

The zref-clever package implementation*

Gustavo Barros[†]

2021-09-13

Contents

1	Initial setup	2
2	Dependencies	2
3	zref setup	3
4	Plumbing	7
4.1	Messages	7
4.2	Reference format	8
4.3	Languages	10
4.4	Dictionaries	11
4.5	Options	17
5	Configuration	28
5.1	\zcsetup	28
5.2	\zcRefTypeSetup	29
5.3	\zcDeclareTranslations	30
6	User interface	32
6.1	\zceref	32
6.2	\zcpageref	34
7	Sorting	34
8	Typesetting	42
9	Special handling	64
9.1	\appendix	64
9.2	\newtheorem	65
9.3	enumitem package	65

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

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

10	Dictionaries	65
10.1	English	65
10.2	German	68
10.3	French	72
10.4	Portuguese	75
10.5	Spanish	79
Index		82

1 Initial setup

Start the DocStrip guards.

```
1 <*package>
```

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

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

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

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

Identify the package.

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

2 Dependencies

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

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

3 zref setup

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

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

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

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

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

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

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

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

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

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

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

Though counters can be reset at any time, and in different ways at that, the most important use case is the automatic resetting of counters when some other counter is stepped, as performed by the standard mechanisms of the kernel (optional argument of `\newcounter`, `\@addtoreset`, `\counterwithin`, and related infrastructure). The canonical optional argument of `\newcounter` establishes that the counter being created (the mandatory argument) gets reset every time the “enclosing counter” gets stepped (this is called in the usual sources “within-counter”, “old counter”, “supercounter” etc.). This information is a little trickier to get. For starters, the counters which may reset the current counter are not retrievable from the counter itself, because this information is stored with the counter that does the resetting, not with the one that gets reset (the list is stored in `\cl@<counter>` with format `\@elt{countera}\@elt{counterb}\@elt{counterc}`, see section ‘ltcounts.dtx’ in ‘source2e’). Besides, there may be a chain of resetting counters, which must be taken into account: if ‘counterC’ gets reset by ‘counterB’, and ‘counterB’ gets reset by ‘counterA’, stepping the latter affects all three of them.

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

```
\__zrefclever_get_enclosing_counters:n
__zrefclever_get_enclosing_counters_value:n
```

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

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

```

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

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

```

Both e and f expansions work for this particular recursive call. I'll stay with the e variant, since conceptually it is what I want (x itself is not expandable), and this package is anyway not compatible with older kernels for which the performance penalty of the e expansion would ensue (see also https://tex.stackexchange.com/q/611370/#comment1529282_611385, thanks Enrico Gregorio, aka 'egreg').

```

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

```

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

_zrefclever_counter_reset_by:n Auxiliary function for _zrefclever_get_enclosing_counters:n and _zrefclever_get_enclosing_counters_value:n. They are broken in parts to be able to use the expandable mapping functions. _zrefclever_counter_reset_by:n leaves in the stream the “enclosing counter” which resets `\counter`.

```

    \_zrefclever_counter_reset_by:n {\counter}

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

```

```

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

```

(End definition for `__zrefclever_counter_reset_by:n`.)

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

```

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

```

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

```

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

```

```

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

```

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

4 Plumbing

4.1 Messages

```

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

```

```

142 \msg_new:nnn { zref-clever } { hyperref-preamble-only }
143 {
144   Option~'hyperref'~only~available~in~the~preamble.~
145   Use~the~starred~version~of~'\iow_char:N\zcref'~instead.
146 }
147 \msg_new:nnn { zref-clever } { missing-hyperref }
148 { Missing~'hyperref'~package.~Setting~'hyperref=false'. }
149 \msg_new:nnn { zref-check } { check-document-only }
150 { Option~'check'~only~available~in~the~document. }
151 \msg_new:nnn { zref-clever } { missing-zref-check }
152 {
153   Option~'check'~requested~\msg_line_context:~
154   But~package~'zref-check'~is~not~loaded,~can't~run~the~checks.
155 }
156 \msg_new:nnn { zref-clever } { counters-not-nested }
157 { Counters~not~nested~for~labels~'#1'~and~'#2'~\msg_line_context:. }
158 \msg_new:nnn { zref-clever } { missing-type }
159 { Reference~type~undefined~for~label~'#1'~\msg_line_context:. }
160 \msg_new:nnn { zref-clever } { missing-name }
161 { Name~undefined~for~type~'#1'~\msg_line_context:. }
162 \msg_new:nnn { zref-clever } { missing-string }
163 {
164   We~couldn't~find~a~value~for~reference~option~'#1'~\msg_line_context:~
165   But~we~should~have:~throw~a~rock~at~the~maintainer.
166 }
167 \msg_new:nnn { zref-clever } { single-element-range }
168 { Range~for~type~'#1'~resulted~in~single~element~\msg_line_context:. }

```

4.2 Reference format

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

With that in mind, we have settled with an user interface for reference formatting which allows settings to be done in different scopes, with more or less overarching effects, and some precedence rules to regulate the relation of settings given in each of these scopes. There are four scopes in which reference formatting can be specified by the user, in the following precedence order: i) as general *options*; ii) as *type-specific options*; iii) as *language-specific and type-specific translations*; and iv) as *default translations* (that is, language-specific but not type-specific). Besides those, there’s actually a fifth *internal* scope, with the least priority of all, a “fallback”, for the cases where it is meaningful to provide some value, even for an unknown language. These precedence rules are handled / enforced in `__zrefclever_get_ref_string:nN`, `__zrefclever_get_ref_font:nN`, and `__zrefclever_type_name_setup`: which are the basic functions to retrieve proper values for reference format settings.

General “options” (i) can be given by the user in the optional argument of `\zcref`, but just as well in `\zcsetup` or as package options at load-time (see Section 4.5).

“Type-specific options” (ii) are handled by `\zcRefTypeSetup`. “Language-specific translations”, be they “type-specific” (iii) or “default” (iv) have their user interface in `\zcDeclareTranslations`, and have their values populated by the package’s dictionaries. The “fallback” settings are stored in `\g__zrefclever_fallback_dict_prop`.

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

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

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

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

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

`f_options_necessarily_not_type_specific_seq` Lists of reference format related options in “categories”. Since these options are set in different scopes, and at different places, storing the actual lists in centralized variables makes the job not only easier later on, but also keeps things consistent.

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

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

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

```

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

```

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

```

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

```

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

4.3 Languages

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

```

225 \prop_new:N \g__zrefclever_languages_prop

```

(End definition for `\g__zrefclever_languages_prop`.)

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

```

\zcDeclareLanguage {\langle language \rangle}

```

```

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

```

(End definition for `\zcDeclareLanguage`.)

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

```

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

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

```

(End definition for `\zcDeclareLanguageAlias`.)

4.4 Dictionaries

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

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

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

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

Provide

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

```

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

```

(End definition for __zrefclever_provide_dictionary:n.)

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

```

\__zrefclever_provide_dictionary_verbose:n {<language>}

303 \cs_new_protected:Npn \__zrefclever_provide_dictionary_verbose:n #1
304 {
305   \group_begin:

```

```

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

```

(End definition for __zrefclever_provide_dictionary_verbose:n.)

__zrefclever_provide_dict_type_transl:nn
__zrefclever_provide_dict_default_transl:nn

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

```

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

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

```

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

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

```

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

```

```

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

```

Fallback

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

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

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

```

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

```

Get translations

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

```

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

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

```

(End definition for `_zrefclever_get_type_transl:nnnNF`.)

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

```

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

```



```

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

```

(End definition for __zrefclever_get_default_transl:nnNF.)

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

```

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

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

```

(End definition for __zrefclever_get_fallback_transl:nNF.)

4.5 Options

Auxiliary

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

```

\__zrefclever_prop_put_non_empty:Nnn <property list> {<key>} {<value>}

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

```

(End definition for __zrefclever_prop_put_non_empty:Nnn.)

countertype option

`\l__zrefclever_counter_type_prop` is used by `zc@type` property, and stores a mapping from “counter” to “reference type”. Only those counters whose type name is different from that of the counter need to be specified, since `zc@type` presumes the counter as the type if the counter is not found in `\l__zrefclever_counter_type_prop`.

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

counterresetters option

`\l__zrefclever_counter_resetters_seq` is used by `__zrefclever_counter_reset_by:n` to populate the `zc@enclcnt` and `zc@enclval` properties, and stores the list of counters which are potential “enclosing counters” for other counters. This option is constructed such that users can only *add* items to the variable. There would be little gain and some risk in allowing removal, and the syntax of the option would become unnecessarily more complicated. Besides, users can already override, for any particular counter, the search done from the set in `\l__zrefclever_counter_resetters_seq` with the `counterresetby` option.

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

```

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

```

counterresetby option

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

```

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

```

The counters for the `enumerate` environment do not use the regular counter machinery for resetting on each level, but are nested nevertheless by other means, treat them as exception.

```

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

```

ref option

`\l__zrefclever_ref_property_tl` stores the property to which the reference is being made. Currently, we restrict `ref=` to these two (or three) alternatives – `zc@thecnt`, `page`, and `title` if `zref-titleref` is loaded –, but there might be a case for making this more flexible. The infrastructure can already handle receiving an arbitrary property, as long as one is satisfied with sorting and compressing from the default counter. If more flexibility is granted, one thing *must* be handled at this point: the existence of the property itself, as far as `zref` is concerned. This because typesetting relies on the check `\zref@ifrefcontainsprop`, which *presumes* the property is defined and silently expands the *true* branch if it is not (see <https://github.com/ho-tex/zref/issues/13>, thanks Ulrike Fischer). Therefore, before adding anything to `\l__zrefclever_ref_property_tl`, check if first here with `\zref@ifpropundefined`: close it at the door.

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

```

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

```

typeset option

```

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

```

sort option

```

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

```

typesort option

\l__zrefclever_typesort_seq is stored reversed, since the sort priorities are computed in the negative range in __zrefclever_sort_default_different_types:nn, so that we can implicitly rely on ‘0’ being the “last value”, and spare creating an integer variable using \seq_map_indexed_inline:Nn.

```

603 \seq_new:N \l__zrefclever_typesort_seq

```

```

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

```

comp option

```

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

```

range option

```

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

```

hyperref option

```

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

```

```

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

```

nameinlink option

```

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

```

cap and capfirst options

```

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

```

```

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

```

abbrev and noabbrevfirst options

```

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

```

lang option

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

The overall setup here seems a little roundabout, but this is actually required. In the preamble, we (potentially) don’t yet have values for the “main” and “current” document languages, this must be retrieved at a `begindocument` hook. The `begindocument` hook is responsible to get values for `\l__zrefclever_main_language_tl` and `\l__zrefclever_current_language_tl`, and to set the default for `\l__zrefclever_ref_language_tl`. Package options, or preamble calls to `\zcsetup` are also hooked at `begindocument`, but come after the first hook, so that the pertinent variables have been set when they are executed. Finally, we set a third `begindocument` hook, at `begindocument/before`, so that it runs after any options set in the preamble. This hook redefines the `lang` option for immediate execution in the document body, and ensures the main language’s dictionary gets loaded, if it hadn’t been already.

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

```

722 \tl_new:N \l__zrefclever_ref_language_tl

```



```

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

```

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

```

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

```

```

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

```

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

```

783         \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }

```

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

```

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

```

font option

```

816 \tl_new:N \l__zrefclever_ref_typeset_font_tl

```

```

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

```

note option

```

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

```

check option

Integration with zref-check.

```

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

```

Reference options

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

The keys are set so that any value, including an empty one, is added to `\l__zrefclever_ref_options_prop`, while a key with *no value* removes the property from the list, so that these options can then fall back to lower precedence levels settings. For discussion about the used technique, see Section 5.2.

```

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

```

Package options

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

```

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

```

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

```

875 \ProcessKeysOptions { zref-clever / zcsetup }

```

5 Configuration

5.1 \zcsetup

`\zcsetup` Provide `\zcsetup`.

```

\zcsetup{<options>}

```

```

876 \NewDocumentCommand \zcsetup { m }
877 { \keys_set:nn { zref-clever / zcsetup } {#1} }

```

(End definition for `\zcsetup`.)

5.2 \zcRefTypeSetup

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

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

(End definition for `\zcRefTypeSetup`.)

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

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

```

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

```

5.3 \zcDeclareTranslations

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

```

\zcDeclareTranslations      \zcDeclareTranslations{<language>}{<options>}

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

```

(End definition for \zcDeclareTranslations.)

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

```

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

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

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

```

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

```

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

```

```

982 {
983   \keys_define:nn { zref-clever / translations }
984   {
985     #1 .value_required:n = true ,
986     #1 .code:n =
987     {
988       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
989       {
990         \__zrefclever_declare_default_transl:Vnn
991         \l__zrefclever_dict_language_tl
992         {#1} {##1}
993       }
994       {
995         \__zrefclever_declare_type_transl:Vnn
996         \l__zrefclever_dict_language_tl
997         \l__zrefclever_setup_type_tl
998         {#1} {##1}
999       }
1000     } ,
1001   }
1002 }
1003 \seq_map_inline:Nn
1004 \c__zrefclever_ref_options_necessarily_type_specific_seq
1005 {
1006   \keys_define:nn { zref-clever / translations }
1007   {
1008     #1 .value_required:n = true ,
1009     #1 .code:n =
1010     {
1011       \tl_if_empty:NTF \l__zrefclever_setup_type_tl
1012       {
1013         \msg_warning:nnn { zref-clever }
1014         { option-only-type-specific } {#1}
1015       }
1016       {
1017         \__zrefclever_declare_type_transl:Vnn
1018         \l__zrefclever_dict_language_tl
1019         \l__zrefclever_setup_type_tl
1020         {#1} {##1}
1021       }
1022     } ,
1023   }
1024 }

```

6 User interface

6.1 \zcref

```

\zcref      \zcref<*>[<options>]{<labels>}
1025 \NewDocumentCommand \zcref { s O { } m }
1026 { \zref@wrapper@babel \__zrefclever_zcref:nnn {#3} {#1} {#2} }

```

(End definition for \zcref.)

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

```

    \__zrefclever_zcref:nnnn {\labels} {\(*)} {\options}}
1027 \cs_new_protected:Npn \__zrefclever_zcref:nnn #1#2#3
1028 {
1029     \group_begin:

```

Set options.

```
1030     \keys_set:nn { zref-clever / reference } {#3}
```

Store arguments values.

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

```

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

```
1033     \__zrefclever_provide_dictionary:x { \l__zrefclever_ref_language_tl }
```

Integration with `zref-check`.

```
1034     \bool_lazy_and:nnT
1035     { \l__zrefclever_zrefcheck_available_bool }
1036     { \l__zrefclever_zcref_with_check_bool }
1037     { \zrefcheck_zcref_beg_label: }

```

Sort the labels.

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

```

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

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

```

Typeset note.

```
1046     \__zrefclever_get_ref_string:nN {notesep} \l__zrefclever_notesep_tl
1047     \l__zrefclever_notesep_tl
1048     \l__zrefclever_zcref_note_tl

```

Integration with `zref-check`.

```
1049     \bool_lazy_and:nnT
1050     { \l__zrefclever_zrefcheck_available_bool }
1051     { \l__zrefclever_zcref_with_check_bool }
1052     {
1053         \zrefcheck_zcref_end_label_maybe:
1054         \zrefcheck_zcref_run_checks_on_labels:n
1055         { \l__zrefclever_zcref_labels_seq }
1056     }
1057     \group_end:
1058 }

```

(End definition for `_zrefclever_zcref:nnnn`.)

