</code>	1241, 1242, 1269, 1292
<code>\tl_if_eq:NNTF</code>	1179, 1315, 1535
<code>\tl_if_eq:NnTF</code>	1104, 1137, 1371, 1374, 1400, 1403, 1484, 2367
<code>\tl_if_eq:nnTF</code>	1194, 1245, 1363, 2369, 2391, 2395
<code>\tl_if_in:NnTF</code>	1274, 1297
<code>\tl_if_novalue:n</code>	865, 906
<code>\tl_map_break:n</code>	81
<code>\tl_map_tokens:Nn</code>	73
<code>\tl_new:N</code>	89, 169, 170, 524, 724, 725, 726, 817, 820, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1422, 1423, 1424, 1425, 1430, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1452, 1453
<code>\tl_put_left:Nn</code>	1740, 1747, 1796
<code>\tl_put_right:Nn</code>	1608, 1624, 1633, 1660, 1670, 1685, 1916, 1927, 1952, 1963, 1976, 2064, 2065, 2074
<code>\tl_reverse_items:n</code>	1084, 1215, 1219, 1223, 1227
<code>\tl_set:Nn</code>	334, 529, 531, 537, 540, 556, 565, 731, 732, 737, 738, 741, 742, 745, 757, 765, 772, 793, 801, 808, 883, 945, 1087, 1146, 1148, 1212, 1214, 1216, 1218, 1220, 1222, 1224, 1226, 1232, 1234, 1272, 1295, 1324, 1326, 1328, 1330, 1486, 1487, 1490, 1495, 1599, 1600, 1723, 1754, 1872, 1873, 1896, 2060, 2061, 2073
<code>\tl_set_eq:NN</code>	1834
<code>\tl_tail:N</code>	1325, 1327, 1329, 1331
<code>\l_tmpa_tl</code>	275, 281, 1272, 1275
<code>\l_tmpb_tl</code>	1295, 1298

U

use commands:

<code>\use:N</code>	21
---------------------	----

Z

<code>\zcDeclareLanguage</code>	228, 245, 2419, 2610, 2790, 2964, 3151
<code>\zcDeclareLanguageAlias</code>	248, 257, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2611, 2612, 2613, 2614, 2615, 2616, 2791, 2792, 2793, 2794, 2965, 2966, 2967
<code>\zcDeclareTranslations</code> .	8, 9, 26, 27, 926
<code>\zcpageref</code>	31, 1065
<code>\zceref</code> 8, 25, 29, 32, 39, 40, 1029, 1068, 1069	
<code>\zcRefTypeSetup</code>	8, 9, 26, 879
<code>\zcsetup</code>	8, 21, 25, 877
zrefcheck commands:	
<code>\zrefcheck_zceref_beg_label:</code> ..	1041
<code>\zrefcheck_zceref_end_label_-</code>	
maybe:	1057
<code>\zrefcheck_zceref_run_checks_on_-</code>	
labels:n	1058
zrefclever internal commands:	
<code>\l__zrefclever_abbrev_bool</code>	710, 714, 2067
<code>\l__zrefclever_capitalize_bool</code> ..	692, 696, 2055
<code>\l__zrefclever_capitalize_first_-</code>	
bool	693, 702, 2057
<code>__zrefclever_counter_reset_by:n</code>	.. 5, 15, 16, 39, 41, 43, 48, 50, 52, 57
<code>__zrefclever_counter_reset_by_-</code>	
aux:nn	64, 67
<code>__zrefclever_counter_reset_by_-</code>	
aux:nnn	74, 78
<code>\l__zrefclever_counter_resetby_-</code>	
prop	4, 16, 60, 61, 500, 512
<code>\l__zrefclever_counter_resetters_-</code>	
seq	4, 15, 16, 63, 474, 481, 484
<code>\l__zrefclever_counter_type_prop</code> 3, 15, 25, 28, 446, 458
<code>\l__zrefclever_current_language_-</code>	
tl ..	21, 726, 731, 737, 741, 766, 802
<code>__zrefclever_declare_default_-</code>	
transl:nnn	957, 962, 973, 994
<code>__zrefclever_declare_type_-</code>	
transl:nnnn	949, 954, 999, 1021
<code>\l__zrefclever_dict_language_tl</code> 169, 266, 270, 273, 277, 278, 283, 285, 291, 317, 325, 389, 391, 404, 406, 930, 974, 995, 1000, 1022
<code>\g__zrefclever_fallback_dict_-</code>	
prop	418, 423, 424
<code>__zrefclever_get_default_-</code>	
transl:nnN	9, 401, 413
<code>__zrefclever_get_default_-</code>	
transl:nnNTF	2329
<code>__zrefclever_get_enclosing_-</code>	
counters:n	5, 37, 42, 84
<code>__zrefclever_get_enclosing_-</code>	
counters_value:n ...	5, 37, 51, 86
<code>__zrefclever_get_fallback_-</code>	
transl:nN	416
<code>__zrefclever_get_fallback_-</code>	
transl:nNTF	2334
<code>__zrefclever_get_ref:n</code> 1611, 1627,	1639, 1643, 1663, 1676, 1679, 1691, 1694, 1728, 1748, 1919, 1932, 1937, 1957, 1968, 1971, 1981, 1984, 2000
<code>__zrefclever_get_ref_first:</code> 40, 51, 1741, 1797, 2157
<code>__zrefclever_get_ref_font:nN</code> 8, 25, 1563, 1564, 1565, 2345
<code>__zrefclever_get_ref_string:nN</code> 8, 9, 14, 25, 1050, 1468, 1469, 1470, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 2298
<code>__zrefclever_get_type_transl:nnnN</code> 9, 386, 398
<code>__zrefclever_get_type_transl:nnnNTF</code> 2085, 2108, 2114, 2323
<code>\l__zrefclever_label_a_tl</code> 1071, 1476, 1493, 1509, 1548, 1549, 1555, 1599, 1611, 1627, 1643, 1679, 1694, 1721, 1728, 1859, 1863, 1872, 1896, 1919, 1937, 1971, 1984
<code>\l__zrefclever_label_b_tl</code> .	1071, 1479, 1482, 1498, 1511, 1516, 1863
<code>\l__zrefclever_label_count_int</code> 39, 1426, 1462, 1561, 1593, 1840, 1868, 1994
<code>\l__zrefclever_label_enclcnt_a_-</code>	
tl	1071, 1212, 1214, 1215, 1233, 1298, 1324, 1325
<code>\l__zrefclever_label_enclcnt_b_-</code>	
tl	1071, 1216, 1218, 1219, 1235, 1275, 1326, 1327
<code>\l__zrefclever_label_enclhead_a_-</code>	
tl	1071, 1232, 1241, 1269, 1316
<code>\l__zrefclever_label_enclhead_b_-</code>	
tl	1071, 1234, 1242, 1292, 1317
<code>\l__zrefclever_label_enclval_a_-</code>	
tl	1071, 1220, 1222, 1223, 1320, 1328, 1329, 1336
<code>\l__zrefclever_label_enclval_b_-</code>	
tl	1071, 1224, 1226, 1227, 1322, 1330, 1331, 1338
<code>\l__zrefclever_label_type_a_tl</code> 1071, 1087, 1089, 1093, 1096, 1146, 1155, 1164, 1172, 1180, 1371, 1400, 1486, 1490, 1523, 1531,