```
\l_zrefclever_zcref_labels_seq
\l_zrefclever_link_star_bool
1059 \seq_new:N \l__zrefclever_zcref_labels_seq
1060 \bool_new:N \l__zrefclever_link_star_bool
```

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

6.2 `\zcpageref`

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

(End definition for `\zcpageref`.)

7 Sorting

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

```
\l__zrefclever_label_a_tl
\l__zrefclever_label_b_tl
\l_zrefclever_label_type_a_tl
\l_zrefclever_label_type_b_tl
\l_zrefclever_label_enclcnt_a_tl
\l_zrefclever_label_enclcnt_b_tl
\l_zrefclever_label_enclval_a_tl
\l_zrefclever_label_enclval_b_tl
\l_zrefclever_label_enclhead_a_tl
\l_zrefclever_label_enclhead_b_tl
1067 \tl_new:N \l__zrefclever_label_a_tl
1068 \tl_new:N \l__zrefclever_label_b_tl
1069 \tl_new:N \l_zrefclever_label_type_a_tl
1070 \tl_new:N \l_zrefclever_label_type_b_tl
1071 \tl_new:N \l_zrefclever_label_enclcnt_a_tl
1072 \tl_new:N \l_zrefclever_label_enclcnt_b_tl
1073 \tl_new:N \l_zrefclever_label_enclval_a_tl
1074 \tl_new:N \l_zrefclever_label_enclval_b_tl
1075 \tl_new:N \l_zrefclever_label_enclhead_a_tl
1076 \tl_new:N \l_zrefclever_label_enclhead_b_tl
```

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

```
1077 \int_new:N \l__zrefclever_sort_prior_a_int
1078 \int_new:N \l__zrefclever_sort_prior_b_int
```

`\l_zrefclever_sort_decided_bool` Auxiliary variable for `_zrefclever_sort_default:nn`, signals if the sorting between two labels has been decided or not.

```
1079 \bool_new:N \l__zrefclever_sort_decided_bool
```

(End definition for `\l__zrefclever_sort_decided_bool`.)

Variant not provided by the kernel.

```
1080 \cs_generate_variant:Nn \tl_reverse_items:n { V }
```

`_zrefclever_label_type_put_new_right:n` 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}
```

```

1081 \cs_new_protected:Npn \__zrefclever_label_type_put_new_right:n #1
1082 {
1083   \tl_set:Nx \l__zrefclever_label_type_a_tl
1084   { \zref@extractdefault {#1} {zc@type} { \c_empty_tl } }
1085   \tl_if_empty:NF \l__zrefclever_label_type_a_tl
1086   {
1087     \seq_if_in:NVF
1088     \l__zrefclever_label_types_seq
1089     \l__zrefclever_label_type_a_tl
1090     {
1091       \seq_put_right:NV \l__zrefclever_label_types_seq
1092       \l__zrefclever_label_type_a_tl
1093     }
1094   }
1095 }

```

(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`.

```

1096 \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.

```

1097 \cs_new_protected:Npn \__zrefclever_sort_labels:
1098 {

```

Store label types sequence.

```

1099   \seq_clear:N \l__zrefclever_label_types_seq
1100   \tl_if_eq:NnF \l__zrefclever_ref_property_tl { page }
1101   {
1102     \seq_map_function:NN \l__zrefclever_zcref_labels_seq
1103     \__zrefclever_label_type_put_new_right:n
1104   }

```

Sort.

```

1105   \seq_sort:Nn \l__zrefclever_zcref_labels_seq
1106   {
1107     \zref@ifrefundefined {##1}
1108     {
1109       \zref@ifrefundefined {##2}
1110       {
1111         % Neither label is defined.
1112         \sort_return_same:
1113       }
1114       {
1115         % The second label is defined, but the first isn't, leave the
1116         % undefined first (to be more visible).
1117         \sort_return_same:

```

```

1118     }
1119   }
1120   {
1121     \zref@ifrefundefined {##2}
1122     {
1123       % The first label is defined, but the second isn't, bring the
1124       % second forward.
1125       \sort_return_swapped:
1126     }
1127     {
1128       % The interesting case: both labels are defined. The
1129       % reference to the "default" property/counter or to the page
1130       % are quite different from our perspective, they rely on
1131       % different fields and even use different information for
1132       % sorting, so we branch them here to specialized functions.
1133       \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1134       { \__zrefclever_sort_page:nn {##1} {##2} }
1135       { \__zrefclever_sort_default:nn {##1} {##2} }
1136     }
1137   }
1138 }
1139 }

```

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

```

```

1162     }
1163     { \sort_return_swapped: }
1164     {
1165         \bool_if:nTF
1166         {
1167             % The interesting case: both labels have a type...
1168             ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1169             ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1170         }
1171         {
1172             % Here we send this to a couple of auxiliary functions for no
1173             % other reason than to keep this long function a little less
1174             % unreadable.
1175             \tl_if_eq:NNTF
1176             \l__zrefclever_label_type_a_tl
1177             \l__zrefclever_label_type_b_tl
1178             {
1179                 % ...and it's the same type.
1180                 \__zrefclever_sort_default_same_type:nn {#1} {#2}
1181             }
1182             {
1183                 % ...and they are different types.
1184                 \__zrefclever_sort_default_different_types:nn {#1} {#2}
1185             }
1186         }
1187         {
1188             % Neither of the labels has a type. We can't do much of
1189             % meaningful here, but if it's the same counter, compare it.
1190             \exp_args:Nxx \tl_if_eq:nnTF
1191             { \zref@extractdefault {#1} { counter } { } }
1192             { \zref@extractdefault {#2} { counter } { } }
1193             {
1194                 \int_compare:nNnTF
1195                 { \zref@extractdefault {#1} { zc@cntval } {-1} }
1196                 >
1197                 { \zref@extractdefault {#2} { zc@cntval } {-1} }
1198                 { \sort_return_swapped: }
1199                 { \sort_return_same: }
1200             }
1201             { \sort_return_same: }
1202         }
1203     }
1204 }
1205 }

```

(End definition for __zrefclever_sort_default:nn.)

__zrefclever_sort_default_same_type:nn

```

1206 \cs_new_protected:Npn \__zrefclever_sort_default_same_type:nn #1#2
1207 {
1208     \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1209     { \zref@extractdefault {#1} { zc@enclcnt } { \c_empty_tl } }
1210     \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1211     { \tl_reverse_items:V \l__zrefclever_label_enclcnt_a_tl }

```

```

1212 \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1213 { \zref@extractdefault {#2} { zc@enclcnt } { \c_empty_tl } }
1214 \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1215 { \tl_reverse_items:V \l__zrefclever_label_enclcnt_b_tl }
1216 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1217 { \zref@extractdefault {#1} { zc@enclval } { \c_empty_tl } }
1218 \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1219 { \tl_reverse_items:V \l__zrefclever_label_enclval_a_tl }
1220 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1221 { \zref@extractdefault {#2} { zc@enclval } { \c_empty_tl } }
1222 \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1223 { \tl_reverse_items:V \l__zrefclever_label_enclval_b_tl }
1224
1225 \bool_set_false:N \l__zrefclever_sort_decided_bool
1226 \bool_until_do:Nn \l__zrefclever_sort_decided_bool
1227 {
1228   \tl_set:Nx \l__zrefclever_label_enclhead_a_tl
1229   { \tl_head:N \l__zrefclever_label_enclcnt_a_tl }
1230   \tl_set:Nx \l__zrefclever_label_enclhead_b_tl
1231   { \tl_head:N \l__zrefclever_label_enclcnt_b_tl }
1232
1233   \bool_if:nTF
1234   {
1235     % Both are empty, meaning: neither labels have any (further)
1236     % "enclosing counters" (left).
1237     \tl_if_empty_p:V \l__zrefclever_label_enclhead_a_tl &&
1238     \tl_if_empty_p:V \l__zrefclever_label_enclhead_b_tl
1239   }
1240   {
1241     \exp_args:Nxx \tl_if_eq:nnTF
1242     { \zref@extractdefault {#1} { counter } { } }
1243     { \zref@extractdefault {#2} { counter } { } }
1244     {
1245       \bool_set_true:N \l__zrefclever_sort_decided_bool
1246       \int_compare:nNnTF
1247       { \zref@extractdefault {#1} { zc@cntval } {-1} }
1248       >
1249       { \zref@extractdefault {#2} { zc@cntval } {-1} }
1250       { \sort_return_swapped: }
1251       { \sort_return_same: }
1252     }
1253   }
1254   {
1255     \msg_warning:nnnn { zref-clever }
1256     { counters-not-nested } {#1} {#2}
1257     \bool_set_true:N \l__zrefclever_sort_decided_bool
1258     \sort_return_same:
1259   }
1260   {
1261     \bool_if:nTF
1262     {
1263       % 'a' is empty (and 'b' is not), meaning: 'b' is (possibly)
1264       % nested in 'a'.
1265       \tl_if_empty_p:V \l__zrefclever_label_enclhead_a_tl

```

```

1266 }
1267 {
1268   \tl_set:Nx \l_tmpa_tl
1269     { {\zref@extractdefault {#1} { counter } { }} }
1270   \exp_args:NNx \tl_if_in:NnTF
1271     \l__zrefclever_label_enclcnt_b_tl { \l_tmpa_tl }
1272     {
1273       \bool_set_true:N \l__zrefclever_sort_decided_bool
1274       \sort_return_same:
1275     }
1276   {
1277     \msg_warning:nnnn { zref-clever }
1278     { counters-not-nested } {#1} {#2}
1279     \bool_set_true:N \l__zrefclever_sort_decided_bool
1280     \sort_return_same:
1281   }
1282 }
1283 {
1284   \bool_if:nTF
1285     {
1286       % 'b' is empty (and 'a' is not), meaning: 'a' is
1287       % (possibly) nested in 'b'.
1288       \tl_if_empty_p:V \l__zrefclever_label_enclhead_b_tl
1289     }
1290     {
1291       \tl_set:Nx \l_tmpb_tl
1292         { {\zref@extractdefault {#2} { counter } { }} }
1293       \exp_args:NNx \tl_if_in:NnTF
1294         \l__zrefclever_label_enclcnt_a_tl { \l_tmpb_tl }
1295         {
1296           \bool_set_true:N \l__zrefclever_sort_decided_bool
1297           \sort_return_swapped:
1298         }
1299       {
1300         \msg_warning:nnnn { zref-clever }
1301         { counters-not-nested } {#1} {#2}
1302         \bool_set_true:N \l__zrefclever_sort_decided_bool
1303         \sort_return_same:
1304       }
1305     }
1306   {
1307     % Neither is empty, meaning: we can (possibly) compare the
1308     % values of the current enclosing counter in the loop, if
1309     % they are equal, we are still in the loop, if they are
1310     % not, a sorting decision can be made directly.
1311     \tl_if_eq:NNTF
1312       \l__zrefclever_label_enclhead_a_tl
1313       \l__zrefclever_label_enclhead_b_tl
1314     {
1315       \int_compare:nNnTF
1316         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1317         =
1318         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1319         {

```

```

1320         \tl_set:Nx \l__zrefclever_label_enclcnt_a_tl
1321         { \tl_tail:N \l__zrefclever_label_enclcnt_a_tl }
1322         \tl_set:Nx \l__zrefclever_label_enclcnt_b_tl
1323         { \tl_tail:N \l__zrefclever_label_enclcnt_b_tl }
1324         \tl_set:Nx \l__zrefclever_label_enclval_a_tl
1325         { \tl_tail:N \l__zrefclever_label_enclval_a_tl }
1326         \tl_set:Nx \l__zrefclever_label_enclval_b_tl
1327         { \tl_tail:N \l__zrefclever_label_enclval_b_tl }
1328     }
1329     {
1330         \bool_set_true:N \l__zrefclever_sort_decided_bool
1331         \int_compare:nNnTF
1332         { \tl_head:N \l__zrefclever_label_enclval_a_tl }
1333         >
1334         { \tl_head:N \l__zrefclever_label_enclval_b_tl }
1335         { \sort_return_swapped: }
1336         { \sort_return_same: }
1337     }
1338 }
1339 {
1340     \msg_warning:nnnn { zref-clever }
1341     { counters-not-nested } {#1} {#2}
1342     \bool_set_true:N \l__zrefclever_sort_decided_bool
1343     \sort_return_same:
1344 }
1345 }
1346 }
1347 }
1348 }
1349 }

```

(End definition for `__zrefclever_sort_default_same_type:nn`.)

`__zrefclever_sort_default_different_types:nn`

```

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

```



```

1370         \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##2}
1371         { \int_set:Nn \l__zrefclever_sort_prior_b_int { - ##1 } }
1372     }
1373 }
1374 }
1375 \bool_if:nTF
1376 {
1377     \int_compare_p:nNn
1378     { \l__zrefclever_sort_prior_a_int } <
1379     { \l__zrefclever_sort_prior_b_int }
1380 }
1381 { \sort_return_same: }
1382 {
1383     \bool_if:nTF
1384     {
1385         \int_compare_p:nNn
1386         { \l__zrefclever_sort_prior_a_int } >
1387         { \l__zrefclever_sort_prior_b_int }
1388     }
1389     { \sort_return_swapped: }
1390     {
1391         % Sort priorities are equal for different types: the type that
1392         % occurs first in 'labels', as given by the user, is kept (or
1393         % brought) forward.
1394         \seq_map_inline:Nn \l__zrefclever_label_types_seq
1395         {
1396             \tl_if_eq:NnTF \l__zrefclever_label_type_a_tl {##1}
1397             { \seq_map_break:n { \sort_return_same: } }
1398             {
1399                 \tl_if_eq:NnT \l__zrefclever_label_type_b_tl {##1}
1400                 { \seq_map_break:n { \sort_return_swapped: } }
1401             }
1402         }
1403     }
1404 }
1405 }

```

(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>}

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

```

1414 }

(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, it is easy to deal with this in case the need arises, by just adding another condition (coming from whatever the chosen syntax was) when we check for `_zrefclever_labels_in_sequence:nn` in `_zrefclever_typeset_refs_aux_not_last_of_type:.`

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.

1415 `\bool_new:N \l_zrefclever_typeset_last_bool`
1416 `\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.

1417 `\seq_new:N \l_zrefclever_typeset_labels_seq`
1418 `\tl_new:N \l_zrefclever_typeset_queue_prev_tl`
1419 `\tl_new:N \l_zrefclever_typeset_queue_curr_tl`
1420 `\tl_new:N \l_zrefclever_type_first_label_tl`
1421 `\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.

1422 `\int_new:N \l_zrefclever_label_count_int`
1423 `\int_new:N \l_zrefclever_type_count_int`

(End definition for `\l_zrefclever_label_count_int` and `\l_zrefclever_type_count_int`.)

`\l_zrefclever_range_count_int` Range related auxiliary variables for `__zrefclever_typeset_refs:`. `\l_zrefclever_range_count_int` counts how many references/labels are in the current ongoing range.
`\l_zrefclever_range_same_count_int` `\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.

```

1424 \int_new:N \l_zrefclever_range_count_int
1425 \int_new:N \l_zrefclever_range_same_count_int
1426 \tl_new:N \l_zrefclever_range_beg_label_tl
1427 \bool_new:N \l_zrefclever_next_maybe_range_bool
1428 \bool_new:N \l_zrefclever_next_is_same_bool

```

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

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

```

1429 \tl_new:N \l_zrefclever_namefont_tl
1430 \tl_new:N \l_zrefclever_reffont_out_tl
1431 \tl_new:N \l_zrefclever_reffont_in_tl
1432 \tl_new:N \l_zrefclever_namesep_tl
1433 \tl_new:N \l_zrefclever_rangesep_tl
1434 \tl_new:N \l_zrefclever_pairsep_tl
1435 \tl_new:N \l_zrefclever_listsep_tl
1436 \tl_new:N \l_zrefclever_lastsep_tl
1437 \tl_new:N \l_zrefclever_tpairsep_tl
1438 \tl_new:N \l_zrefclever_tlistsep_tl
1439 \tl_new:N \l_zrefclever_tlastsep_tl
1440 \tl_new:N \l_zrefclever_notesep_tl
1441 \tl_new:N \l_zrefclever_refpre_out_tl
1442 \tl_new:N \l_zrefclever_refpos_out_tl
1443 \tl_new:N \l_zrefclever_refpre_in_tl
1444 \tl_new:N \l_zrefclever_refpos_in_tl

```

(End definition for .)

`\l_zrefclever_type_name_tl` Auxiliary variables for `__zrefclever_get_ref_first:` and `__zrefclever_type_name_setup:`.
`\l_zrefclever_name_in_link_bool`

```

1445 \tl_new:N \l_zrefclever_type_name_tl
1446 \bool_new:N \l_zrefclever_name_in_link_bool
1447 \tl_new:N \l_zrefclever_name_format_tl
1448 \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`.

```

1449 \cs_new_protected:Npn \__zrefclever_typeset_refs:
1450 {
1451   \seq_set_eq:NN \l_zrefclever_typeset_labels_seq
1452   \l_zrefclever_zceref_labels_seq
1453   \tl_clear:N \l_zrefclever_typeset_queue_prev_tl

```

```

1454 \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1455 \tl_clear:N \l__zrefclever_type_first_label_tl
1456 \tl_clear:N \l__zrefclever_type_first_label_type_tl
1457 \tl_clear:N \l__zrefclever_range_beg_label_tl
1458 \int_zero:N \l__zrefclever_label_count_int
1459 \int_zero:N \l__zrefclever_type_count_int
1460 \int_zero:N \l__zrefclever_range_count_int
1461 \int_zero:N \l__zrefclever_range_same_count_int
1462
1463 % Get not-type-specific separators and refpre/pos options.
1464 \__zrefclever_get_ref_string:nN { tpairsep }
1465 \l__zrefclever_tpairsep_tl
1466 \__zrefclever_get_ref_string:nN { tlistsep }
1467 \l__zrefclever_tlistsep_tl
1468 \__zrefclever_get_ref_string:nN { tlastsep }
1469 \l__zrefclever_tlastsep_tl
1470
1471 % Loop over the label list in sequence.
1472 \bool_set_false:N \l__zrefclever_typeset_last_bool
1473 \bool_until_do:Nn \l__zrefclever_typeset_last_bool
1474 {
1475   \seq_pop_left:NN \l__zrefclever_typeset_labels_seq
1476   \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   {
1483     \seq_get_left:NN \l__zrefclever_typeset_labels_seq
1484     \l__zrefclever_label_b_tl
1485   }
1486
1487   \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
1488   {
1489     \tl_set:Nn \l__zrefclever_label_type_a_tl { page }
1490     \tl_set:Nn \l__zrefclever_label_type_b_tl { page }
1491   }
1492   {
1493     \tl_set:Nx \l__zrefclever_label_type_a_tl
1494     {
1495       \zref@extractdefault
1496       { \l__zrefclever_label_a_tl } { zc@type } { \c_empty_tl }
1497     }
1498     \tl_set:Nx \l__zrefclever_label_type_b_tl
1499     {
1500       \zref@extractdefault
1501       { \l__zrefclever_label_b_tl } { zc@type } { \c_empty_tl }
1502     }
1503   }
1504
1505   % First, we establish whether the "current label" (i.e. 'a') is the
1506   % last one of its type. This can happen because the "next label"
1507   % (i.e. 'b') is of a different type (or different definition status),

```

```

1508 % or because we are at the end of the list.
1509 \bool_if:NTF \l__zrefclever_typeset_last_bool
1510 { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1511 {
1512   \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1513   {
1514     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1515     { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1516     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1517   }
1518   {
1519     \zref@ifrefundefined { \l__zrefclever_label_b_tl }
1520     { \bool_set_true:N \l__zrefclever_last_of_type_bool }
1521     {
1522       % Neither is undefined, we must check the types.
1523       \bool_if:nTF
1524       % Both empty: same "type".
1525       {
1526         \tl_if_empty_p:N \l__zrefclever_label_type_a_tl &&
1527         \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1528       }
1529       { \bool_set_false:N \l__zrefclever_last_of_type_bool }
1530       {
1531         \bool_if:nTF
1532         % Neither empty: compare types.
1533         {
1534           ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl
1535           &&
1536           ! \tl_if_empty_p:N \l__zrefclever_label_type_b_tl
1537         }
1538         {
1539           \tl_if_eq:NNTF
1540           \l__zrefclever_label_type_a_tl
1541           \l__zrefclever_label_type_b_tl
1542           {
1543             \bool_set_false:N
1544             \l__zrefclever_last_of_type_bool
1545           }
1546           {
1547             \bool_set_true:N
1548             \l__zrefclever_last_of_type_bool
1549           }
1550         }
1551         % One empty, the other not: different "types".
1552         {
1553           \bool_set_true:N
1554           \l__zrefclever_last_of_type_bool
1555         }
1556       }
1557     }
1558   }
1559 }
1560
1561 % Handle warnings in case of reference or type undefined.

```

```

1562 \zref@refused { \l__zrefclever_label_a_tl }
1563 \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1564 {}
1565 {
1566   \tl_if_empty:NT \l__zrefclever_label_type_a_tl
1567   {
1568     \msg_warning:nxx { zref-clever } { missing-type }
1569     { \l__zrefclever_label_a_tl }
1570   }
1571 }
1572
1573 % Get type-specific separators, refpre/pos and font options, once per
1574 % type.
1575 \int_compare:nNnT { \l__zrefclever_label_count_int } = { 0 }
1576 {
1577   \__zrefclever_get_ref_font:nN { namefont }
1578   \l__zrefclever_namefont_tl
1579   \__zrefclever_get_ref_font:nN { reffont }
1580   \l__zrefclever_reffont_out_tl
1581   \__zrefclever_get_ref_font:nN { reffont-in }
1582   \l__zrefclever_reffont_in_tl
1583   \__zrefclever_get_ref_string:nN { namesep }
1584   \l__zrefclever_namesep_tl
1585   \__zrefclever_get_ref_string:nN { rangesep }
1586   \l__zrefclever_rangesep_tl
1587   \__zrefclever_get_ref_string:nN { pairsep }
1588   \l__zrefclever_pairsep_tl
1589   \__zrefclever_get_ref_string:nN { listsep }
1590   \l__zrefclever_listsep_tl
1591   \__zrefclever_get_ref_string:nN { lastsep }
1592   \l__zrefclever_lastsep_tl
1593   \__zrefclever_get_ref_string:nN { refpre }
1594   \l__zrefclever_refpre_out_tl
1595   \__zrefclever_get_ref_string:nN { refpos }
1596   \l__zrefclever_refpos_out_tl
1597   \__zrefclever_get_ref_string:nN { refpre-in }
1598   \l__zrefclever_refpre_in_tl
1599   \__zrefclever_get_ref_string:nN { refpos-in }
1600   \l__zrefclever_refpos_in_tl
1601 }
1602
1603 % Here we send this to a couple of auxiliary functions for no other
1604 % reason than to keep this long function a little less unreadable.
1605 \bool_if:NTF \l__zrefclever_last_of_type_bool
1606 {
1607   % There exists no next label of the same type as the current.
1608   \__zrefclever_typeset_refs_aux_last_of_type:
1609 }
1610 {
1611   % There exists a next label of the same type as the current.
1612   \__zrefclever_typeset_refs_aux_not_last_of_type:
1613 }
1614 }
1615 }

```

(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.
1616 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_last_of_type:
1617 {
1618   % Process the current label to the current queue.
1619   \int_case:nnF { \l__zrefclever_label_count_int }
1620   {
1621     % It is the last label of its type, but also the first one, and that's
1622     % what matters here: just store it.
1623     { 0 }
1624     {
1625       \tl_set:NV \l__zrefclever_type_first_label_tl
1626       \l__zrefclever_label_a_tl
1627       \tl_set:NV \l__zrefclever_type_first_label_type_tl
1628       \l__zrefclever_label_type_a_tl
1629     }
1630
1631     % The last is the second: we have a pair (if not repeated).
1632     { 1 }
1633     {
1634       \int_compare:nNnF { \l__zrefclever_range_same_count_int } = { 1 }
1635       {
1636         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1637         {
1638           \exp_not:V \l__zrefclever_pairsep_tl
1639           \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1640         }
1641       }
1642     }
1643   }
1644   % If neither the first, nor the second: we have the last label
1645   % on the current type list (if not repeated).
1646   {
1647     \int_case:nnF { \l__zrefclever_range_count_int }
1648     {
1649       % There was no range going on.
1650       { 0 }
1651       {
1652         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1653         {
1654           \exp_not:V \l__zrefclever_lastsep_tl
1655           \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1656         }
1657       }
1658       % Last in the range is also the second in it.
1659       { 1 }
1660       {
1661         \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1662         {
1663           % We know 'range_beg_label' is not empty, since this is the
1664           % second element in the range, but the third or more in the
1665           % type list.
1666           \exp_not:V \l__zrefclever_listsep_tl

```

```

1667         \_zrefclever_get_ref:V \l__zrefclever_range_beg_label_tl
1668         \int_compare:nNnF
1669             { \l__zrefclever_range_same_count_int } = { 1 }
1670         {
1671             \exp_not:V \l__zrefclever_lastsep_tl
1672             \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1673         }
1674     }
1675 }
1676 }
1677 % Last in the range is third or more in it.
1678 {
1679     \int_case:nnF
1680     {
1681         \l__zrefclever_range_count_int -
1682         \l__zrefclever_range_same_count_int
1683     }
1684     {
1685         % Repetition, not a range.
1686         { 0 }
1687         {
1688             % If 'range_beg_label' is empty, it means it was also the
1689             % first of the type, and hence was already handled.
1690             \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1691             {
1692                 \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1693                 {
1694                     \exp_not:V \l__zrefclever_lastsep_tl
1695                     \_zrefclever_get_ref:V
1696                     \l__zrefclever_range_beg_label_tl
1697                 }
1698             }
1699         }
1700         % A 'range', but with no skipped value, treat as list.
1701         { 1 }
1702         {
1703             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1704             {
1705                 % Ditto.
1706                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1707                 {
1708                     \exp_not:V \l__zrefclever_listsep_tl
1709                     \_zrefclever_get_ref:V
1710                     \l__zrefclever_range_beg_label_tl
1711                 }
1712                 \exp_not:V \l__zrefclever_lastsep_tl
1713                 \_zrefclever_get_ref:V \l__zrefclever_label_a_tl
1714             }
1715         }
1716     }
1717 {
1718     % An actual range.
1719     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1720     {

```



```

1721         % Ditto.
1722         \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1723         {
1724             \exp_not:V \l__zrefclever_lastsep_tl
1725             \__zrefclever_get_ref:V
1726             \l__zrefclever_range_beg_label_tl
1727         }
1728         \exp_not:V \l__zrefclever_rangesep_tl
1729         \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1730     }
1731 }
1732 }
1733 }
1734
1735 % Handle "range" option. The idea is simple: if the queue is not empty,
1736 % we replace it with the end of the range (or pair). We can still
1737 % retrieve the end of the range from 'label_a' since we know to be
1738 % processing the last label of its type at this point.
1739 \bool_if:NT \l__zrefclever_typeset_range_bool
1740 {
1741     \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
1742     {
1743         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1744         { }
1745         {
1746             \msg_warning:nxx { zref-clever } { single-element-range }
1747             { \l__zrefclever_type_first_label_type_tl }
1748         }
1749     }
1750     {
1751         \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1752         \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
1753         { }
1754         {
1755             \__zrefclever_labels_in_sequence:nn
1756             { \l__zrefclever_type_first_label_tl }
1757             { \l__zrefclever_label_a_tl }
1758         }
1759         \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1760         {
1761             \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1762             { \exp_not:V \l__zrefclever_pairsep_tl }
1763             { \exp_not:V \l__zrefclever_rangesep_tl }
1764             \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1765         }
1766     }
1767 }
1768
1769 % Now that the type is finished, we can add the name and the first ref to
1770 % the queue. Or, if "typeset" option is not "both", handle it here too.
1771 \__zrefclever_type_name_setup:
1772 \bool_if:nTF
1773 { \l__zrefclever_typeset_ref_bool && \l__zrefclever_typeset_name_bool }
1774 {

```

```

1775 \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1776 { \__zrefclever_get_ref_first: }
1777 }
1778 {
1779 \bool_if:nTF
1780 { \l__zrefclever_typeset_ref_bool }
1781 {
1782 \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1783 { \__zrefclever_get_ref:V \l__zrefclever_type_first_label_tl }
1784 }
1785 {
1786 \bool_if:nTF
1787 { \l__zrefclever_typeset_name_bool }
1788 {
1789 \tl_set:Nx \l__zrefclever_typeset_queue_curr_tl
1790 {
1791 \bool_if:NTF \l__zrefclever_name_in_link_bool
1792 {
1793 \exp_not:N \group_begin:
1794 \exp_not:V \l__zrefclever_namefont_tl
1795 % It's two '@s', but escaped for DocStrip.
1796 \exp_not:N \hyper@@link
1797 {
1798 \zref@ifrefcontainsprop
1799 { \l__zrefclever_type_first_label_tl }
1800 { urluse }
1801 {
1802 \zref@extractdefault
1803 { \l__zrefclever_type_first_label_tl }
1804 { urluse } {}
1805 }
1806 {
1807 \zref@extractdefault
1808 { \l__zrefclever_type_first_label_tl }
1809 { url } {}
1810 }
1811 }
1812 {
1813 \zref@extractdefault
1814 { \l__zrefclever_type_first_label_tl }
1815 { anchor } {}
1816 }
1817 { \exp_not:V \l__zrefclever_type_name_tl }
1818 \exp_not:N \group_end:
1819 }
1820 {
1821 \exp_not:N \group_begin:
1822 \exp_not:V \l__zrefclever_namefont_tl
1823 \exp_not:V \l__zrefclever_type_name_tl
1824 \exp_not:N \group_end:
1825 }
1826 }
1827 }
1828 {

```

```

1829         % This case would correspond to "typeset=none" but should not
1830         % happen, given the options are set up to typeset at least one
1831         % of "ref" or "name", but a sensible fallback, equal to the
1832         % behavior of "both".
1833         \tl_put_left:Nx \l__zrefclever_typeset_queue_curr_tl
1834         { \l__zrefclever_get_ref_first: }
1835     }
1836 }
1837 }
1838
1839 % Typeset the previous type, if there is one.
1840 \int_compare:nNnT { \l__zrefclever_type_count_int } > { 0 }
1841 {
1842     \int_compare:nNnT { \l__zrefclever_type_count_int } > { 1 }
1843     { \l__zrefclever_tlistsep_tl }
1844     \l__zrefclever_typeset_queue_prev_tl
1845 }
1846
1847 % Wrap up loop, or prepare for next iteration.
1848 \bool_if:NTF \l__zrefclever_typeset_last_bool
1849 {
1850     % We are finishing, typeset the current queue.
1851     \int_case:nnF { \l__zrefclever_type_count_int }
1852     {
1853         % Single type.
1854         { 0 }
1855         { \l__zrefclever_typeset_queue_curr_tl }
1856         % Pair of types.
1857         { 1 }
1858         {
1859             \l__zrefclever_tpairsep_tl
1860             \l__zrefclever_typeset_queue_curr_tl
1861         }
1862     }
1863     {
1864         % Last in list of types.
1865         \l__zrefclever_tlastsep_tl
1866         \l__zrefclever_typeset_queue_curr_tl
1867     }
1868 }
1869 {
1870     % There are further labels, set variables for next iteration.
1871     \tl_set_eq:NN \l__zrefclever_typeset_queue_prev_tl
1872     \l__zrefclever_typeset_queue_curr_tl
1873     \tl_clear:N \l__zrefclever_typeset_queue_curr_tl
1874     \tl_clear:N \l__zrefclever_type_first_label_tl
1875     \tl_clear:N \l__zrefclever_type_first_label_type_tl
1876     \tl_clear:N \l__zrefclever_range_beg_label_tl
1877     \int_zero:N \l__zrefclever_label_count_int
1878     \int_incr:N \l__zrefclever_type_count_int
1879     \int_zero:N \l__zrefclever_range_count_int
1880     \int_zero:N \l__zrefclever_range_same_count_int
1881 }
1882 }

```

(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.