1536, 1552, 1600, 1873, 2307, 2310,
2314, 2319, 2325, 2352, 2355, 2359
\l_zrefclever_label_type_b_tl ..
..... 1071,
1148, 1156, 1165, 1173, 1181, 1374,
1403, 1487, 1495, 1524, 1532, 1536
_zrefclever_label_type_put_-
new_right:n 31, 1085, 1107
\l_zrefclever_label_types_seq ..
.... 31, 1092, 1095, 1100, 1103, 1398
_zrefclever_labels_in_sequence:nn
..... 1720, 1862, 2365
\g_zrefclever_language_aliases_-
prop 225, 232, 235, 239,
242, 250, 252, 253, 265, 388, 403, 929
\l_zrefclever_last_of_type_bool
..... 39, 1419, 1507, 1512, 1513,
1517, 1526, 1537, 1538, 1541, 1579
\l_zrefclever_lastsep_tl . 1441,
1570, 1626, 1642, 1662, 1678, 1690
\l_zrefclever_link_star_bool ...
..... 1036, 1063, 2005, 2133, 2243
\l_zrefclever_listsep_tl
... 1440, 1569, 1638, 1675, 1918,
1931, 1936, 1956, 1967, 1970, 1980
\l_zrefclever_load_dict_-
verbose_bool ... 259, 288, 297, 307
\g_zrefclever_loaded_dictionaries_-
seq 258, 269, 282
\l_zrefclever_main_language_tl .
. 21, 725, 732, 738, 742, 745, 758, 794
_zrefclever_name_default:
..... 51, 1998, 2227
\l_zrefclever_name_format_-
fallback_tl
.. 1450, 2073, 2076, 2078, 2105, 2117
\l_zrefclever_name_format_tl ...
... 1450, 2060, 2061, 2064, 2065,
2073, 2074, 2082, 2088, 2100, 2111
\l_zrefclever_name_in_link_bool
.. 1450, 1756, 2137, 2153, 2154, 2162
\l_zrefclever_namefont_tl 1434,
1563, 1759, 1785, 2187, 2217, 2232
\l_zrefclever_nameinlink_str ...
..... 677, 682,
684, 686, 688, 2135, 2141, 2143, 2147
\l_zrefclever_namesep_tl
.. 1437, 1566, 2190, 2220, 2228, 2235
\l_zrefclever_next_is_same_bool
..... 40, 58, 1428,
1852, 1882, 1898, 1904, 2385, 2411
\l_zrefclever_next_maybe_range_-
bool
.. 40, 58, 1428, 1716, 1725, 1851,
1878, 1889, 2377, 2384, 2403, 2410
\l_zrefclever_noabbrev_first_-
bool 711, 720, 2070
\l_zrefclever_noteseq_tl
..... 1050, 1051, 1445
_zrefclever_page_format_aux: ..
..... 90, 94
\g_zrefclever_page_format_tl ...
..... 6, 89, 95, 98
\l_zrefclever_pairsep_tl
..... 1439, 1568, 1610, 1726
_zrefclever_prop_put_non_-
empty:Nnn 14, 440, 457, 511
_zrefclever_provide_dict_-
default_transl:nn .. 322, 346, 363
_zrefclever_provide_dict_type_-
transl:nn 314, 364, 381
_zrefclever_provide_dictionary:n
..... 9, 30, 262, 302, 308, 784, 1037
_zrefclever_provide_dictionary_-
verbose:n 304,
311, 759, 767, 773, 795, 803, 809
\l_zrefclever_range_beg_label_-
tl 40, 1428, 1461,
1639, 1658, 1663, 1673, 1676, 1688,
1691, 1839, 1880, 1896, 1929, 1932,
1954, 1957, 1965, 1968, 1978, 1981
\l_zrefclever_range_count_int ..
..... 40,
1428, 1464, 1619, 1651, 1842, 1881,
1893, 1897, 1903, 1911, 1948, 1989
\l_zrefclever_range_inhibit_-
next_bool 39, 40, 1428, 1857
\l_zrefclever_range_same_count_-
int 40,
1428, 1465, 1606, 1640, 1651, 1843,
1883, 1899, 1905, 1934, 1948, 1990
\l_zrefclever_rangesep_tl
..... 1438, 1567, 1693, 1727, 1983
_zrefclever_ref_default:
.... 51, 1996, 2043, 2160, 2221, 2294
\l_zrefclever_ref_language_tl ..
..... 21, 22,
724, 745, 757, 760, 765, 768, 772,
774, 784, 793, 796, 801, 804, 808,
810, 1037, 2086, 2109, 2115, 2324, 2330
\c_zrefclever_ref_options_font_-
seq 10, 171
\c_zrefclever_ref_options_-
necessarily_not_type_specific_-
seq 14, 171, 338, 887, 964
\c_zrefclever_ref_options_-
necessarily_type_specific_seq