```

1883 \cs_new_protected:Npn \_zrefclever_typeset_refs_aux_not_last_of_type:
1884 {
1885   % Signal if next label may form a range with the current one (of course,
1886   % only considered if compression is enabled in the first place).
1887   \bool_set_false:N \l__zrefclever_next_maybe_range_bool
1888   \bool_set_false:N \l__zrefclever_next_is_same_bool
1889   \bool_if:NT \l__zrefclever_typeset_compress_bool
1890   {
1891     \zref@ifrefundefined { \l__zrefclever_label_a_tl }
1892     { }
1893     {
1894       \__zrefclever_labels_in_sequence:nn
1895       { \l__zrefclever_label_a_tl } { \l__zrefclever_label_b_tl }
1896     }
1897   }
1898
1899   % Process the current label to the current queue.
1900   \int_compare:nNnTF { \l__zrefclever_label_count_int } = { 0 }
1901   {
1902     % Current label is the first of its type (also not the last, but it
1903     % doesn't matter here): just store the label.
1904     \tl_set:NV \l__zrefclever_type_first_label_tl
1905     \l__zrefclever_label_a_tl
1906     \tl_set:NV \l__zrefclever_type_first_label_type_tl
1907     \l__zrefclever_label_type_a_tl
1908
1909     % If the next label may be part of a range, we set 'range_beg_label'
1910     % to "empty" (we deal with it as the "first", and must do it there, to
1911     % handle hyperlinking), but also step the range counters.
1912     \bool_if:NT \l__zrefclever_next_maybe_range_bool
1913     {
1914       \tl_clear:N \l__zrefclever_range_beg_label_tl
1915       \int_incr:N \l__zrefclever_range_count_int
1916       \bool_if:NT \l__zrefclever_next_is_same_bool
1917       { \int_incr:N \l__zrefclever_range_same_count_int }
1918     }
1919   }
1920   {
1921     % Current label is neither the first (nor the last) of its type.
1922     \bool_if:NTF \l__zrefclever_next_maybe_range_bool
1923     {
1924       % Starting, or continuing a range.
1925       \int_compare:nNnTF
1926       { \l__zrefclever_range_count_int } = { 0 }
1927       {
1928         % There was no range going, we are starting one.
1929         \tl_set:NV \l__zrefclever_range_beg_label_tl
1930         \l__zrefclever_label_a_tl
1931         \int_incr:N \l__zrefclever_range_count_int
1932         \bool_if:NT \l__zrefclever_next_is_same_bool
1933         { \int_incr:N \l__zrefclever_range_same_count_int }

```

```

1934     }
1935     {
1936         % Second or more in the range, but not the last.
1937         \int_incr:N \l__zrefclever_range_count_int
1938         \bool_if:NT \l__zrefclever_next_is_same_bool
1939         { \int_incr:N \l__zrefclever_range_same_count_int }
1940     }
1941 }
1942 {
1943     % Next element is not in sequence, meaning: there was no range, or
1944     % we are closing one.
1945     \int_case:nnF { \l__zrefclever_range_count_int }
1946     {
1947         % There was no range going on.
1948         { 0 }
1949         {
1950             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1951             {
1952                 \exp_not:V \l__zrefclever_listsep_tl
1953                 \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
1954             }
1955         }
1956         % Last is second in the range: if 'range_same_count' is also
1957         % '1', it's a repetition (drop it), otherwise, it's a "pair
1958         % within a list", treat as list.
1959         { 1 }
1960         {
1961             \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1962             {
1963                 \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1964                 {
1965                     \exp_not:V \l__zrefclever_listsep_tl
1966                     \__zrefclever_get_ref:V
1967                     \l__zrefclever_range_beg_label_tl
1968                 }
1969                 \int_compare:nNnF
1970                 { \l__zrefclever_range_same_count_int } = { 1 }
1971                 {
1972                     \exp_not:V \l__zrefclever_listsep_tl
1973                     \__zrefclever_get_ref:V
1974                     \l__zrefclever_label_a_tl
1975                 }
1976             }
1977         }
1978     }
1979 {
1980     % Last is third or more in the range: if 'range_count' and
1981     % 'range_same_count' are the same, its a repetition (drop it),
1982     % if they differ by '1', its a list, if they differ by more,
1983     % it is a real range.
1984     \int_case:nnF
1985     {
1986         \l__zrefclever_range_count_int -
1987         \l__zrefclever_range_same_count_int

```

```

1988     }
1989     {
1990     { 0 }
1991     {
1992     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
1993     {
1994     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
1995     {
1996     \exp_not:V \l__zrefclever_listsep_tl
1997     \__zrefclever_get_ref:V
1998     \l__zrefclever_range_beg_label_tl
1999     }
2000     }
2001     }
2002     { 1 }
2003     {
2004     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2005     {
2006     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2007     {
2008     \exp_not:V \l__zrefclever_listsep_tl
2009     \__zrefclever_get_ref:V
2010     \l__zrefclever_range_beg_label_tl
2011     }
2012     \exp_not:V \l__zrefclever_listsep_tl
2013     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2014     }
2015     }
2016     }
2017     {
2018     \tl_put_right:Nx \l__zrefclever_typeset_queue_curr_tl
2019     {
2020     \tl_if_empty:VF \l__zrefclever_range_beg_label_tl
2021     {
2022     \exp_not:V \l__zrefclever_listsep_tl
2023     \__zrefclever_get_ref:V
2024     \l__zrefclever_range_beg_label_tl
2025     }
2026     \exp_not:V \l__zrefclever_rangesep_tl
2027     \__zrefclever_get_ref:V \l__zrefclever_label_a_tl
2028     }
2029     }
2030     }
2031     % Reset counters.
2032     \int_zero:N \l__zrefclever_range_count_int
2033     \int_zero:N \l__zrefclever_range_same_count_int
2034     }
2035     }
2036     % Step label counter for next iteration.
2037     \int_incr:N \l__zrefclever_label_count_int
2038     }

```

(End definition for __zrefclever_typeset_refs_aux_not_last_of_type:.)

Aux functions

```

2039 \cs_new_protected:Npn \__zrefclever_ref_default:
2040 { \zref@default }
2041 \cs_new_protected:Npn \__zrefclever_name_default:
2042 { \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.

```

2043 \cs_new:Npn \__zrefclever_get_ref:n #1
2044 {
2045   \zref@ifrefcontainsprop {#1} { \l__zrefclever_ref_property_tl }
2046   {
2047     \bool_if:nTF
2048     {
2049       \l__zrefclever_use_hyperref_bool &&
2050       ! \l__zrefclever_link_star_bool
2051     }
2052     {
2053       \exp_not:N \group_begin:
2054       \exp_not:V \l__zrefclever_reffont_out_tl
2055       \exp_not:V \l__zrefclever_refpre_out_tl
2056       \exp_not:N \group_begin:
2057       \exp_not:V \l__zrefclever_reffont_in_tl
2058       % It's two '@s', but escaped for DocStrip.
2059       \exp_not:N \hyper@@link
2060       {
2061         \zref@ifrefcontainsprop {#1} { urluse }
2062         { \zref@extractdefault {#1} { urluse } { } }
2063         { \zref@extractdefault {#1} { url } { } }
2064       }
2065       { \zref@extractdefault {#1} { anchor } { } }
2066       {
2067         \exp_not:V \l__zrefclever_refpre_in_tl
2068         \zref@extractdefault {#1}
2069         { \l__zrefclever_ref_property_tl } { }
2070         \exp_not:V \l__zrefclever_refpos_in_tl
2071       }
2072       \exp_not:N \group_end:
2073       \exp_not:V \l__zrefclever_refpos_out_tl
2074       \exp_not:N \group_end:
2075     }
2076     {
2077       \exp_not:N \group_begin:
2078       \exp_not:V \l__zrefclever_reffont_out_tl
2079       \exp_not:V \l__zrefclever_refpre_out_tl
2080       \exp_not:N \group_begin:
2081       \exp_not:V \l__zrefclever_reffont_in_tl
2082       \exp_not:V \l__zrefclever_refpre_in_tl

```

```

2083         \zref@extractdefault {#1} { \l__zrefclever_ref_property_tl } { }
2084         \exp_not:V \l__zrefclever_refpos_in_tl
2085         \exp_not:N \group_end:
2086         \exp_not:V \l__zrefclever_refpos_out_tl
2087         \exp_not:N \group_end:
2088     }
2089 }
2090 { \exp_not:N \__zrefclever_ref_default: }
2091 }
2092 \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.

```

2093 \cs_new_protected:Npn \__zrefclever_type_name_setup:
2094 {
2095     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2096     { \tl_clear:N \l__zrefclever_type_name_tl }
2097     {
2098         \tl_if_empty:NTF \l__zrefclever_type_first_label_type_tl
2099         { \tl_clear:N \l__zrefclever_type_name_tl }
2100         {

```

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

```

2101         \bool_lazy_or:nnTF
2102         { \l__zrefclever_capitalize_bool }
2103         {
2104             \l__zrefclever_capitalize_first_bool &&
2105             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2106         }
2107         { \tl_set:Nn \l__zrefclever_name_format_tl {Name} }
2108         { \tl_set:Nn \l__zrefclever_name_format_tl {name} }
2109         % If the queue is empty, we have a singular, otherwise, plural.
2110         \tl_if_empty:NTF \l__zrefclever_typeset_queue_curr_tl
2111         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -sg } }
2112         { \tl_put_right:Nn \l__zrefclever_name_format_tl { -pl } }
2113         \bool_lazy_and:nnTF
2114         { \l__zrefclever_abbrev_bool }
2115         {
2116             ! \int_compare_p:nNn
2117             { \l__zrefclever_type_count_int } = { 0 } ||
2118             ! \l__zrefclever_noabbrev_first_bool
2119         }
2120         {
2121             \tl_set:NV \l__zrefclever_name_format_fallback_tl
2122             \l__zrefclever_name_format_tl
2123             \tl_put_right:Nn \l__zrefclever_name_format_tl { -ab }
2124         }
2125         { \tl_clear:N \l__zrefclever_name_format_fallback_tl }
2126
2127         \tl_if_empty:NTF \l__zrefclever_name_format_fallback_tl
2128         {
2129             \prop_get:cVNF
2130             {

```



```

2131         l__zrefclever_type_
2132         \l__zrefclever_type_first_label_type_tl _options_prop
2133     }
2134     \l__zrefclever_name_format_tl
2135     \l__zrefclever_type_name_tl
2136     {
2137         \__zrefclever_get_type_transl:xxxNF
2138         { \l__zrefclever_ref_language_tl }
2139         { \l__zrefclever_type_first_label_type_tl }
2140         { \l__zrefclever_name_format_tl }
2141         \l__zrefclever_type_name_tl
2142         {
2143             \tl_clear:N \l__zrefclever_type_name_tl
2144             \msg_warning:nxx { zref-clever } { missing-name }
2145             { \l__zrefclever_type_first_label_type_tl }
2146         }
2147     }
2148 }
2149 {
2150     \prop_get:cVNF
2151     {
2152         l__zrefclever_type_
2153         \l__zrefclever_type_first_label_type_tl _options_prop
2154     }
2155     \l__zrefclever_name_format_tl
2156     \l__zrefclever_type_name_tl
2157     {
2158         \prop_get:cVNF
2159         {
2160             l__zrefclever_type_
2161             \l__zrefclever_type_first_label_type_tl _options_prop
2162         }
2163         \l__zrefclever_name_format_fallback_tl
2164         \l__zrefclever_type_name_tl
2165         {
2166             \__zrefclever_get_type_transl:xxxNF
2167             { \l__zrefclever_ref_language_tl }
2168             { \l__zrefclever_type_first_label_type_tl }
2169             { \l__zrefclever_name_format_tl }
2170             \l__zrefclever_type_name_tl
2171             {
2172                 \__zrefclever_get_type_transl:xxxNF
2173                 { \l__zrefclever_ref_language_tl }
2174                 { \l__zrefclever_type_first_label_type_tl }
2175                 { \l__zrefclever_name_format_fallback_tl }
2176                 \l__zrefclever_type_name_tl
2177                 {
2178                     \tl_clear:N \l__zrefclever_type_name_tl
2179                     \msg_warning:nxx { zref-clever }
2180                     { missing-name }
2181                     { \l__zrefclever_type_first_label_type_tl }
2182                 }
2183             }
2184         }
2185     }

```

```

2185         }
2186     }
2187 }
2188 }

```

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

```

2189 \bool_lazy_any:nTF
2190 {
2191     { ! \l__zrefclever_use_hyperref_bool }
2192     { \l__zrefclever_link_star_bool }
2193     { \tl_if_empty_p:N \l__zrefclever_type_name_tl }
2194     { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { false } }
2195 }
2196 { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2197 {
2198     \bool_lazy_any:nTF
2199     {
2200         { \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { true } }
2201         {
2202             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { tsingle } &&
2203             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl
2204         }
2205         {
2206             \str_if_eq_p:Vn \l__zrefclever_nameinlink_str { single } &&
2207             \tl_if_empty_p:N \l__zrefclever_typeset_queue_curr_tl &&
2208             \l__zrefclever_typeset_last_bool &&
2209             \int_compare_p:nNn { \l__zrefclever_type_count_int } = { 0 }
2210         }
2211     }
2212     { \bool_set_true:N \l__zrefclever_name_in_link_bool }
2213     { \bool_set_false:N \l__zrefclever_name_in_link_bool }
2214 }
2215 }

```

(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.

```

2216 \cs_new:Npn \__zrefclever_get_ref_first:
2217 {
2218     \zref@ifrefundefined { \l__zrefclever_type_first_label_tl }
2219     { \exp_not:N \__zrefclever_ref_default: }
2220     {
2221         \bool_if:NTF \l__zrefclever_name_in_link_bool
2222         {
2223             \zref@ifrefcontainsprop
2224             { \l__zrefclever_type_first_label_tl }
2225             { \l__zrefclever_ref_property_tl }
2226             {
2227                 % It's two '@s', but escaped for DocStrip.
2228                 \exp_not:N \hyper@@link
2229                 {
2230                     \zref@ifrefcontainsprop
2231                     { \l__zrefclever_type_first_label_tl } { urluse }

```

```

2232         {
2233             \zref@extractdefault
2234             { \l__zrefclever_type_first_label_tl }
2235             { urluse } { }
2236         }
2237         {
2238             \zref@extractdefault
2239             { \l__zrefclever_type_first_label_tl }
2240             { url } { }
2241         }
2242     }
2243     {
2244         \zref@extractdefault
2245         { \l__zrefclever_type_first_label_tl }
2246         { anchor } { }
2247     }
2248     {
2249         \exp_not:N \group_begin:
2250         \exp_not:V \l__zrefclever_namefont_tl
2251         \exp_not:V \l__zrefclever_type_name_tl
2252         \exp_not:N \group_end:
2253         \exp_not:V \l__zrefclever_namesep_tl
2254         \exp_not:N \group_begin:
2255         \exp_not:V \l__zrefclever_reffont_out_tl
2256         \exp_not:V \l__zrefclever_refpre_out_tl
2257         \exp_not:N \group_begin:
2258         \exp_not:V \l__zrefclever_reffont_in_tl
2259         \exp_not:V \l__zrefclever_refpre_in_tl
2260         \zref@extractdefault
2261         { \l__zrefclever_type_first_label_tl }
2262         { \l__zrefclever_ref_property_tl } { }
2263         \exp_not:V \l__zrefclever_refpos_in_tl
2264         \exp_not:N \group_end:
2265         % hyperlink makes it's own group, we'd like to close the
2266         % 'refpre-out' group after 'refpos-out', but... we close
2267         % it here, and give the trailing 'refpos-out' its own
2268         % group. This will result that formatting given to
2269         % 'refpre-out' will not reach 'refpos-out', but I see no
2270         % alternative, and this has to be handled specially.
2271         \exp_not:N \group_end:
2272     }
2273     \exp_not:N \group_begin:
2274     % Ditto: special treatment.
2275     \exp_not:V \l__zrefclever_reffont_out_tl
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_namefont_tl
2282     \exp_not:V \l__zrefclever_type_name_tl
2283     \exp_not:N \group_end:
2284     \exp_not:V \l__zrefclever_namesep_tl
2285     \exp_not:N \l__zrefclever_ref_default:

```

```

2286     }
2287 }
2288 {
2289   \tl_if_empty:NTF \l__zrefclever_type_name_tl
2290   {
2291     \exp_not:N \__zrefclever_name_default:
2292     \exp_not:V \l__zrefclever_namesep_tl
2293   }
2294   {
2295     \exp_not:N \group_begin:
2296     \exp_not:V \l__zrefclever_namefont_tl
2297     \exp_not:V \l__zrefclever_type_name_tl
2298     \exp_not:N \group_end:
2299     \exp_not:V \l__zrefclever_namesep_tl
2300   }
2301   \zref@ifrefcontainsprop
2302   { \l__zrefclever_type_first_label_tl }
2303   { \l__zrefclever_ref_property_tl }
2304   {
2305     \bool_if:nTF
2306     {
2307       \l__zrefclever_use_hyperref_bool &&
2308       ! \l__zrefclever_link_star_bool
2309     }
2310     {
2311       \exp_not:N \group_begin:
2312       \exp_not:V \l__zrefclever_reffont_out_tl
2313       \exp_not:V \l__zrefclever_refpre_out_tl
2314       \exp_not:N \group_begin:
2315       \exp_not:V \l__zrefclever_reffont_in_tl
2316       % It's two '@s', but escaped for DocStrip.
2317       \exp_not:N \hyper@@link
2318       {
2319         \zref@ifrefcontainsprop
2320         { \l__zrefclever_type_first_label_tl } { urluse }
2321         {
2322           \zref@extractdefault
2323           { \l__zrefclever_type_first_label_tl }
2324           { urluse } { }
2325         }
2326         {
2327           \zref@extractdefault
2328           { \l__zrefclever_type_first_label_tl }
2329           { url } { }
2330         }
2331       }
2332     }
2333     {
2334       \zref@extractdefault
2335       { \l__zrefclever_type_first_label_tl }
2336       { anchor } { }
2337     }
2338     {
2339       \exp_not:V \l__zrefclever_refpre_in_tl
2340       \zref@extractdefault

```

```

2340         { \l__zrefclever_type_first_label_tl }
2341         { \l__zrefclever_ref_property_tl } { }
2342         \exp_not:V \l__zrefclever_refpos_in_tl
2343     }
2344     \exp_not:N \group_end:
2345     \exp_not:V \l__zrefclever_refpos_out_tl
2346     \exp_not:N \group_end:
2347 }
2348 {
2349     \exp_not:N \group_begin:
2350     \exp_not:V \l__zrefclever_reffont_out_tl
2351     \exp_not:V \l__zrefclever_refpre_out_tl
2352     \exp_not:N \group_begin:
2353     \exp_not:V \l__zrefclever_reffont_in_tl
2354     \exp_not:V \l__zrefclever_refpre_in_tl
2355     \zref@extractdefault
2356     { \l__zrefclever_type_first_label_tl }
2357     { \l__zrefclever_ref_property_tl } { }
2358     \exp_not:V \l__zrefclever_refpos_in_tl
2359     \exp_not:N \group_end:
2360     \exp_not:V \l__zrefclever_refpos_out_tl
2361     \exp_not:N \group_end:
2362 }
2363 }
2364 { \exp_not:N \__zrefclever_ref_default: }
2365 }
2366 }
2367 }

```

(End definition for __zrefclever_get_ref_first:.)

__zrefclever_get_ref_string:nN

```

2368 % \Arg{option} \Arg{var to store result}
2369 \cs_new_protected:Npn \__zrefclever_get_ref_string:nN #1#2
2370 {

```

First attempt options stored in \l__zrefclever_ref_options_prop.

```

2371     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2372     {

```

If not found, try the type specific options.

```

2373     \bool_lazy_all:nTF
2374     {
2375         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2376         {
2377             \prop_if_exist_p:c
2378             {
2379                 \l__zrefclever_type_
2380                 \l__zrefclever_label_type_a_tl _options_prop
2381             }
2382         }
2383         {
2384             \prop_if_in_p:cn
2385             {
2386                 \l__zrefclever_type_

```

```

2387         \l__zrefclever_label_type_a_tl _options_prop
2388     }
2389     {#1}
2390 }
2391 }
2392 {
2393     \prop_get:cnN
2394     {
2395         l__zrefclever_type_
2396         \l__zrefclever_label_type_a_tl _options_prop
2397     }
2398     {#1} #2
2399 }
2400 {

```

If not found, try the type specific translations.

```

2401     \__zrefclever_get_type_transl:xnNF
2402     { \l__zrefclever_ref_language_tl }
2403     { \l__zrefclever_label_type_a_tl }
2404     {#1} #2
2405     {

```

If not found, try default translations.

```

2406         \__zrefclever_get_default_transl:xnNF
2407         { \l__zrefclever_ref_language_tl }
2408         {#1} #2
2409         {

```

If not found, try fallback.

```

2410         \__zrefclever_get_fallback_transl:nNF {#1} #2
2411         {
2412             \tl_clear:N #2
2413             \msg_warning:nnn { zref-clever }
2414             { missing-string } {#1}
2415         }
2416     }
2417 }
2418 }
2419 }
2420 }

```

(End definition for __zrefclever_get_ref_string:nN.)

__zrefclever_get_ref_font:nN

```

2421 \cs_new_protected:Npn \__zrefclever_get_ref_font:nN #1#2
2422 {

```

First attempt options stored in \l__zrefclever_ref_options_prop.

```

2423     \prop_get:NnNF \l__zrefclever_ref_options_prop {#1} #2
2424     {

```

If not found, try the type specific options.

```

2425         \bool_lazy_and:nnTF
2426         { ! \tl_if_empty_p:N \l__zrefclever_label_type_a_tl }
2427         {
2428             \prop_if_exist_p:c

```

```

2429         {
2430             l__zrefclever_type_
2431             \l__zrefclever_label_type_a_tl _options_prop
2432         }
2433     }
2434     {
2435         \prop_get:cnNF
2436         {
2437             l__zrefclever_type_
2438             \l__zrefclever_label_type_a_tl _options_prop
2439         }
2440         {#1} #2
2441         { \tl_clear:N #2 }
2442     }
2443     { \tl_clear:N #2 }
2444 }
2445 }

```

(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”.

```

2446 \cs_new_protected:Npn \__zrefclever_labels_in_sequence:nn #1#2
2447 {
2448     \tl_if_eq:NnTF \l__zrefclever_ref_property_tl { page }
2449     {
2450         \exp_args:Nxx \tl_if_eq:nnT
2451         { \zref@extractdefault {#1} { zc@pgfmt } { } }
2452         { \zref@extractdefault {#2} { zc@pgfmt } { } }
2453         {
2454             \int_compare:nNnTF
2455             { \zref@extractdefault {#1} { zc@pgval } { -2 } + 1 }
2456             =
2457             { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2458             { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2459             {
2460                 \int_compare:nNnT
2461                 { \zref@extractdefault {#1} { zc@pgval } { -1 } }
2462                 =
2463                 { \zref@extractdefault {#2} { zc@pgval } { -1 } }
2464                 {
2465                     \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2466                     \bool_set_true:N \l__zrefclever_next_is_same_bool
2467                 }
2468             }
2469         }
2470     }
2471     {
2472         \exp_args:Nxx \tl_if_eq:nnT
2473         { \zref@extractdefault {#1} { counter } { } }
2474         { \zref@extractdefault {#2} { counter } { } }
2475         {
2476             \exp_args:Nxx \tl_if_eq:nnT

```

```

2477 { \zref@extractdefault {#1} { zc@enclval } { } }
2478 { \zref@extractdefault {#2} { zc@enclval } { } }
2479 {
2480   \int_compare:nNnTF
2481     { \zref@extractdefault {#1} { zc@cntval } { -2 } + 1 }
2482     =
2483     { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2484     { \bool_set_true:N \l__zrefclever_next_maybe_range_bool }
2485     {
2486       \int_compare:nNnTF
2487         { \zref@extractdefault {#1} { zc@cntval } { -1 } }
2488         =
2489         { \zref@extractdefault {#2} { zc@cntval } { -1 } }
2490         {
2491           \bool_set_true:N \l__zrefclever_next_maybe_range_bool
2492           \bool_set_true:N \l__zrefclever_next_is_same_bool
2493         }
2494       }
2495     }
2496   }
2497 }
2498 }

```

(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.

```
2499 </package>
```

10 Dictionaries

10.1 English

```
2500 <package>\zcDeclareLanguage { english }
2501 <package>\zcDeclareLanguageAlias { american } { english }
2502 <package>\zcDeclareLanguageAlias { australian } { english }
2503 <package>\zcDeclareLanguageAlias { british } { english }
2504 <package>\zcDeclareLanguageAlias { canadian } { english }
2505 <package>\zcDeclareLanguageAlias { newzealand } { english }
2506 <package>\zcDeclareLanguageAlias { UKenglish } { english }
2507 <package>\zcDeclareLanguageAlias { USenglish } { english }

2508 <*dict-english>

2509 namesep = {\nobreakspace} ,
2510 pairsep = {\~and\nobreakspace} ,
2511 listsep = {\~,~} ,
2512 lastsep = {\~and\nobreakspace} ,
2513 tpairsep = {\~and\nobreakspace} ,
2514 tlistsep = {\~,~} ,
2515 tlastsep = {\~,~and\nobreakspace} ,
2516 notesep = {\~} ,
2517 rangesep = {\~to\nobreakspace} ,

2518
2519 type = part ,
2520   Name-sg = Part ,
2521   name-sg = part ,
2522   Name-pl = Parts ,
2523   name-pl = parts ,
2524
2525 type = chapter ,
2526   Name-sg = Chapter ,
2527   name-sg = chapter ,
2528   Name-pl = Chapters ,
2529   name-pl = chapters ,
2530
2531 type = section ,
2532   Name-sg = Section ,
2533   name-sg = section ,
2534   Name-pl = Sections ,
2535   name-pl = sections ,
2536
2537 type = paragraph ,
2538   Name-sg = Paragraph ,
2539   name-sg = paragraph ,
```

```

2540 Name-pl = Paragraphs ,
2541 name-pl = paragraphs ,
2542 Name-sg-ab = Par. ,
2543 name-sg-ab = par. ,
2544 Name-pl-ab = Par. ,
2545 name-pl-ab = par. ,
2546
2547 type = appendix ,
2548 Name-sg = Appendix ,
2549 name-sg = appendix ,
2550 Name-pl = Appendices ,
2551 name-pl = appendices ,
2552
2553 type = page ,
2554 Name-sg = Page ,
2555 name-sg = page ,
2556 Name-pl = Pages ,
2557 name-pl = pages ,
2558 name-sg-ab = p. ,
2559 name-pl-ab = pp. ,
2560
2561 type = line ,
2562 Name-sg = Line ,
2563 name-sg = line ,
2564 Name-pl = Lines ,
2565 name-pl = lines ,
2566
2567 type = figure ,
2568 Name-sg = Figure ,
2569 name-sg = figure ,
2570 Name-pl = Figures ,
2571 name-pl = figures ,
2572 Name-sg-ab = Fig. ,
2573 name-sg-ab = fig. ,
2574 Name-pl-ab = Figs. ,
2575 name-pl-ab = figs. ,
2576
2577 type = table ,
2578 Name-sg = Table ,
2579 name-sg = table ,
2580 Name-pl = Tables ,
2581 name-pl = tables ,
2582
2583 type = item ,
2584 Name-sg = Item ,
2585 name-sg = item ,
2586 Name-pl = Items ,
2587 name-pl = items ,
2588
2589 type = footnote ,
2590 Name-sg = Footnote ,
2591 name-sg = footnote ,
2592 Name-pl = Footnotes ,
2593 name-pl = footnotes ,

```

```

2594
2595 type = note ,
2596     Name-sg = Note ,
2597     name-sg = note ,
2598     Name-pl = Notes ,
2599     name-pl = notes ,
2600
2601 type = equation ,
2602     Name-sg = Equation ,
2603     name-sg = equation ,
2604     Name-pl = Equations ,
2605     name-pl = equations ,
2606     Name-sg-ab = Eq. ,
2607     name-sg-ab = eq. ,
2608     Name-pl-ab = Eqs. ,
2609     name-pl-ab = eqs. ,
2610     refpre-in = {()} ,
2611     refpos-in = {} ,
2612
2613 type = theorem ,
2614     Name-sg = Theorem ,
2615     name-sg = theorem ,
2616     Name-pl = Theorems ,
2617     name-pl = theorems ,
2618
2619 type = lemma ,
2620     Name-sg = Lemma ,
2621     name-sg = lemma ,
2622     Name-pl = Lemmas ,
2623     name-pl = lemmas ,
2624
2625 type = corollary ,
2626     Name-sg = Corollary ,
2627     name-sg = corollary ,
2628     Name-pl = Corollaries ,
2629     name-pl = corollaries ,
2630
2631 type = proposition ,
2632     Name-sg = Proposition ,
2633     name-sg = proposition ,
2634     Name-pl = Propositions ,
2635     name-pl = propositions ,
2636
2637 type = definition ,
2638     Name-sg = Definition ,
2639     name-sg = definition ,
2640     Name-pl = Definitions ,
2641     name-pl = definitions ,
2642
2643 type = proof ,
2644     Name-sg = Proof ,
2645     name-sg = proof ,
2646     Name-pl = Proofs ,
2647     name-pl = proofs ,

```

```

2648
2649 type = result ,
2650     Name-sg = Result ,
2651     name-sg = result ,
2652     Name-pl = Results ,
2653     name-pl = results ,
2654
2655 type = example ,
2656     Name-sg = Example ,
2657     name-sg = example ,
2658     Name-pl = Examples ,
2659     name-pl = examples ,
2660
2661 type = remark ,
2662     Name-sg = Remark ,
2663     name-sg = remark ,
2664     Name-pl = Remarks ,
2665     name-pl = remarks ,
2666
2667 type = algorithm ,
2668     Name-sg = Algorithm ,
2669     name-sg = algorithm ,
2670     Name-pl = Algorithms ,
2671     name-pl = algorithms ,
2672
2673 type = listing ,
2674     Name-sg = Listing ,
2675     name-sg = listing ,
2676     Name-pl = Listings ,
2677     name-pl = listings ,
2678
2679 type = exercise ,
2680     Name-sg = Exercise ,
2681     name-sg = exercise ,
2682     Name-pl = Exercises ,
2683     name-pl = exercises ,
2684
2685 type = solution ,
2686     Name-sg = Solution ,
2687     name-sg = solution ,
2688     Name-pl = Solutions ,
2689     name-pl = solutions ,
2690 </dict-english>

```

10.2 German

```

2691 <package>\zcDeclareLanguage { german }
2692 <package>\zcDeclareLanguageAlias { austrian      } { german }
2693 <package>\zcDeclareLanguageAlias { germanb       } { german }
2694 <package>\zcDeclareLanguageAlias { ngerman       } { german }
2695 <package>\zcDeclareLanguageAlias { naustrian     } { german }
2696 <package>\zcDeclareLanguageAlias { nswissgerman } { german }
2697 <package>\zcDeclareLanguageAlias { swissgerman  } { german }
2698 <*dict-german>

```

```

2699 namesep = {\nobreakspace} ,
2700 pairsep = {\~und\nobreakspace} ,
2701 listsep = {,~} ,
2702 lastsep = {\~und\nobreakspace} ,
2703 tpairsep = {\~und\nobreakspace} ,
2704 tlistsep = {,~} ,
2705 tlastsep = {\~und\nobreakspace} ,
2706 notesep = {\~} ,
2707 rangesep = {\~bis\nobreakspace} ,
2708
2709 type = part ,
2710     Name-sg = Teil ,
2711     name-sg = Teil ,
2712     Name-pl = Teile ,
2713     name-pl = Teile ,
2714
2715 type = chapter ,
2716     Name-sg = Kapitel ,
2717     name-sg = Kapitel ,
2718     Name-pl = Kapitel ,
2719     name-pl = Kapitel ,
2720
2721 type = section ,
2722     Name-sg = Abschnitt ,
2723     name-sg = Abschnitt ,
2724     Name-pl = Abschnitte ,
2725     name-pl = Abschnitte ,
2726
2727 type = paragraph ,
2728     Name-sg = Absatz ,
2729     name-sg = Absatz ,
2730     Name-pl = Absätze ,
2731     name-pl = Absätze ,
2732
2733 type = appendix ,
2734     Name-sg = Anhang ,
2735     name-sg = Anhang ,
2736     Name-pl = Anhänge ,
2737     name-pl = Anhänge ,
2738
2739 type = page ,
2740     Name-sg = Seite ,
2741     name-sg = Seite ,
2742     Name-pl = Seiten ,
2743     name-pl = Seiten ,
2744
2745 type = line ,
2746     Name-sg = Zeile ,
2747     name-sg = Zeile ,
2748     Name-pl = Zeilen ,
2749     name-pl = Zeilen ,
2750
2751 type = figure ,
2752     Name-sg = Abbildung ,

```