..... [171](#), [369](#), [1008](#)
 \c_zrefclever_ref_options_-
 possibly_type_specific_seq ..
 [14](#), [171](#), [355](#), [985](#)
 \l_zrefclever_ref_options_prop ..
 [25](#), [26](#), [856](#), [866](#), [867](#), [2302](#), [2348](#)
 \c_zrefclever_ref_options_-
 reference_seq [171](#), [858](#)
 \c_zrefclever_ref_options_-
 typesetup_seq [171](#), [899](#)
 \l_zrefclever_ref_property_tl ..
 [17](#),
 [524](#), [529](#), [531](#), [537](#), [540](#), [556](#), [565](#),
 [1104](#), [1137](#), [1484](#), [2002](#), [2022](#), [2036](#),
 [2165](#), [2198](#), [2238](#), [2272](#), [2287](#), [2367](#)
 \l_zrefclever_ref_typeset_font_-
 tl [817](#), [819](#), [1047](#)
 \l_zrefclever_reffont_in_tl [1436](#),
 [1565](#), [2011](#), [2034](#), [2195](#), [2250](#), [2284](#)
 \l_zrefclever_reffont_out_tl ...
 [1435](#), [1564](#),
 [2008](#), [2031](#), [2192](#), [2211](#), [2247](#), [2281](#)
 \l_zrefclever_refpos_in_tl [1449](#),
 [1574](#), [2023](#), [2037](#), [2199](#), [2273](#), [2288](#)
 \l_zrefclever_refpos_out_tl [1447](#),
 [1572](#), [2026](#), [2039](#), [2212](#), [2276](#), [2290](#)
 \l_zrefclever_refpre_in_tl [1448](#),
 [1573](#), [2021](#), [2035](#), [2196](#), [2270](#), [2285](#)
 \l_zrefclever_refpre_out_tl [1446](#),
 [1571](#), [2009](#), [2032](#), [2193](#), [2248](#), [2282](#)
 \l_zrefclever_setup_type_tl ...
 [169](#), [279](#), [318](#), [333](#), [334](#),
 [345](#), [362](#), [376](#), [883](#), [911](#), [919](#), [932](#),
 [944](#), [945](#), [971](#), [992](#), [1001](#), [1015](#), [1023](#)
 \l_zrefclever_sort_decided_bool
 ... [1083](#), [1229](#), [1230](#), [1249](#), [1260](#),
 [1277](#), [1283](#), [1300](#), [1306](#), [1334](#), [1346](#)
 _zrefclever_sort_default:nn ...
 [31–33](#), [1139](#), [1144](#)
 _zrefclever_sort_default_-
 different_types:nn . [19](#), [1188](#), [1354](#)
 _zrefclever_sort_default_same_-
 type:nn [1184](#), [1210](#)
 _zrefclever_sort_labels:
 [31](#), [33](#), [38](#), [1045](#), [1101](#)
 _zrefclever_sort_page:nn
 [38](#), [1138](#), [1410](#)
 \l_zrefclever_sort_prior_a_int ..
 [1081](#),
 [1356](#), [1365](#), [1366](#), [1372](#), [1382](#), [1390](#)
 \l_zrefclever_sort_prior_b_int ..
 [1082](#),
 [1357](#), [1367](#), [1368](#), [1375](#), [1383](#), [1391](#)