```

2753 name-sg = Abbildung ,
2754 Name-pl = Abbildungen ,
2755 name-pl = Abbildungen ,
2756 Name-sg-ab = Abb. ,
2757 name-sg-ab = Abb. ,
2758 Name-pl-ab = Abb. ,
2759 name-pl-ab = Abb. ,
2760
2761 type = table ,
2762 Name-sg = Tabelle ,
2763 name-sg = Tabelle ,
2764 Name-pl = Tabellen ,
2765 name-pl = Tabellen ,
2766
2767 type = item ,
2768 Name-sg = Punkt ,
2769 name-sg = Punkt ,
2770 Name-pl = Punkte ,
2771 name-pl = Punkte ,
2772
2773 type = footnote ,
2774 Name-sg = Fußnote ,
2775 name-sg = Fußnote ,
2776 Name-pl = Fußnoten ,
2777 name-pl = Fußnoten ,
2778
2779 type = note ,
2780 Name-sg = Anmerkung ,
2781 name-sg = Anmerkung ,
2782 Name-pl = Anmerkungen ,
2783 name-pl = Anmerkungen ,
2784
2785 type = equation ,
2786 Name-sg = Gleichung ,
2787 name-sg = Gleichung ,
2788 Name-pl = Gleichungen ,
2789 name-pl = Gleichungen ,
2790 refpre-in = {} ,
2791 refpos-in = {} ,
2792
2793 type = theorem ,
2794 Name-sg = Theorem ,
2795 name-sg = Theorem ,
2796 Name-pl = Theoreme ,
2797 name-pl = Theoreme ,
2798
2799 type = lemma ,
2800 Name-sg = Lemma ,
2801 name-sg = Lemma ,
2802 Name-pl = Lemmata ,
2803 name-pl = Lemmata ,
2804
2805 type = corollary ,
2806 Name-sg = Korollar ,

```

```

2807     name-sg = Korollar ,
2808     Name-pl = Korollare ,
2809     name-pl = Korollare ,
2810
2811 type = proposition ,
2812     Name-sg = Satz ,
2813     name-sg = Satz ,
2814     Name-pl = Sätze ,
2815     name-pl = Sätze ,
2816
2817 type = definition ,
2818     Name-sg = Definition ,
2819     name-sg = Definition ,
2820     Name-pl = Definitionen ,
2821     name-pl = Definitionen ,
2822
2823 type = proof ,
2824     Name-sg = Beweis ,
2825     name-sg = Beweis ,
2826     Name-pl = Beweise ,
2827     name-pl = Beweise ,
2828
2829 type = result ,
2830     Name-sg = Ergebnis ,
2831     name-sg = Ergebnis ,
2832     Name-pl = Ergebnisse ,
2833     name-pl = Ergebnisse ,
2834
2835 type = example ,
2836     Name-sg = Beispiel ,
2837     name-sg = Beispiel ,
2838     Name-pl = Beispiele ,
2839     name-pl = Beispiele ,
2840
2841 type = remark ,
2842     Name-sg = Bemerkung ,
2843     name-sg = Bemerkung ,
2844     Name-pl = Bemerkungen ,
2845     name-pl = Bemerkungen ,
2846
2847 type = algorithm ,
2848     Name-sg = Algorithmus ,
2849     name-sg = Algorithmus ,
2850     Name-pl = Algorithmen ,
2851     name-pl = Algorithmen ,
2852
2853 type = listing ,
2854     Name-sg = Listing , % CHECK
2855     name-sg = Listing , % CHECK
2856     Name-pl = Listings , % CHECK
2857     name-pl = Listings , % CHECK
2858
2859 type = exercise ,
2860     Name-sg = Übungsaufgabe ,

```

```

2861 name-sg = Übungsaufgabe ,
2862 Name-pl = Übungsaufgaben ,
2863 name-pl = Übungsaufgaben ,
2864
2865 type = solution ,
2866 Name-sg = Lösung ,
2867 name-sg = Lösung ,
2868 Name-pl = Lösungen ,
2869 name-pl = Lösungen ,
2870 </dict-german>

```

10.3 French

```

2871 <package>\zcDeclareLanguage { french }
2872 <package>\zcDeclareLanguageAlias { acadian } { french }
2873 <package>\zcDeclareLanguageAlias { canadien } { french }
2874 <package>\zcDeclareLanguageAlias { francais } { french }
2875 <package>\zcDeclareLanguageAlias { frenchb } { french }
2876 <*dict-french>
2877 namesep = {\nobreakspace} ,
2878 pairsep = {\~et\nobreakspace} ,
2879 listsep = {\~,~} ,
2880 lastsep = {\~et\nobreakspace} ,
2881 tpairsep = {\~et\nobreakspace} ,
2882 tlistsep = {\~,~} ,
2883 tlastsep = {\~et\nobreakspace} ,
2884 notesep = {\~} ,
2885 rangesep = {\~à\nobreakspace} ,
2886
2887 type = part ,
2888 Name-sg = Partie ,
2889 name-sg = partie ,
2890 Name-pl = Parties ,
2891 name-pl = parties ,
2892
2893 type = chapter ,
2894 Name-sg = Chapitre ,
2895 name-sg = chapitre ,
2896 Name-pl = Chapitres ,
2897 name-pl = chapitres ,
2898
2899 type = section ,
2900 Name-sg = Section ,
2901 name-sg = section ,
2902 Name-pl = Sections ,
2903 name-pl = sections ,
2904
2905 type = paragraph ,
2906 Name-sg = Paragraphe ,
2907 name-sg = paragraphe ,
2908 Name-pl = Paragraphes ,
2909 name-pl = paragraphes ,
2910
2911 type = appendix ,

```



```

2912     Name-sg = Annexe ,
2913     name-sg = annexe ,
2914     Name-pl = Annexes ,
2915     name-pl = annexes ,
2916
2917 type = page ,
2918     Name-sg = Page ,
2919     name-sg = page ,
2920     Name-pl = Pages ,
2921     name-pl = pages ,
2922
2923 type = line ,
2924     Name-sg = Ligne ,
2925     name-sg = ligne ,
2926     Name-pl = Lignes ,
2927     name-pl = lignes ,
2928
2929 type = figure ,
2930     Name-sg = Figure ,
2931     name-sg = figure ,
2932     Name-pl = Figures ,
2933     name-pl = figures ,
2934
2935 type = table ,
2936     Name-sg = Table ,
2937     name-sg = table ,
2938     Name-pl = Tables ,
2939     name-pl = tables ,
2940
2941 type = item ,
2942     Name-sg = Point ,
2943     name-sg = point ,
2944     Name-pl = Points ,
2945     name-pl = points ,
2946
2947 type = footnote ,
2948     Name-sg = Note ,
2949     name-sg = note ,
2950     Name-pl = Notes ,
2951     name-pl = notes ,
2952
2953 type = note ,
2954     Name-sg = Note ,
2955     name-sg = note ,
2956     Name-pl = Notes ,
2957     name-pl = notes ,
2958
2959 type = equation ,
2960     Name-sg = Équation ,
2961     name-sg = équation ,
2962     Name-pl = Équations ,
2963     name-pl = équations ,
2964     refpre-in = {() ,
2965     refpos-in = {)} ,

```

```

2966
2967 type = theorem ,
2968     Name-sg = Théorème ,
2969     name-sg = théorème ,
2970     Name-pl = Théorèmes ,
2971     name-pl = théorèmes ,
2972
2973 type = lemma ,
2974     Name-sg = Lemme ,
2975     name-sg = lemme ,
2976     Name-pl = Lemmes ,
2977     name-pl = lemmes ,
2978
2979 type = corollary ,
2980     Name-sg = Corollaire ,
2981     name-sg = corollaire ,
2982     Name-pl = Corollaires ,
2983     name-pl = corollaires ,
2984
2985 type = proposition ,
2986     Name-sg = Proposition ,
2987     name-sg = proposition ,
2988     Name-pl = Propositions ,
2989     name-pl = propositions ,
2990
2991 type = definition ,
2992     Name-sg = Définition ,
2993     name-sg = définition ,
2994     Name-pl = Définitions ,
2995     name-pl = définitions ,
2996
2997 type = proof ,
2998     Name-sg = Démonstration ,
2999     name-sg = démonstration ,
3000     Name-pl = Démonstrations ,
3001     name-pl = démonstrations ,
3002
3003 type = result ,
3004     Name-sg = Résultat ,
3005     name-sg = résultat ,
3006     Name-pl = Résultats ,
3007     name-pl = résultats ,
3008
3009 type = example ,
3010     Name-sg = Exemple ,
3011     name-sg = exemple ,
3012     Name-pl = Exemples ,
3013     name-pl = exemples ,
3014
3015 type = remark ,
3016     Name-sg = Remarque ,
3017     name-sg = remarque ,
3018     Name-pl = Remarques ,
3019     name-pl = remarques ,

```

```

3020
3021 type = algorithm ,
3022   Name-sg = Algorithme ,
3023   name-sg = algorithme ,
3024   Name-pl = Algorithmes ,
3025   name-pl = algorithmes ,
3026
3027 type = listing ,
3028   Name-sg = Liste ,
3029   name-sg = liste ,
3030   Name-pl = Listes ,
3031   name-pl = listes ,
3032
3033 type = exercise ,
3034   Name-sg = Exercice ,
3035   name-sg = exercice ,
3036   Name-pl = Exercices ,
3037   name-pl = exercices ,
3038
3039 type = solution ,
3040   Name-sg = Solution ,
3041   name-sg = solution ,
3042   Name-pl = Solutions ,
3043   name-pl = solutions ,
3044 </dict-french>

```

10.4 Portuguese

```

3045 <package>\zcDeclareLanguage { portuguese }
3046 <package>\zcDeclareLanguageAlias { brazilian } { portuguese }
3047 <package>\zcDeclareLanguageAlias { brazil   } { portuguese }
3048 <package>\zcDeclareLanguageAlias { portuges } { portuguese }
3049 <*dict-portuguese>
3050
3051 namesep = {\nobreakspace} ,
3052 pairsep = {\nobreakspace} ,
3053 listsep = {,~} ,
3054 lastsep = {\nobreakspace} ,
3055 tpairsep = {\nobreakspace} ,
3056 tlistsep = {,~} ,
3057 tlastsep = {\nobreakspace} ,
3058 notesep = {~} ,
3059 rangesep = {\nobreakspace} ,
3060
3061 type = part ,
3062   Name-sg = Parte ,
3063   name-sg = parte ,
3064   Name-pl = Partes ,
3065   name-pl = partes ,
3066
3067 type = chapter ,
3068   Name-sg = Capítulo ,
3069   name-sg = capítulo ,
3070   Name-pl = Capítulos ,
3071   name-pl = capítulos ,

```

```

3071
3072 type = section ,
3073     Name-sg = Seção ,
3074     name-sg = seção ,
3075     Name-pl = Seções ,
3076     name-pl = seções ,
3077
3078 type = paragraph ,
3079     Name-sg = Parágrafo ,
3080     name-sg = parágrafo ,
3081     Name-pl = Parágrafos ,
3082     name-pl = parágrafos ,
3083     Name-sg-ab = Par. ,
3084     name-sg-ab = par. ,
3085     Name-pl-ab = Par. ,
3086     name-pl-ab = par. ,
3087
3088 type = appendix ,
3089     Name-sg = Apêndice ,
3090     name-sg = apêndice ,
3091     Name-pl = Apêndices ,
3092     name-pl = apêndices ,
3093
3094 type = page ,
3095     Name-sg = Página ,
3096     name-sg = página ,
3097     Name-pl = Páginas ,
3098     name-pl = páginas ,
3099     name-sg-ab = p. ,
3100     name-pl-ab = pp. ,
3101
3102 type = line ,
3103     Name-sg = Linha ,
3104     name-sg = linha ,
3105     Name-pl = Linhas ,
3106     name-pl = linhas ,
3107
3108 type = figure ,
3109     Name-sg = Figura ,
3110     name-sg = figura ,
3111     Name-pl = Figuras ,
3112     name-pl = figuras ,
3113     Name-sg-ab = Fig. ,
3114     name-sg-ab = fig. ,
3115     Name-pl-ab = Figs. ,
3116     name-pl-ab = figs. ,
3117
3118 type = table ,
3119     Name-sg = Tabela ,
3120     name-sg = tabela ,
3121     Name-pl = Tabelas ,
3122     name-pl = tabelas ,
3123
3124 type = item ,

```

```

3125     Name-sg = Item ,
3126     name-sg = item ,
3127     Name-pl = Itens ,
3128     name-pl = itens ,
3129
3130 type = footnote ,
3131     Name-sg = Nota ,
3132     name-sg = nota ,
3133     Name-pl = Notas ,
3134     name-pl = notas ,
3135
3136 type = note ,
3137     Name-sg = Nota ,
3138     name-sg = nota ,
3139     Name-pl = Notas ,
3140     name-pl = notas ,
3141
3142 type = equation ,
3143     Name-sg = Equação ,
3144     name-sg = equação ,
3145     Name-pl = Equações ,
3146     name-pl = equações ,
3147     Name-sg-ab = Eq. ,
3148     name-sg-ab = eq. ,
3149     Name-pl-ab = Eqs. ,
3150     name-pl-ab = eqs. ,
3151     refpres-in = {} ,
3152     refpos-in = {} ,
3153
3154 type = theorem ,
3155     Name-sg = Teorema ,
3156     name-sg = teorema ,
3157     Name-pl = Teoremas ,
3158     name-pl = teoremas ,
3159
3160 type = lemma ,
3161     Name-sg = Lema ,
3162     name-sg = lema ,
3163     Name-pl = Lemas ,
3164     name-pl = lemas ,
3165
3166 type = corollary ,
3167     Name-sg = Corolário ,
3168     name-sg = corolário ,
3169     Name-pl = Corolários ,
3170     name-pl = corolários ,
3171
3172 type = proposition ,
3173     Name-sg = Proposição ,
3174     name-sg = proposição ,
3175     Name-pl = Proposições ,
3176     name-pl = proposições ,
3177
3178 type = definition ,

```

```

3179     Name-sg = Definição ,
3180     name-sg = definição ,
3181     Name-pl = Definições ,
3182     name-pl = definições ,
3183
3184     type = proof ,
3185     Name-sg = Demonstração ,
3186     name-sg = demonstração ,
3187     Name-pl = Demonstrações ,
3188     name-pl = demonstrações ,
3189
3190     type = result ,
3191     Name-sg = Resultado ,
3192     name-sg = resultado ,
3193     Name-pl = Resultados ,
3194     name-pl = resultados ,
3195
3196     type = example ,
3197     Name-sg = Exemplo ,
3198     name-sg = exemplo ,
3199     Name-pl = Exemplos ,
3200     name-pl = exemplos ,
3201
3202     type = remark ,
3203     Name-sg = Observação ,
3204     name-sg = observação ,
3205     Name-pl = Observações ,
3206     name-pl = observações ,
3207
3208     type = algorithm ,
3209     Name-sg = Algoritmo ,
3210     name-sg = algoritmo ,
3211     Name-pl = Algoritmos ,
3212     name-pl = algoritmos ,
3213
3214     type = listing ,
3215     Name-sg = Listagem ,
3216     name-sg = listagem ,
3217     Name-pl = Listagens ,
3218     name-pl = listagens ,
3219
3220     type = exercise ,
3221     Name-sg = Exercício ,
3222     name-sg = exercício ,
3223     Name-pl = Exercícios ,
3224     name-pl = exercícios ,
3225
3226     type = solution ,
3227     Name-sg = Solução ,
3228     name-sg = solução ,
3229     Name-pl = Soluções ,
3230     name-pl = soluções ,
3231 </dict-portuguese>

```

10.5 Spanish