\l_zrefclever_tlastsep_tl
 [1444](#), [1470](#), [1828](#)
 \l_zrefclever_tlistsep_tl
 [1443](#), [1469](#), [1806](#)
 \l_zrefclever_tpairsep_tl
 [1442](#), [1468](#), [1822](#)
 \l_zrefclever_type_<type>-
 options_prop [26](#)
 \l_zrefclever_type_count_int ...
 [39](#), [1426](#), [1463](#), [1803](#),
 [1805](#), [1814](#), [1841](#), [2058](#), [2069](#), [2150](#)
 \l_zrefclever_type_first_label_-
 tl [1421](#), [1459](#), [1599](#), [1708](#),
 [1717](#), [1721](#), [1748](#), [1764](#), [1767](#), [1772](#),
 [1778](#), [1837](#), [1872](#), [2048](#), [2159](#), [2165](#),
 [2171](#), [2173](#), [2177](#), [2182](#), [2197](#), [2238](#),
 [2255](#), [2257](#), [2261](#), [2266](#), [2271](#), [2286](#)
 \l_zrefclever_type_first_label_-
 type_tl [1421](#), [1460](#), [1600](#),
 [1712](#), [1838](#), [1873](#), [2051](#), [2081](#), [2087](#),
 [2093](#), [2099](#), [2104](#), [2110](#), [2116](#), [2122](#)
 _zrefclever_type_name_setup: ..
 [9](#), [40](#), [1736](#), [2046](#)
 \l_zrefclever_type_name_tl . [52](#),
 [1450](#), [1780](#), [1786](#), [2049](#), [2052](#), [2083](#),
 [2089](#), [2091](#), [2101](#), [2106](#), [2112](#), [2118](#),
 [2120](#), [2134](#), [2188](#), [2218](#), [2225](#), [2233](#)
 \l_zrefclever_typeset_compress_-
 bool [620](#), [623](#), [1854](#)
 \l_zrefclever_typeset_labels_-
 seq ... [1421](#), [1456](#), [1476](#), [1477](#), [1482](#)
 \l_zrefclever_typeset_last_bool
 [39](#), [1419](#),
 [1473](#), [1474](#), [1480](#), [1506](#), [1811](#), [2149](#)
 \l_zrefclever_typeset_name_bool
 [571](#), [578](#), [583](#), [588](#), [1738](#), [1752](#)
 \l_zrefclever_typeset_queue_-
 curr_tl [1421](#), [1458](#), [1608](#),
 [1624](#), [1633](#), [1660](#), [1670](#), [1685](#), [1706](#),
 [1723](#), [1740](#), [1747](#), [1754](#), [1797](#), [1818](#),
 [1823](#), [1829](#), [1835](#), [1836](#), [1916](#), [1927](#),
 [1952](#), [1963](#), [1976](#), [2063](#), [2144](#), [2148](#)
 \l_zrefclever_typeset_queue_-
 prev_tl [1421](#), [1457](#), [1807](#), [1835](#)
 \l_zrefclever_typeset_range_-
 bool [629](#), [632](#), [1044](#), [1704](#)
 \l_zrefclever_typeset_ref_bool ..
 [570](#), [577](#), [582](#), [587](#), [1738](#), [1745](#)
 _zrefclever_typeset_refs:
 [39](#), [40](#), [51](#), [52](#), [54](#), [1048](#), [1454](#)
 _zrefclever_typeset_refs_aux_-
 last_of_type: [1582](#), [1590](#)
 _zrefclever_typeset_refs_aux_-
 not_last_of_type: [1586](#), [1846](#)

\l__zrefclever_typeset_sort_bool	__zrefclever_zcref:nnnn	29, 32, 1031
..... 596, 599, 1043	\l__zrefclever_zcref_labels_seq	.
\l__zrefclever_typesort_seq 32, 1035, 1059, 1063 , 1106, 1109, 1456	
..... 19, 605, 610, 611, 617, 1361	\l__zrefclever_zcref_note_tl	...
\l__zrefclever_use_hyperref_bool 820, 823, 1052	
..... 636, 643,	\l__zrefclever_zcref_with_check_-	
648, 653, 663, 669, 2005, 2132, 2242	bool 827, 842, 1040, 1055
\l__zrefclever_warn_hyperref_-	\l__zrefclever_zrefcheck_-	
bool	available_bool
__zrefclever_zcref:nnn 826, 837, 848, 1039, 1054	
.. 1030, 1031		