```
3232 <package>\zcDeclareLanguage { spanish }
3233 <*dict-spanish>
3234 namesep = {\nobreakspace} ,
3235 pairsep = {\~y\nobreakspace} ,
3236 listsep = {,~} ,
3237 lastsep = {\~y\nobreakspace} ,
3238 tpairsep = {\~y\nobreakspace} ,
3239 tlistsep = {,~} ,
3240 tlastsep = {\~y\nobreakspace} ,
3241 notesep = {~} ,
3242 rangesep = {\~a\nobreakspace} ,
3243
3244 type = part ,
3245   Name-sg = Parte ,
3246   name-sg = parte ,
3247   Name-pl = Partes ,
3248   name-pl = partes ,
3249
3250 type = chapter ,
3251   Name-sg = Capítulo ,
3252   name-sg = capítulo ,
3253   Name-pl = Capítulos ,
3254   name-pl = capítulos ,
3255
3256 type = section ,
3257   Name-sg = Sección ,
3258   name-sg = sección ,
3259   Name-pl = Secciones ,
3260   name-pl = secciones ,
3261
3262 type = paragraph ,
3263   Name-sg = Párrafo ,
3264   name-sg = párrafo ,
3265   Name-pl = Párrafos ,
3266   name-pl = párrafos ,
3267
3268 type = appendix ,
3269   Name-sg = Apéndice ,
3270   name-sg = apéndice ,
3271   Name-pl = Apéndices ,
3272   name-pl = apéndices ,
3273
3274 type = page ,
3275   Name-sg = Página ,
3276   name-sg = página ,
3277   Name-pl = Páginas ,
3278   name-pl = páginas ,
3279
3280 type = line ,
3281   Name-sg = Línea ,
3282   name-sg = línea ,
3283   Name-pl = Líneas ,
```

```

3284     name-pl = líneas ,
3285
3286 type = figure ,
3287     Name-sg = Figura ,
3288     name-sg = figura ,
3289     Name-pl = Figuras ,
3290     name-pl = figuras ,
3291
3292 type = table ,
3293     Name-sg = Cuadro ,
3294     name-sg = cuadro ,
3295     Name-pl = Cuadros ,
3296     name-pl = cuadros ,
3297
3298 type = item ,
3299     Name-sg = Punto ,
3300     name-sg = punto ,
3301     Name-pl = Puntos ,
3302     name-pl = puntos ,
3303
3304 type = footnote ,
3305     Name-sg = Nota ,
3306     name-sg = nota ,
3307     Name-pl = Notas ,
3308     name-pl = notas ,
3309
3310 type = note ,
3311     Name-sg = Nota ,
3312     name-sg = nota ,
3313     Name-pl = Notas ,
3314     name-pl = notas ,
3315
3316 type = equation ,
3317     Name-sg = Ecuación ,
3318     name-sg = ecuación ,
3319     Name-pl = Ecuaciones ,
3320     name-pl = ecuaciones ,
3321     refpre-in = {(} ,
3322     refpos-in = {)} ,
3323
3324 type = theorem ,
3325     Name-sg = Teorema ,
3326     name-sg = teorema ,
3327     Name-pl = Teoremas ,
3328     name-pl = teoremas ,
3329
3330 type = lemma ,
3331     Name-sg = Lema ,
3332     name-sg = lema ,
3333     Name-pl = Lemas ,
3334     name-pl = lemas ,
3335
3336 type = corollary ,
3337     Name-sg = Corolario ,

```



```

3338     name-sg = corolario ,
3339     Name-pl = Corolarios ,
3340     name-pl = corolarios ,
3341
3342 type = proposition ,
3343     Name-sg = Proposición ,
3344     name-sg = proposición ,
3345     Name-pl = Proposiciones ,
3346     name-pl = proposiciones ,
3347
3348 type = definition ,
3349     Name-sg = Definición ,
3350     name-sg = definición ,
3351     Name-pl = Definiciones ,
3352     name-pl = definiciones ,
3353
3354 type = proof ,
3355     Name-sg = Demostración ,
3356     name-sg = demostración ,
3357     Name-pl = Demostraciones ,
3358     name-pl = demostraciones ,
3359
3360 type = result ,
3361     Name-sg = Resultado ,
3362     name-sg = resultado ,
3363     Name-pl = Resultados ,
3364     name-pl = resultados ,
3365
3366 type = example ,
3367     Name-sg = Ejemplo ,
3368     name-sg = ejemplo ,
3369     Name-pl = Ejemplos ,
3370     name-pl = ejemplos ,
3371
3372 type = remark ,
3373     Name-sg = Observación ,
3374     name-sg = observación ,
3375     Name-pl = Observaciones ,
3376     name-pl = observaciones ,
3377
3378 type = algorithm ,
3379     Name-sg = Algoritmo ,
3380     name-sg = algoritmo ,
3381     Name-pl = Algoritmos ,
3382     name-pl = algoritmos ,
3383
3384 type = listing ,
3385     Name-sg = Listado ,
3386     name-sg = listado ,
3387     Name-pl = Listados ,
3388     name-pl = listados ,
3389
3390 type = exercise ,
3391     Name-sg = Ejercicio ,

```

```

3392 name-sg = ejercicio ,
3393 Name-pl = Ejercicios ,
3394 name-pl = ejercicios ,
3395
3396 type = solution ,
3397 Name-sg = Solución ,
3398 name-sg = solución ,
3399 Name-pl = Soluciones ,
3400 name-pl = soluciones ,
3401 </dict-spanish>

```

Index

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

Symbols

\\ 103, 109, 118, 119, 124, 125, 134, 135, 145

A

\AddToHook 91,
 532, 547, 657, 725, 750, 779, 781, 832
 \appendix 64
 \appendixname 64
 \Arg 2368

B

\babelname 735
 \babelprovide 12, 24
 bool commands:
 \bool_case_true: 2
 \bool_if:NTF 288, 297, 661, 665, 1509,
 1605, 1739, 1761, 1791, 1848, 1889,
 1912, 1916, 1922, 1932, 1938, 2221
 \bool_if:nTF . 59, 1147, 1156, 1165,
 1233, 1261, 1284, 1375, 1383, 1523,
 1531, 1772, 1779, 1786, 2047, 2305
 \bool_lazy_all:nTF 2373
 \bool_lazy_and:nnTF
 1034, 1049, 2113, 2425
 \bool_lazy_any:nTF 2189, 2198
 \bool_lazy_or:nnTF 1038, 2101
 \bool_new:N 251,
 568, 569, 594, 618, 627, 634, 635,
 690, 691, 708, 709, 825, 826, 1060,
 1079, 1415, 1416, 1427, 1428, 1446
 \bool_set:Nn 1032
 \bool_set_false:N
 581, 585, 642, 651, 652,
 667, 847, 1225, 1472, 1515, 1529,
 1543, 1751, 1887, 1888, 2196, 2213

\bool_set_true:N
 . 306, 575, 576, 580, 586, 641, 646,
 647, 836, 841, 1245, 1256, 1273,
 1279, 1296, 1302, 1330, 1342, 1480,
 1510, 1516, 1520, 1547, 1553, 2212,
 2458, 2465, 2466, 2484, 2491, 2492
 \bool_until_do:Nn 1226, 1473

C

clist commands:
 \clist_map_inline:nn 477
 \counterwithin 4
 cs commands:
 \cs_generate_variant:Nn
 55, 56, 302, 310, 943, 949, 1080, 2092
 \cs_if_exist:NTF 39, 48, 69
 \cs_new:Npn 37, 46, 57, 67, 78, 2043, 2216
 \cs_new_protected:Npn
 252, 303, 311, 317, 438,
 938, 944, 1027, 1081, 1097, 1140,
 1206, 1350, 1406, 1449, 1616, 1883,
 2039, 2041, 2093, 2369, 2421, 2446
 \cs_new_protected:Npx 90
 \cs_set_eq:NN 94

E

\endinput 12
 exp commands:
 \exp_args:NNe 27
 \exp_args:NNnx 242
 \exp_args:NnV 280
 \exp_args:NNx 95, 1270, 1293
 \exp_args:Nnx 313
 \exp_args:Nx 262
 \exp_args:Nxx
 1190, 1241, 2450, 2472, 2476

<code>\exp_not:N</code>	<code>\int_incr:N</code> 1878, 1915, 1917, 1931, 1933, 1937, 1939, 2037
. . . 1793, 1796, 1818, 1821, 1824, 2053, 2056, 2059, 2072, 2074, 2077, 2080, 2085, 2087, 2090, 2219, 2228, 2249, 2252, 2254, 2257, 2264, 2271, 2273, 2277, 2280, 2283, 2285, 2291, 2295, 2298, 2311, 2314, 2317, 2344, 2346, 2349, 2352, 2359, 2361, 2364	<code>\int_new:N</code>
<code>\exp_not:n</code> . 1638, 1654, 1666, 1671, 1694, 1708, 1712, 1724, 1728, 1762, 1763, 1794, 1817, 1822, 1823, 1952, 1965, 1972, 1996, 2008, 2012, 2022, 2026, 2054, 2055, 2057, 2067, 2070, 2073, 2078, 2079, 2081, 2082, 2084, 2086, 2250, 2251, 2253, 2255, 2256, 2258, 2259, 2263, 2275, 2276, 2281, 2282, 2284, 2292, 2296, 2297, 2299, 2312, 2313, 2315, 2338, 2342, 2345, 2350, 2351, 2353, 2354, 2358, 2360	. . 1077, 1078, 1422, 1423, 1424, 1425
<code>\ExplSyntaxOn</code> 12, 264	<code>\int_set:Nn</code> . . . 1362, 1364, 1368, 1371
	<code>\int_use:N</code> 33, 35, 50
	<code>\int_zero:N</code>
	. . . 1352, 1353, 1458, 1459, 1460, 1461, 1877, 1879, 1880, 2032, 2033
	iow commands:
	<code>\iow_char:N</code> 103, 109, 118, 119, 124, 125, 134, 135, 145
	K
	keys commands:
	<code>\keys_define:nn</code>
	. 29, 323, 335, 352, 366, 445, 473, 499, 523, 551, 558, 570, 595, 604, 619, 628, 636, 669, 676, 692, 710, 746, 784, 817, 820, 827, 837, 848, 859, 870, 888, 900, 950, 962, 983, 1006
	<code>\keys_set:nn</code>
	. 29, 33, 281, 842, 877, 883, 932, 1030
	keyval commands:
	<code>\keyval_parse:nnn</code> 449, 503
	L
	<code>\labelformat</code> 3
	<code>\languagename</code> 24, 729
	M
	<code>\mainbabelname</code> 24, 736
	<code>\MessageBreak</code> 10
	msg commands:
	<code>\msg_info:nnn</code> 343, 373
	<code>\msg_line_context:</code>
 102, 108, 112, 130, 133, 139, 153, 157, 159, 161, 164, 168
	<code>\msg_new:nnn</code> . . . 100, 106, 111, 113, 115, 121, 127, 129, 131, 137, 142, 147, 149, 151, 156, 158, 160, 162, 167
	<code>\msg_note:nnn</code> 284
	<code>\msg_warning:nn</code>
 537, 562, 666, 672, 830, 851
	<code>\msg_warning:nnn</code>
	. 231, 246, 290, 298, 505, 892, 934, 974, 1013, 1568, 1746, 2144, 2179, 2413
	<code>\msg_warning:nnnn</code>
 451, 1254, 1277, 1300, 1340
	N
	<code>\newcounter</code> 4
	<code>\NewDocumentCommand</code>
	. . 226, 236, 876, 878, 925, 1025, 1061
	<code>\newtheorem</code> 65
<code>\exp_not:N</code>	
. . . 1793, 1796, 1818, 1821, 1824, 2053, 2056, 2059, 2072, 2074, 2077, 2080, 2085, 2087, 2090, 2219, 2228, 2249, 2252, 2254, 2257, 2264, 2271, 2273, 2277, 2280, 2283, 2285, 2291, 2295, 2298, 2311, 2314, 2317, 2344, 2346, 2349, 2352, 2359, 2361, 2364	
<code>\exp_not:n</code> . 1638, 1654, 1666, 1671, 1694, 1708, 1712, 1724, 1728, 1762, 1763, 1794, 1817, 1822, 1823, 1952, 1965, 1972, 1996, 2008, 2012, 2022, 2026, 2054, 2055, 2057, 2067, 2070, 2073, 2078, 2079, 2081, 2082, 2084, 2086, 2250, 2251, 2253, 2255, 2256, 2258, 2259, 2263, 2275, 2276, 2281, 2282, 2284, 2292, 2296, 2297, 2299, 2312, 2313, 2315, 2338, 2342, 2345, 2350, 2351, 2353, 2354, 2358, 2360	
<code>\ExplSyntaxOn</code> 12, 264	
F	
file commands:	
<code>\file_get:nnNTF</code> 262	
<code>\fmtversion</code> 3	
G	
group commands:	
<code>\group_begin:</code> . . 93, 254, 305, 927, 1029, 1042, 1793, 1821, 2053, 2056, 2077, 2080, 2249, 2254, 2257, 2273, 2280, 2295, 2311, 2314, 2349, 2352	
<code>\group_end:</code> 96, 300, 308, 935, 1045, 1057, 1818, 1824, 2072, 2074, 2085, 2087, 2252, 2264, 2271, 2277, 2283, 2298, 2344, 2346, 2359, 2361	
H	
<code>\hyperlink</code> 55	
I	
<code>\IfBooleanTF</code> 1063	
<code>\IfFormatAtLeastTF</code> 3, 4	
<code>\input</code> 12	
int commands:	
<code>\int_case:nnTF</code>	
. . 1619, 1647, 1679, 1851, 1945, 1984	
<code>\int_compare:nNnTF</code> 1194, 1246, 1315, 1331, 1361, 1363, 1408, 1575, 1634, 1668, 1840, 1842, 1900, 1925, 1969, 2454, 2460, 2480, 2486	
<code>\int_compare_p:nNn</code>	
. 1377, 1385, 2105, 2116, 2209	
<code>\int_eval:n</code> 90	

<code>\nobreakspace</code>	387, 2509, 2510, 2512, 2513, 2515, 2517, 2699, 2700, 2702, 2703, 2705, 2707, 2877, 2878, 2880, 2881, 2883, 2885, 3050, 3051, 3053, 3054, 3056, 3058, 3234, 3235, 3237, 3238, 3240, 3242
P	
<code>\PackageError</code>	7
<code>\pagenumbering</code>	6
prg commands:	
<code>\prg_generate_conditional_-</code> variant:Nnn	411, 427
<code>\prg_new_protected_conditional:Npnn</code>	397, 413, 430
<code>\prg_return_false:</code>	407, 409, 423, 425, 436
<code>\prg_return_true:</code>	406, 422, 435
<code>\ProcessKeysOptions</code>	875
prop commands:	
<code>\prop_get:NnN</code>	2393
<code>\prop_get:NnNTF</code>	255, 400, 403, 416, 419, 433, 928, 2129, 2150, 2158, 2371, 2423, 2435
<code>\prop_gput:Nnn</code>	232, 243, 940, 946
<code>\prop_gput_if_new:Nnn</code>	313, 319
<code>\prop_gset_from_keyval:Nn</code>	381
<code>\prop_if_exist:NTF</code>	267, 880
<code>\prop_if_exist_p:N</code>	2377, 2428
<code>\prop_if_in:NnTF</code>	25, 230, 240
<code>\prop_if_in_p:Nn</code>	60, 2384
<code>\prop_item:Nn</code>	27, 61, 244
<code>\prop_new:N</code>	225, 273, 380, 444, 498, 855, 881
<code>\prop_put:Nnn</code>	442, 866, 915
<code>\prop_remove:Nn</code>	441, 865, 907
<code>\providecommand</code>	3
<code>\ProvidesExplPackage</code>	14
<code>\ProvidesFile</code>	12
R	
<code>\refstepcounter</code>	3
<code>\RequirePackage</code>	16, 17, 18, 19, 20, 662
S	
seq commands:	
<code>\seq_clear:N</code>	615, 1099
<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>	1483
<code>\seq_gput_right:Nn</code>	282
<code>\seq_if_empty:NTF</code>	1477
<code>\seq_if_in:NnTF</code>	258, 479, 1087
<code>\seq_map_break:n</code>	81, 1397, 1400
<code>\seq_map_function:NN</code>	1102
<code>\seq_map_indexed_inline:Nn</code>	21, 1357
<code>\seq_map_inline:Nn</code>	332, 349, 363, 856, 885, 897, 959, 980, 1003, 1394
<code>\seq_map_tokens:Nn</code>	63
<code>\seq_new:N</code>	211, 218, 250, 472, 603, 1059, 1096, 1417
<code>\seq_pop_left:NN</code>	1475
<code>\seq_put_right:Nn</code>	481, 1091
<code>\seq_reverse:N</code>	609
<code>\seq_set_eq:NN</code>	1451
<code>\seq_set_from_clist:Nn</code>	608, 1031
<code>\seq_sort:Nn</code>	1105
sort commands:	
<code>\sort_return_same:</code>	36, 41, 1112, 1117, 1154, 1199, 1201, 1251, 1257, 1274, 1280, 1303, 1336, 1343, 1381, 1397, 1413
<code>\sort_return_swapped:</code>	36, 41, 1125, 1163, 1198, 1250, 1297, 1335, 1389, 1400, 1412
str commands:	
<code>\str_case:nnTF</code>	752, 788
<code>\str_if_eq:nnTF</code>	80
<code>\str_if_eq_p:nn</code>	2194, 2200, 2202, 2206
<code>\str_new:N</code>	675
<code>\str_set:Nn</code>	680, 682, 684, 686
T	
T _E X and L ^A T _E X 2 _ε commands:	
<code>\@Alph</code>	64
<code>\@addtoreset</code>	4
<code>\@chapapp</code>	64
<code>\@currentcounter</code>	4, 21, 25, 28, 30, 33, 84, 86
<code>\@currentlabel</code>	3
<code>\@ifl@t@r</code>	3
<code>\@ifpackageloaded</code>	534, 549, 659, 727, 733, 834
<code>\@onlypreamble</code>	235, 249, 937
<code>\bbl@loaded</code>	24
<code>\bbl@main@language</code>	24, 730
<code>\c@</code>	3
<code>\c@page</code>	6, 94
<code>\cl@</code>	4
<code>\hyper@@link</code>	1796, 2059, 2228, 2317
<code>\p@...</code>	3
<code>\zref@addprop</code>	22, 32, 34, 36, 87, 88, 99
<code>\zref@default</code>	2040, 2042
<code>\zref@extractdefault</code>	55, 1084, 1143, 1145, 1191, 1192, 1195, 1197, 1209, 1213, 1217, 1221, 1242, 1243, 1247, 1249, 1269,

\l_zrefclever_capitalize_bool ..	_zrefclever_get_type_transl:nnnN
..... 690, 694, 2102 9, 398, 412
\l_zrefclever_capitalize_first_-	_zrefclever_get_type_transl:nnnNTF
bool 691, 700, 2104 16, 397, 2137, 2166, 2172, 2401
_zrefclever_counter_reset_by:n	\l_zrefclever_label_a_tl
.. 5, 18, 19, 39, 41, 43, 48, 50, 52, 57	... 1067, 1476, 1496, 1512, 1562,
_zrefclever_counter_reset_by_-	1563, 1569, 1626, 1639, 1655, 1672,
aux:nn 64, 67	1713, 1729, 1757, 1764, 1891, 1895,
_zrefclever_counter_reset_by_-	1905, 1930, 1953, 1974, 2013, 2027
aux:nnn 74, 78	\l_zrefclever_label_b_tl . 1067,
\l_zrefclever_counter_resetby_-	1479, 1484, 1501, 1514, 1519, 1895
prop 4, 19, 60, 61, 498, 510	\l_zrefclever_label_count_int ..
\l_zrefclever_counter_resettters_- 42, 1422,
seq 4, 18, 19, 63, 472, 479, 482	1458, 1575, 1619, 1877, 1900, 2037
\l_zrefclever_counter_type_prop	\l_zrefclever_label_enclcnt_a_-
..... 3, 18, 25, 28, 444, 456	tl 1067, 1208,
\l_zrefclever_current_language_-	1210, 1211, 1229, 1294, 1320, 1321
tl .. 24, 724, 729, 735, 739, 765, 801	\l_zrefclever_label_enclcnt_b_-
_zrefclever_declare_default_-	tl 1067, 1212,
transl:nnn 31, 938, 969, 990	1214, 1215, 1231, 1271, 1322, 1323
_zrefclever_declare_type_-	\l_zrefclever_label_enclhead_a_-
transl:nnnn 31, 938, 995, 1017	tl 1067, 1228, 1237, 1265, 1312
\g_zrefclever_dict_⟨language⟩_prop	\l_zrefclever_label_enclhead_b_-
..... 12	tl 1067, 1230, 1238, 1288, 1313
\l_zrefclever_dict_language_tl .	\l_zrefclever_label_enclval_a_-
..... 169, 256, 260, 263, 270,	tl 1067, 1216,
276, 283, 285, 291, 314, 320, 401,	1218, 1219, 1316, 1324, 1325, 1332
404, 417, 420, 929, 970, 991, 996, 1018	\l_zrefclever_label_enclval_b_-
\g_zrefclever_fallback_dict_-	tl 1067, 1220,
prop 9, 380, 381, 433	1222, 1223, 1318, 1326, 1327, 1334
_zrefclever_get_default_-	\l_zrefclever_label_type_a_tl ..
transl:nnN 9, 414, 428 1067, 1083, 1085, 1089,
_zrefclever_get_default_-	1092, 1142, 1151, 1160, 1168, 1176,
transl:nnNTF 16, 413, 2406	1367, 1396, 1489, 1493, 1526, 1534,
_zrefclever_get_enclosing_-	1540, 1566, 1628, 1907, 2375, 2380,
counters:n 5, 37, 42, 84	2387, 2396, 2403, 2426, 2431, 2438
_zrefclever_get_enclosing_-	\l_zrefclever_label_type_b_tl ..
counters_value:n ... 5, 37, 51, 86 1067,
_zrefclever_get_fallback_-	1144, 1152, 1161, 1169, 1177, 1370,
transl:nN 431	1399, 1490, 1498, 1527, 1536, 1541
_zrefclever_get_fallback_-	_zrefclever_label_type_put_-
transl:nNTF 17, 429, 2410	new_right:n 34, 1081, 1103
_zrefclever_get_ref:n 1639, 1655,	\l_zrefclever_label_types_seq ..
1667, 1672, 1695, 1709, 1713, 1725, 34, 1088, 1091, 1096, 1099, 1394
1729, 1764, 1783, 1953, 1966, 1973,	_zrefclever_labels_in_sequence:nn
1997, 2009, 2013, 2023, 2027, 2043 42, 1755, 1894, 2446
_zrefclever_get_ref_first: ..	\g_zrefclever_languages_prop ...
..... 43, 55, 1776, 1834, 2216 11, 225, 230,
_zrefclever_get_ref_font:nN ...	232, 240, 243, 244, 255, 400, 416, 928
.... 8, 15, 27, 1577, 1579, 1581, 2421	\l_zrefclever_last_of_type_bool
_zrefclever_get_ref_string:nN 42, 1415, 1510, 1515, 1516,
..... 8, 9, 15, 27, 1046, 1464,	1520, 1529, 1544, 1548, 1554, 1605
1466, 1468, 1583, 1585, 1587, 1589,	\l_zrefclever_lastsep_tl . 1436,
1591, 1593, 1595, 1597, 1599, 2368	1592, 1654, 1671, 1694, 1712, 1724

\l_zrefclever_link_star_bool ...	_zrefclever_provide_dictionary_-
..... 1032, 1059, 2050, 2192, 2308	verbose:n 13,
\l_zrefclever_listsep_tl	303, 758, 766, 772, 794, 802, 808
... 1435, 1590, 1666, 1708, 1952,	\l_zrefclever_range_beg_label_-
1965, 1972, 1996, 2008, 2012, 2022	tl 43, 1424, 1457,
\l_zrefclever_load_dict_-	1667, 1690, 1696, 1706, 1710, 1722,
verbose_bool ... 251, 288, 297, 306	1726, 1876, 1914, 1929, 1963, 1967,
\g_zrefclever_loaded_dictionaries_-	1994, 1998, 2006, 2010, 2020, 2024
seq 250, 259, 282	\l_zrefclever_range_count_int ..
\l_zrefclever_main_language_tl 43,
. 24, 723, 730, 736, 740, 744, 757, 793	1424, 1460, 1647, 1681, 1879, 1915,
_zrefclever_name_default:	1926, 1931, 1937, 1945, 1986, 2032
..... 55, 2041, 2291	\l_zrefclever_range_same_count_-
\l_zrefclever_name_format_-	int 43,
fallback_tl 1445, 2121, 2125, 2127, 2163, 2175	1424, 1461, 1634, 1669, 1682, 1880,
\l_zrefclever_name_format_tl ...	1917, 1933, 1939, 1970, 1987, 2033
... 1445, 2107, 2108, 2111, 2112,	\l_zrefclever_rangesep_tl
2122, 2123, 2134, 2140, 2155, 2169 1433, 1586, 1728, 1763, 2026
\l_zrefclever_name_in_link_bool	_zrefclever_ref_default:
... 1445, 1791, 2196, 2212, 2213, 2221 55, 2039, 2090, 2219, 2285, 2364
\l_zrefclever_namefont_tl 1429,	\l_zrefclever_ref_language_tl ..
1578, 1794, 1822, 2250, 2281, 2296 24, 25,
\l_zrefclever_nameinlink_str ...	722, 743, 756, 759, 764, 767, 771,
..... 675, 680,	773, 783, 792, 795, 800, 803, 807,
682, 684, 686, 2194, 2200, 2202, 2206	809, 1033, 2138, 2167, 2173, 2402, 2407
\l_zrefclever_namesep_tl	\c_zrefclever_ref_options_font_-
... 1432, 1584, 2253, 2284, 2292, 2299	seq 10, 15, 171
\l_zrefclever_next_is_same_bool	\c_zrefclever_ref_options_-
..... 43, 63, 1424,	necessarily_not_type_specific_-
1888, 1916, 1932, 1938, 2466, 2492	seq 15, 171, 333, 886, 960
\l_zrefclever_next_maybe_range_-	\c_zrefclever_ref_options_-
bool 43, 63, 1424, 1751, 1761, 1887,	necessarily_type_specific_seq
1912, 1922, 2458, 2465, 2484, 2491 171, 364, 1004
\l_zrefclever_noabbrev_first_-	\c_zrefclever_ref_options_-
bool 709, 718, 2118	possibly_type_specific_seq ..
\l_zrefclever_noteseq_tl 15, 171, 350, 981
..... 1046, 1047, 1440	\l_zrefclever_ref_options_prop .
_zrefclever_page_format_aux: 27-
..... 90, 94	29, 61, 62, 855, 865, 866, 2371, 2423
\g_zrefclever_page_format_tl ...	\c_zrefclever_ref_options_-
..... 6, 89, 95, 98	reference_seq 171, 857
\l_zrefclever_pairsep_tl	\c_zrefclever_ref_options_-
..... 1434, 1588, 1638, 1762	typesetup_seq 171, 898
_zrefclever_prop_put_non_-	\l_zrefclever_ref_property_tl ..
empty:Nnn 17, 438, 455, 509 20,
_zrefclever_provide_dict_-	522, 527, 529, 535, 538, 554, 563,
default_transl:nn 14, 311, 341, 358	1100, 1133, 1487, 2045, 2069, 2083,
_zrefclever_provide_dict_type_-	2225, 2262, 2303, 2341, 2357, 2448
transl:nn 14, 311, 359, 376	\l_zrefclever_ref_typeset_font_-
_zrefclever_provide_dictionary:n	tl 816, 818, 1043
.... 9, 12-14, 33, 252, 307, 783, 1033	\l_zrefclever_reffont_in_tl 1431,
	1582, 2057, 2081, 2258, 2315, 2353
	\l_zrefclever_reffont_out_tl ...
 1430, 1580,

2054, 2078, 2255, 2275, 2312, 2350
 \l_zrefclever_refpos_in_tl 1444,
 1600, 2070, 2084, 2263, 2342, 2358
 \l_zrefclever_refpos_out_tl 1442,
 1596, 2073, 2086, 2276, 2345, 2360
 \l_zrefclever_refpre_in_tl 1443,
 1598, 2067, 2082, 2259, 2338, 2354
 \l_zrefclever_refpre_out_tl 1441,
 1594, 2055, 2079, 2256, 2313, 2351
 \l_zrefclever_setup_type_tl ...
 14, 169, 279, 315, 328,
 329, 340, 357, 371, 882, 910, 918,
 931, 955, 956, 967, 988, 997, 1011, 1019
 \l_zrefclever_sort_decided_bool
 ... 1079, 1225, 1226, 1245, 1256,
 1273, 1279, 1296, 1302, 1330, 1342
 _zrefclever_sort_default:nn ...
 34–36, 1135, 1140
 _zrefclever_sort_default_-
 different_types:nn . 21, 1184, 1350
 _zrefclever_sort_default_same_-
 type:nn 1180, 1206
 _zrefclever_sort_labels:
 34, 36, 41, 1041, 1097
 _zrefclever_sort_page:nn
 41, 1134, 1406
 \l_zrefclever_sort_prior_a_int .
 1077,
 1352, 1361, 1362, 1368, 1378, 1386
 \l_zrefclever_sort_prior_b_int .
 1078,
 1353, 1363, 1364, 1371, 1379, 1387
 \l_zrefclever_tlastsep_tl
 1439, 1469, 1865
 \l_zrefclever_tlistsep_tl
 1438, 1467, 1843
 \l_zrefclever_tpairsep_tl
 1437, 1465, 1859
 \l_zrefclever_type_<type>-
 options_prop 29
 \l_zrefclever_type_count_int ...
 42, 1422, 1459, 1840,
 1842, 1851, 1878, 2105, 2117, 2209
 \l_zrefclever_type_first_label_-
 tl 1417, 1455, 1625, 1743,
 1752, 1756, 1783, 1799, 1803, 1808,
 1814, 1874, 1904, 2095, 2218, 2224,
 2231, 2234, 2239, 2245, 2261, 2302,
 2320, 2323, 2328, 2334, 2340, 2356
 \l_zrefclever_type_first_label_-
 type_tl 1417, 1456, 1627,
 1747, 1875, 1906, 2098, 2132, 2139,
 2145, 2153, 2161, 2168, 2174, 2181
 _zrefclever_type_name_setup: ..
 8, 9, 43, 1771, 2093
 \l_zrefclever_type_name_tl . 56,
 1445, 1817, 1823, 2096, 2099, 2135,
 2141, 2143, 2156, 2164, 2170, 2176,
 2178, 2193, 2251, 2282, 2289, 2297
 \l_zrefclever_typeset_compress_-
 bool 618, 621, 1889
 \l_zrefclever_typeset_labels_-
 seq ... 1417, 1451, 1475, 1477, 1483
 \l_zrefclever_typeset_last_bool
 42, 1415,
 1472, 1473, 1480, 1509, 1848, 2208
 \l_zrefclever_typeset_name_bool
 569, 576, 581, 586, 1773, 1787
 \l_zrefclever_typeset_queue_-
 curr_tl 1417, 1454, 1636,
 1652, 1661, 1692, 1703, 1719, 1741,
 1759, 1775, 1782, 1789, 1833, 1855,
 1860, 1866, 1872, 1873, 1950, 1961,
 1992, 2004, 2018, 2110, 2203, 2207
 \l_zrefclever_typeset_queue_-
 prev_tl 1417, 1453, 1844, 1871
 \l_zrefclever_typeset_range_-
 bool 627, 630, 1040, 1739
 \l_zrefclever_typeset_ref_bool .
 568, 575, 580, 585, 1773, 1780
 _zrefclever_typeset_refs:
 42, 43, 55, 56, 58, 1044, 1449
 _zrefclever_typeset_refs_aux_-
 last_of_type: 1608, 1616
 _zrefclever_typeset_refs_aux_-
 not_last_of_type: . 42, 1612, 1883
 \l_zrefclever_typeset_sort_bool
 594, 597, 1039
 \l_zrefclever_typesort_seq
 21, 603, 608, 609, 615, 1357
 \l_zrefclever_use_hyperref_bool
 634, 641,
 646, 651, 661, 667, 2049, 2191, 2307
 \l_zrefclever_warn_hyperref_-
 bool 635, 642, 647, 652, 665
 _zrefclever_zcref:nnn .. 1026, 1027
 _zrefclever_zcref:nnnn 33, 35, 1027
 \l_zrefclever_zcref_labels_seq .
 35, 1031, 1055, 1059, 1102, 1105, 1452
 \l_zrefclever_zcref_note_tl ...
 819, 822, 1048
 \l_zrefclever_zcref_with_check_-
 bool 826, 841, 1036, 1051
 \l_zrefclever_zrefcheck_-
 available_bool
 825, 836, 847, 1035, 1050